# Table of Contents

## 1 Altova MapForce 2019 Enterprise Edition
1.1 What's new... ................................................................. 4

## 2 Introduction
2.1 Support Notes ........................................................... 15
2.2 What Is MapForce? ......................................................... 16
2.3 Basic Concepts ........................................................... 23
2.4 User Interface Overview ............................................... 25
2.5 Conventions ............................................................... 33

## 3 Tutorials
3.1 Convert XML to New Schema ......................................... 37
3.2 Map Multiple Sources to One Target ................................. 47
3.3 Work with Multiple Target Schemas ................................. 53
3.4 Process and Generate Files Dynamically .......................... 61

## 4 Common Tasks
4.1 Working with Mappings ................................................ 73
4.1.1 Adding Components to the Mapping ............................ 73
4.1.2 Adding Components from a URL ................................. 74
4.1.3 About Data Streaming .............................................. 77
4.1.4 Selecting a Transformation Language .......................... 78
4.1.5 Validating Mappings ................................................ 79
4.1.6 Validating the Mapping Output ................................... 81
4.1.7 Previewing the Output ............................................. 82
4.1.8 Text View Features ................................................ 83
4.1.9 Searching in Text View .......................................... 88
4.1.10 Previewing the XSLT Code ....................................... 92
4.1.11 Generating XSLT Code ......................................... 92
4.1.12 Previewing the XQuery Code ................................. 93
5 Designing Mappings

5.1 Using Relative and Absolute Paths ......................................................... 132
  5.1.1 Using Relative Paths on a Component .................................................. 132
  5.1.2 Setting the Path to File-Based Databases .............................................. 134
  5.1.3 Fixing Broken Path References ............................................................ 136
  5.1.4 Paths in Various Execution Environments .......................................... 137
  5.1.5 Copy-Paste and Relative Paths ............................................................ 138
5.2 Connection Types .................................................................................... 139
  5.2.1 Target-driven connections .................................................................... 139
  5.2.2 Source-driven connections ................................................................. 139
  5.2.3 Copy-All Connections ......................................................................... 147
5.3 Chained Mappings ................................................................................... 151
  5.3.1 Example: Pass-Through Active ............................................................ 153
5.3.2 Example: Pass-Through Inactive ................................................................. 157
5.4 Processing Multiple Input or Output Files Dynamically ..................................... 161
  5.4.1 Mapping Multiple Input Files to a Single Output File .................................. 163
  5.4.2 Mapping Multiple Input Files to Multiple Output Files ............................... 165
  5.4.3 Supplying File Names as Mapping Parameters ......................................... 166
  5.4.4 Previewing Multiple Output Files .............................................................. 166
  5.4.5 Example: Split One XML File into Many .................................................. 167
  5.4.6 Example: Split Database Table into Many XML Files ............................... 169
  5.4.7 Multiple XML files from Excel rows ......................................................... 171
5.5 Supplying Parameters to the Mapping .............................................................. 177
  5.5.1 Adding Simple Input Components ......................................................... 178
  5.5.2 Simple Input Component Settings ............................................................ 178
  5.5.3 Creating a Default Input Value ................................................................. 180
  5.5.4 Example: Using File Names as Mapping Parameters ............................... 181
5.6 Returning String Values from a Mapping .......................................................... 184
  5.6.1 Adding Simple Output Components ......................................................... 185
  5.6.2 Example: Previewing Function Output ..................................................... 186
5.7 Using Variables ................................................................................................. 188
  5.7.1 Adding Variables ...................................................................................... 190
  5.7.2 Changing the Context and Scope of Variables .......................................... 193
  5.7.3 Example: Counting Database Table Rows ............................................... 195
  5.7.4 Example: Filtering and Numbering Nodes ............................................... 196
  5.7.5 Example: Grouping and Subgrouping Records ......................................... 197
5.8 Sorting Data ....................................................................................................... 200
  5.8.1 Sorting by Multiple Keys .......................................................................... 202
  5.8.2 Sorting with Variables ............................................................................. 203
5.9 Grouping Data ................................................................................................... 206
  5.9.1 Example: Grouping Records by Key .......................................................... 209
5.10 Filters and Conditions ..................................................................................... 212
  5.10.1 Example: Filtering Nodes ......................................................................... 214
  5.10.2 Example: Returning a Value Conditionally ............................................ 216
5.11 Joining Data ..................................................................................................... 218
  5.11.1 Adding Join Conditions ........................................................................... 220
  5.11.2 Joining Three or More Structures .......................................................... 223
  5.11.3 Example: Join XML Structures .............................................................. 224
5.12 Using Value-Maps .......................................................................................... 230
  5.12.1 Passing data through a Value-Map unchanged ........................................ 233
5.12 Value-Map component properties ................................................................. 235
5.13 Adding Exceptions .......................................................................................... 238
  5.13.1 Example: Exception on "Greater Than" Condition .............................. 239
  5.13.2 Example: Exception When Node Does Not Exist ......................... 240
5.14 Parsing and Serializing Strings ..................................................................... 242
  5.14.1 About the Parse/Serialize Component .............................................. 242
  5.14.2 Example: Parse String (Fixed-Length Text to Excel) ....................... 244
  5.14.3 Example: Serialize to String (XML to Database) ......................... 249
5.15 Mapping Node Names .................................................................................... 254
  5.15.1 Getting Access to Node Names ......................................................... 255
  5.15.2 Accessing Nodes of Specific Type ................................................... 262
  5.15.3 Example: Map Element Names to Attribute Values .................... 266
  5.15.4 Example: Group and Filter Nodes by Name ..................................... 270
5.16 Mapping Rules and Strategies ...................................................................... 274
  5.16.1 Changing the Processing Order of Mapping Components ................ 278
  5.16.2 Priority Context node/item ................................................................. 281
  5.16.3 Overriding the Mapping Context ....................................................... 283

6 Debugging Mappings ......................................................................................... 290
  6.1 Debugger Preparation .................................................................................. 293
  6.2 Debugger Commands ................................................................................ 294
  6.3 About the Debug Mode .............................................................................. 296
  6.4 Adding and Removing Breakpoints ......................................................... 299
  6.5 Using the Values Window .......................................................................... 301
  6.6 Using the Context Window ...................................................................... 303
  6.7 Using the Breakpoints Window ................................................................. 305
  6.8 Previewing Partially Generated Output .................................................... 307
  6.9 Viewing the Current Value of a Connector ............................................. 308
  6.10 Stepping back into Recent Past .............................................................. 309
  6.11 Viewing the History of Values Processed by a Connector ................... 310
  6.12 Setting the Context to a Value ............................................................... 311
  6.13 Debugger Settings .................................................................................... 312

7 Data Sources and Targets ................................................................................ 314
  7.1 XML and XML schema ............................................................................. 315
    7.1.1 Generating an XML Schema .............................................................. 315
## 7.2 Databases and MapForce

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2.1 Connecting to a Database</td>
<td>348</td>
</tr>
<tr>
<td>7.2.2 Introduction to Database Mappings</td>
<td>424</td>
</tr>
<tr>
<td>7.2.3 Mapping Data to Databases</td>
<td>455</td>
</tr>
<tr>
<td>7.2.4 Joining Database Data</td>
<td>505</td>
</tr>
<tr>
<td>7.2.5 Filtering and Sorting Database Data (SQL WHERE/ORDER)</td>
<td>521</td>
</tr>
<tr>
<td>7.2.6 SQL SELECT Statements as Virtual Tables</td>
<td>527</td>
</tr>
<tr>
<td>7.2.7 Mapping XML Data to / from Database Fields</td>
<td>537</td>
</tr>
<tr>
<td>7.2.8 Browsing and Querying Databases</td>
<td>547</td>
</tr>
<tr>
<td>7.2.9 Stored Procedures</td>
<td>565</td>
</tr>
<tr>
<td>7.2.10 Handling Database Exceptions</td>
<td>589</td>
</tr>
<tr>
<td>7.2.11 Database Tracing and Error Logging</td>
<td>601</td>
</tr>
</tbody>
</table>

## 7.3 CSV and Text Files

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3.1 Example: Mapping CSV Files to XML</td>
<td>611</td>
</tr>
<tr>
<td>7.3.2 Example: Iterating Through Items</td>
<td>614</td>
</tr>
<tr>
<td>7.3.3 Example: Creating Hierarchies from CSV and Fixed-Length Text Files</td>
<td>616</td>
</tr>
<tr>
<td>7.3.4 Setting the CSV Options</td>
<td>619</td>
</tr>
<tr>
<td>7.3.5 Example: Mapping Fixed-Length Text Files to Databases</td>
<td>623</td>
</tr>
<tr>
<td>7.3.6 Setting the FLF Options</td>
<td>629</td>
</tr>
</tbody>
</table>

## 7.4 MapForce FlexText

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4.1 Overview</td>
<td>636</td>
</tr>
<tr>
<td>7.4.2 FlexText Tutorial</td>
<td>639</td>
</tr>
<tr>
<td>7.4.3 FlexText Component Settings</td>
<td>651</td>
</tr>
<tr>
<td>7.4.4 Using FlexText as a Target Component</td>
<td>653</td>
</tr>
<tr>
<td>7.4.5 FlexText Reference</td>
<td>654</td>
</tr>
<tr>
<td>7.4.6 FlexText and Regular Expressions</td>
<td>685</td>
</tr>
</tbody>
</table>
7.5 EDI .......................................................................................................................... 692
  7.5.1 EDI Terminology ................................................................................................. 694
  7.5.2 Adding EDI Components to the Mapping ......................................................... 695
  7.5.3 Running EDI Mappings ....................................................................................... 697
  7.5.4 EDI Component Settings ..................................................................................... 698
  7.5.5 EDI Component Validation .................................................................................. 702
  7.5.6 ASC X12 ............................................................................................................. 706
  7.5.7 HIPAA X12 ......................................................................................................... 710
  7.5.8 HL7 Version 2 ..................................................................................................... 715
  7.5.9 IATA PADIS ......................................................................................................... 718
  7.5.10 NCPDP SCRIPT ................................................................................................. 721
  7.5.11 SAP IDoc ........................................................................................................... 723
  7.5.12 TRADACOMS ..................................................................................................... 726
  7.5.13 UN/EDIFACT ..................................................................................................... 741
  7.5.14 Customizing EDI Structure ................................................................................ 753
  7.5.15 Customizing EDI Validation .............................................................................. 786

7.6 JSON .......................................................................................................................... 797
  7.6.1 Adding JSON Files as Mapping Components ..................................................... 799
  7.6.2 JSON Component Settings .................................................................................. 800
  7.6.3 JSON5 Support .................................................................................................... 802
  7.6.4 Example: Convert JSON to CSV ......................................................................... 803
  7.6.5 Example: Convert Excel to JSON .......................................................................... 808

7.7 Microsoft OOOXML Excel 2007+ ............................................................................ 813
  7.7.1 Adding Excel 2007+ Files as Mapping Components ........................................... 815
  7.7.2 About the Excel 2007+ Component ..................................................................... 816
  7.7.3 Adding and Removing Worksheets ...................................................................... 818
  7.7.4 Adding and Removing Row Ranges ..................................................................... 820
  7.7.5 Selecting Ranges of Cells ..................................................................................... 821
  7.7.6 Inserting Columns Between Existing Ones .......................................................... 825
  7.7.7 Excel 2007+ Component Settings ....................................................................... 827
  7.7.8 Example: Mapping Excel 2007+ to XML .............................................................. 829
  7.7.9 Example: Mapping Database Data to Excel 2007+ .............................................. 832
  7.7.10 Example: Supplying Data to Preformatted Excel Sheets .................................... 834

7.8 XBRL .......................................................................................................................... 838
  7.8.1 Adding XBRL Files as Mapping Components .................................................... 839
  7.8.2 Reading Data from Inline XBRL .......................................................................... 840
  7.8.3 XBRL Taxonomy Packages .................................................................................. 842
8 Functions

8.1 How To... ................................................................. 899
  8.1.1 Add a Built-in Function to the Mapping .................. 899
  8.1.2 Add a Constant to the Mapping ............................ 901
  8.1.3 Search for a Function ........................................ 902
  8.1.4 View a Function's Type and Description ............... 903
  8.1.5 Add or Delete Function Arguments ..................... 904

8.2 Defaults and Node Functions ..................................... 905
  8.2.1 Creating Defaults and Node Functions .................. 905
  8.2.2 Editing and Deleting Existing Rules ...................... 909
  8.2.3 How Defaults and Node Functions Work ................. 911
  8.2.4 Applying Node Functions and Defaults Conditionally ... 917
  8.2.5 Supplying Node Metadata to Node Functions ........... 922
  8.2.6 Example: Replace Empty CSV Fields .................... 926

8.3 User-Defined Functions ......................................... 932
  8.3.1 Creating User-Defined Functions ........................ 934
  8.3.2 Parameters in User-Defined Functions .................. 937
  8.3.3 Inline and Regular User-Defined Functions ............. 942
  8.3.4 Navigating User-Defined Functions ..................... 944
  8.3.5 Editing User-Defined Functions ........................... 944
  8.3.6 Deleting User-Defined Functions .......................... 945
  8.3.7 Calling and Importing User-Defined Functions .......... 946
8.3.8 Example: Look-up and Concatenation ................................................. 947
8.3.9 Example: Recursive Search ................................................................. 950
8.4 Importing Custom XSLT 1.0 or 2.0 Functions ....................................... 955
8.4.1 Example: Adding Custom XSLT Functions ......................................... 956
8.4.2 Example: Summing Node Values ....................................................... 959
8.5 Importing Custom XQuery 1.0 Functions ............................................. 962
8.5.1 Example: Import Custom XQuery Function ........................................ 962
8.6 Importing Custom Java and .NET Libraries .......................................... 967
8.6.1 Example: Import Custom Java Class .................................................. 969
8.6.2 Example: Import Custom .NET DLL Assembly ................................... 970
8.7 Referencing Java, C# and C++ Libraries Manually .................................. 972
8.7.1 Configuring the .mff File ................................................................. 972
8.7.2 Importing the .mff File Into MapForce .............................................. 978
8.7.3 Data Type Mapping ........................................................................... 978
8.7.4 Example: Create a Custom C# Library ............................................. 980
8.7.5 Example: Create a Custom C++ Library ............................................ 982
8.7.6 Example: Create a Custom Java Library .......................................... 984
8.8 Regular Expressions ............................................................................ 987
8.9 Function Library Reference ................................................................. 990
8.9.1 core | aggregate functions .............................................................. 990
8.9.2 core | conversion functions .............................................................. 995
8.9.3 core | file path functions .................................................................. 1006
8.9.4 core | generator functions ............................................................... 1009
8.9.5 core | logical functions ..................................................................... 1011
8.9.6 core | math functions ....................................................................... 1015
8.9.7 core | node functions ....................................................................... 1018
8.9.8 core | QName functions ...................................................................... 1021
8.9.9 core | sequence functions ................................................................... 1023
8.9.10 core | string functions ....................................................................... 1042
8.9.11 db ................................................................................................. 1052
8.9.12 edifact ............................................................................................ 1054
8.9.13 lang | QName functions ................................................................... 1056
8.9.14 lang | datetime functions ............................................................... 1057
8.9.15 lang | generator functions ............................................................... 1071
8.9.16 lang | logical functions ................................................................. 1072
8.9.17 lang | math functions ....................................................................... 1073
8.9.18 lang | string functions ....................................................................... 1077
9 Implementing SOAP Web Services  1108

9.1 SOAP/WSDL Support Notes ................................................................. 1111
9.2 Creating SOAP Web Service Projects .................................................. 1114
9.3 Defining SOAP Web Service Faults ....................................................... 1122
9.4 Java SOAP Web Services Specifics ....................................................... 1124
9.5 Example: Generating SOAP Web Services (C#) ................................... 1127
9.6 Example: Generating SOAP Web Services (Java) ................................. 1141

10 Calling Web Services  1150

10.1 Adding a Web Service Call (REST-Style) ........................................... 1152
  10.1.1 Defining Web Service Parameters ............................................... 1156
  10.1.2 Importing Web Service Parameters from URL ............................. 1161
  10.1.3 Importing Web Service Information from WADL ......................... 1162
  10.1.4 Defining the Request Structure .................................................... 1163
  10.1.5 Defining the Response Structure .................................................. 1164
  10.1.6 Handling the HTTP Response Conditionally .................................. 1166
10.2 Adding a Web Service Call (SOAP) ..................................................... 1169
10.3 Web Service Call Settings .................................................................... 1170
10.4 Setting HTTP Security ......................................................................... 1173
10.5 Setting WS-Security ............................................................................ 1175
10.6 Example: Calling a REST-Style Web Service ....................................... 1178
10.7 Example: Mapping Data from an RSS Feed .................................................. 1183
10.8 Example: Calling a SOAP Web Service ....................................................... 1188
10.9 Digital Certificate Management ................................................................. 1192
   10.9.1 Trusting Server Certificates on Linux .................................................. 1194
   10.9.2 Trusting Server Certificates on macOS .............................................. 1196
   10.9.3 Trusting Server Certificates on Windows .......................................... 1197
   10.9.4 Accessing the Certificate Stores on Windows .................................... 1198
   10.9.5 Exporting Certificates from Windows .............................................. 1200
   10.9.6 Client Certificates on Linux ............................................................... 1206
   10.9.7 Client Certificates on macOS ............................................................ 1207
   10.9.8 Client Certificates on Windows .......................................................... 1208

11 Automating Mappings and MapForce ......................................................... 1212
   11.1 Automation with RaptorXML Server .................................................. 1213
   11.2 Automation with MapForce Server ..................................................... 1214
   11.3 Preparing Mappings for Server Execution ........................................... 1215
   11.4 Compiling Mappings to MapForce Server Execution Files ................. 1221
   11.5 Deploying Mappings to FlowForce Server .......................................... 1223
   11.6 AS2 Integration ................................................................................. 1227
   11.7 MapForce Command Line Interface .................................................. 1233

12 Customizing MapForce .............................................................................. 1240
   12.1 Changing the MapForce Options .......................................................... 1241
       12.1.1 Java Settings .............................................................................. 1243
       12.1.2 Network Proxy Settings .............................................................. 1244
   12.2 Altova Global Resources ...................................................................... 1246
       12.2.1 Creating Global Resources ......................................................... 1246
       12.2.2 Databases as Global Resources .................................................. 1248
       12.2.3 MapForce and StyleVision Transformation Result as Global Resource ..... 1249
       12.2.4 The Global Resources XML File ............................................... 1250
       12.2.5 Global Resources in Various Execution Environments ............... 1250
       12.2.6 Example: Run Mapping with Variable Input Files ...................... 1252
       12.2.7 Example: Generate Output to Variable Folders ......................... 1254
       12.2.8 Example: Switch Databases ....................................................... 1255
       12.2.9 Example: Create an Application Workflow ............................... 1258
   12.3 Styling Mapping Output with StyleVision ......................................... 1263
12.3.1 Examples of Mappings with StyleVision Stylesheets ............................................. 1264
12.4 Generating and Customizing Mapping Documentation ........................................... 1268
  12.4.1 Predefined StyleVision Power Stylesheets ......................................................... 1274
  12.4.2 Custom Design ................................................................................................. 1277
12.5 Customizing Keyboard Shortcuts ............................................................................. 1279
12.6 Catalog Files ........................................................................................................... 1282

13 MapForce Plug-in for Visual Studio ................................................................. 1288
  13.1 Enabling the Plug-in ............................................................................................ 1289
  13.2 Working with Mappings and Projects .................................................................. 1291
  13.3 Accessing Common Menus and Functions ......................................................... 1293

14 MapForce Plug-in for Eclipse ............................................................................. 1298
  14.1 Installing the MapForce Plug-in for Eclipse ......................................................... 1299
  14.2 The MapForce Perspective .................................................................................. 1303
  14.3 Accessing Common Menus and Functions ......................................................... 1306
  14.4 Working with Mappings and Projects ................................................................. 1309
     14.4.1 Creating a MapForce/Eclipse Project ............................................................ 1309
     14.4.2 Creating New Mappings ................................................................................. 1311
     14.4.3 Importing Existing Mappings into an Eclipse Project ................................. 1313
     14.4.4 Configuring Automatic Build and Generation of MapForce Code .......... 1316
  14.5 Extending MapForce Plug-in for Eclipse ........................................................... 1319

15 Menu Reference ..................................................................................................... 1326
  15.1 File ....................................................................................................................... 1327
  15.2 Edit ........................................................................................................................ 1330
  15.3 Insert ...................................................................................................................... 1331
  15.4 Project ................................................................................................................... 1334
  15.5 Component ........................................................................................................... 1336
  15.6 Connection ........................................................................................................... 1338
  15.7 Function ................................................................................................................ 1339
  15.8 Output .................................................................................................................... 1340
  15.9 Debug ................................................................................................................... 1342
  15.10 View ..................................................................................................................... 1343
  15.11 Tools .................................................................................................................... 1345

Altova MapForce 2019 Enterprise Edition
16 Code Generator

16.1 Introduction to code generator ................................................................. 1355
16.2 What's new ............................................................................................... 1357
16.3 Generating C++ code ............................................................................. 1359
  16.3.1 Generating code from a mapping ....................................................... 1360
  16.3.2 Generating code from a mapping project ........................................... 1360
  16.3.3 Building the project ........................................................................... 1361
  16.3.4 Running the application ..................................................................... 1362
16.4 Generating C# code .................................................................................. 1363
  16.4.1 Generating code from a mapping ....................................................... 1364
  16.4.2 Generating code from a mapping project ......................................... 1364
  16.4.3 Building the project ........................................................................... 1365
  16.4.4 Running the application ..................................................................... 1365
16.5 Generating Java code ............................................................................... 1366
  16.5.1 Generating code from a mapping ....................................................... 1367
  16.5.2 Generating code from a mapping project ......................................... 1368
  16.5.3 Handling JDBC references ............................................................... 1368
  16.5.4 Building the project with Ant ........................................................... 1369
  16.5.5 Example: Build a Java application with Eclipse and Ant ............... 1370
16.6 Integrating MapForce-Generated Code .................................................. 1380
  16.6.1 Java example ..................................................................................... 1381
  16.6.2 C# example ....................................................................................... 1383
  16.6.3 C++ example ..................................................................................... 1385
  16.6.4 Changing the data type of the mapping input/output (C#, Java) ...... 1386
16.7 Generating Code from XML Schemas or DTDs ...................................... 1391
  16.7.1 About Schema Wrapper Libraries (C++) ......................................... 1393
  16.7.2 About Schema Wrapper Libraries (C#) .......................................... 1396
  16.7.3 About Schema Wrapper Libraries (Java) ........................................ 1398
  16.7.4 Integrating Schema Wrapper Libraries ......................................... 1400
  16.7.5 Example: Using the Schema Wrapper Libraries ............................. 1403
16.8 Reference to Generated Classes (C++) ................................................. 1428
  16.8.1 altova::DateTime .............................................................................. 1428
  16.8.2 altova::Duration .............................................................................. 1431
  16.8.3 altova::DayTimeDuration ............................................................... 1433
16.8.4 altova::YearMonthDuration ............................................................... 1434
16.8.5 altova::meta::Attribute ................................................................. 1435
16.8.6 altova::meta::ComplexType .......................................................... 1435
16.8.7 altova::meta::Element ................................................................. 1436
16.8.8 altova::meta::SimpleType .............................................................. 1437
16.8.9 [YourSchema].[CDoc] ................................................................. 1438
16.8.10 [YourSchema].:MemberAttribute ............................................. 1440
16.8.11 [YourSchema].:MemberElement .............................................. 1441

16.9 Reference to Generated Classes (C#) ........................................... 1442
16.9.1 Altova.Types.DateTime ................................................................. 1442
16.9.2 Altova.Types.DateTimeFormat .................................................... 1446
16.9.3 Altova.Types.Duration ............................................................... 1447
16.9.4 Altova.Xml.Meta.Attribute ........................................................ 1449
16.9.5 Altova.Xml.Meta.ComplexType .................................................. 1450
16.9.6 Altova.Xml.Meta.Element .......................................................... 1451
16.9.7 Altova.Xml.Meta.SimpleType ..................................................... 1451
16.9.8 [YourSchema],[Doc] ................................................................. 1452
16.9.9 [YourSchemaType].:MemberAttribute ....................................... 1454
16.9.10 [YourSchemaType].:MemberElement ...................................... 1455

16.10 Reference to Generated Classes (Java) ....................................... 1457
16.10.1 com.altova.types.DateTime ...................................................... 1457
16.10.2 com.altova.types.Duration ...................................................... 1462
16.10.3 com.altova.xml.meta.Attribute .............................................. 1465
16.10.4 com.altova.xml.meta.ComplexType ....................................... 1466
16.10.5 com.altova.xml.meta.Element ................................................. 1466
16.10.6 com.altova.xml.meta.SimpleType ......................................... 1467
16.10.7 com.[YourSchema],[Doc] ......................................................... 1468
16.10.8 com.[YourSchema],[YourSchemaType].:MemberAttribute .......... 1470
16.10.9 com.[YourSchema],[YourSchemaType].:MemberElement .......... 1470

16.11 Code Generation Tips ................................................................. 1472

16.12 Code Generator Options .............................................................. 1473

16.13 SPL (Spy Programming Language) .............................................. 1475
16.13.1 Basic SPL structure ............................................................... 1475
16.13.2 Declarations ......................................................................... 1476
16.13.3 Variables .............................................................................. 1478
16.13.4 Predefined variables ............................................................. 1479
16.13.5 Creating output files ............................................................. 1480
16.13.6 Operators ................................................................. 1482
16.13.7 Conditions ............................................................. 1482
16.13.8 Collections and foreach ........................................... 1483
16.13.9 Subroutines ............................................................ 1485
16.13.10 Built in Types ......................................................... 1487

17 The MapForce API ....................................................... 1494
17.1 Overview ........................................................................ 1495
  17.1.1 Accessing the API .................................................... 1495
  17.1.2 The Object Model ................................................... 1497
  17.1.3 Error Handling ....................................................... 1498
  17.1.4 Examples ............................................................. 1499
17.2 Object Reference ........................................................ 1522
  17.2.1 Interfaces .............................................................. 1522
  17.2.2 Enumerations ........................................................ 1654

18 ActiveX Integration ...................................................... 1664
18.1 Prerequisites .................................................................. 1665
18.2 Adding the ActiveX Controls to the Toolbox .................... 1667
18.3 Integration at Application Level ..................................... 1669
18.4 Integration at Document Level ...................................... 1672
18.5 ActiveX Integration Examples ....................................... 1676
  18.5.1 C# ......................................................................... 1676
  18.5.2 HTML ..................................................................... 1683
  18.5.3 Java ....................................................................... 1690
  18.5.4 VB.NET ................................................................. 1700
18.6 Command Reference .................................................. 1703
  18.6.1 "File" Menu .......................................................... 1703
  18.6.2 "Edit" Menu ........................................................ 1704
  18.6.3 "Insert" Menu ....................................................... 1705
  18.6.4 "Project" Menu ...................................................... 1705
  18.6.5 "Component" Menu .............................................. 1706
  18.6.6 "Connection" Menu ............................................... 1708
  18.6.7 "Function" Menu .................................................. 1708
  18.6.8 "Output" Menu ..................................................... 1708
  18.6.9 "Debug" Menu ...................................................... 1709
Index
1 Altova MapForce 2019 Enterprise Edition

MapForce® 2019 Enterprise Edition is a visual data mapping tool for advanced data integration projects. MapForce® is a 32/64-bit Windows application that runs on Windows 7 SP1 with Platform Update, Windows 8, Windows 10, and Windows Server 2008 R2 SP1 with Platform Update or newer. 64-bit support is available for the Enterprise and Professional editions. MapForce also integrates with Visual Studio and Eclipse, as well as Microsoft Office products, see Support Notes.

Last updated: 28 March 2019
1.1 What’s new...

New in MapForce 2019 Release 3:

- Major parts of the graphical user interface are now optimized for monitors with high pixel density (HiDPI)
- Support for Database Tracing and Error Logging
- Better support for database transaction handling in generated program code and server execution. If an error occurs when a mapping updates a database, it is possible to continue the mapping execution even if some operations failed, see Handling Database Exceptions.
- Web service enhancements, including the following:
  - Ability to map the HTTP status code returned by the Web service (applicable for RESTful Web services)
  - Ability to supply the username, password, and URL of the Web service from the mapping (applies both to WSDL-style and RESTful Web services), see Web Service Call Settings
  - Ability to create the raw body (MIME entity) of the Web service from the mapping, or map the raw body returned by the Web service call, with the help of new mime functions. Applicable for RESTful Web services.
  - Ability to handle the Web service response conditionally, depending on the HTTP status code and content type (for RESTful Web services), see Handling the HTTP Response Conditionally.
- A mapping can encode or decode Base64 data with the help of the new charset-encode and charset-decode functions.
- Support for UN/EDIFACT D.18B Directory, see EDI.
- New database versions are supported: PostgreSQL 11, Db2 for i 7.2 and 7.3
- Support for Eclipse 4.7 - 4.10, see MapForce Plug-in for Eclipse
- Support for explicitly setting the Java Virtual Machine path from MapForce, see Java Settings
- Internal updates and optimizations

New in MapForce 2019:

- MapForce (and MapForce Server) can now map data to or from binary files in Protocol Buffers format, see Protocol Buffers.
- New database versions are supported: Firebird 3.0, Informix 12.10, MariaDB 10.2, MySQL 8.0, PostgreSQL 10, SQL Server 2017, SQL Server for Linux (adds to support for previous versions)
- You can selectively apply a default value or node function to specific nodes that meet your custom-defined criteria. For example, you can apply a node function only to nodes of specific type whose name matches a regular expression, see Applying Node Functions and Defaults Conditionally.
- When creating node functions, you can process data conditionally based on metadata such as node name or node annotation, see Supplying Node Metadata to Node Functions.
- Mappings that contain grouping functions run much faster in generated C#, C++, or Java code
• Internal updates and optimizations

New in MapForce 2018 Release 2:

• You can apply default values and functions not only to a single item, but also to multiple
descendant items in a structure, without copy-pasting the function. This greatly
simplifies the mapping and provides new ways to solve miscellaneous mapping
requirements, see Defaults and Node Functions.
• Mappings that include JSON components can be run not only by the BUILT-IN
transformation engine, but also with generated code (C# and Java)
• Support for the following databases: MariaDB 10.2, Teradata 16
• Built-in functions, user-defined functions, and constants can be conveniently added to
the mapping by double-clicking an empty area on the mapping (see Add a Built-in
Function to the Mapping and Add a Constant to the Mapping)
• Internal updates and optimizations

New in MapForce 2018:

• When FlowForce Server Advanced Edition is licensed alongside MapForce and
MapForce Server, you can create data workflows that automate the AS2 process, see
AS2 Integration.
• Support for NCPDP (National Council for Prescription Drug Programs) SCRIPT standard,
see NCPDP SCRIPT.
• Support for generating program code for Visual Studio 2013, 2015, and 2017, see Code
Generator
• Support for the following database versions: Sybase ASE 16, PostgreSQL 9.6, MySQL
5.7
• When mapping data to Microsoft Excel, it is possible to add, directly from the mapping,
new Excel columns in between existing columns, without redoing any existing
connections, see Inserting Columns Between Existing Ones.
• Internal updates and optimizations

New in MapForce 2017 Release 3:

• A new component type (Join) has been introduced which can be used to join data from
two or more different structures based on custom-defined conditions (see Joining Data).
When the mapping reads data from a database, it is possible to join database tables or
views in SQL JOIN mode, see Joining Database Data.
• Support is available for mapping data from JSON files which use JSON5 syntax. It is
also possible to generate JSON5 files with unquoted object keys (see JSON5 Support).
• The text search options in the Output pane, the XQuery pane, as well as the XSLT
pane have been enhanced (see Searching in Text View). Also, text highlighting is
available in the above-mentioned panes (see Text Highlighting).
• Mappings which update databases can be optionally configured to compare data in a
NULL-aware manner. NULL-aware comparisons provide a better way (tailored to each
specific database) to handle data that contains null values (see Handling Nulls in
Database Table Actions).
In the MapForce ActiveX control, the structure of the MapForceCommand object has been enhanced to include a new Name property, which can be used to get the unique name of the command. This simplifies retrieving information about MapForce commands programmatically (see Retrieving Command Information).

Internal updates and optimizations

New in MapForce 2017:

- It is possible to read node names from a source XML (or field names from a CSV/Fixed-length field file) and map this information to a target. It is also possible to dynamically create new XML attributes or elements in a target based on values supplied from a source. See Mapping Node Names.
- XML instance files can be created with custom namespaces, at element level (see Declaring Custom Namespaces).
- Support for XBRL taxonomy packages.
- Support for reading inline XBRL.
- Support for EDIFACT D.15B and D.16A directories (see EDI). The D.16A directory includes support for the Verification of Mass (VERMAS) message.
- Validation of EDI messages can be customized through configuration files. This makes it possible to relax the validation rules when necessary or adapt to MapForce external code values referenced by the EDI standard (see Customizing EDI Validation Through Configuration Files).
- Debugging of mappings is also available in the MapForce Plug-in for Visual Studio and MapForce Plug-in for Eclipse.
- MapForce Server execution files (.mfx) can be compiled for specific MapForce Server versions (see Compiling mappings for a specific MapForce Server version).
- Mappings can connect to PostgreSQL databases through native connections (see Setting up a PostgreSQL Connection).
- Mappings can connect to SQL Server and other database types through ADO.NET providers (see Setting up an ADO.NET Connection).
- A new database type is supported: Progress OpenEdge. See Connecting to Progress OpenEdge (ODBC) and Connecting to Progress OpenEdge (JDBC).
- When connecting to a database through JDBC, the search path to .jar libraries can be specified directly in the database connection dialog box (see Setting up a JDBC Connection).
- When a database is updated by the mapping through "Update if... Insert Rest" actions, MERGE statements are created for selected databases (see MERGE statements).
- Internal updates and optimizations

New in MapForce 2016 R2:

- More intuitive code folding in the XSLT pane: collapsed text is displayed with an ellipsis symbol and can be previewed as a tooltip. The same rules apply for text in the XQuery pane and the SQL Editor.
- You can search for all occurrences of a function within the active mapping (in the Libraries window, right-click the function, and select Find All Calls).
- You can call REST-style Web services from a mapping, see Adding a Web Service Call (REST-style). This adds to existing support for WSDL-style Web services.
- Support for the EDIFACT 2015A directory (see EDI).
• Support for Eclipse 4.5 (see MapForce Plug-in for Eclipse)
• Internal updates and optimizations

New features in MapForce 2016:

• Improved generation of XSLT 1.0 code (generated stylesheets are easier to read and often faster to execute)
• Two new aggregate functions are available in the MapForce core library: min-string and max-string. These functions enable you to get the minimum or maximum value from a sequence of strings.
• Mappings written for the Built-in execution engine can be debugged (see Debugging Mappings)
• The MapForce Plug-in for Visual Studio supports Visual Studio 2015 (adds to support for previous versions)
• New database versions are supported: SQL Server 2014, Oracle 12c, IBM DB2 10.5, PostgreSQL 9.4, MySQL 5.6 (adds to support for previous versions)
• Firebird databases are supported (see Connecting to Firebird (ODBC) and Connecting to Firebird (JDBC))

New features in MapForce Version 2015 R4:

• Support for mapping data to or from TRADACOMS format
• In the MapForce plug-in for Eclipse, the commands specific to MapForce files are now available under a new MapForce menu (see Accessing Common Menus and Functions)
• Internal updates and optimizations

New features in MapForce Version 2015 R3 include:

• Option to suppress the <\?xml \ldots \?> declaration in XML output
• When calling Web services, you can use preemptive HTTP authentication, HTTPS, and basic WS-Security (see Setting HTTP Security and Setting WS-Security)
• Support for reading Strict Open XML Excel 2013 files
• FlexText split and switch using regular expressions (see FlexText and Regular Expressions)
• Text-based components (including EDI, CSV, fixed-length field, JSON, and XML) can parse and serialize strings in addition to plain files
• SQLite database support (see Setting up a SQLite connection)
• New string padding functions: pad-string-left and pad-string-right
• New component type: Simple Output
• Internal updates and optimizations

New features in MapForce Version 2015 include:

• New language argument available in the format-date and format-dateTime functions
• New sequence function: replicate-item
• XBRL table linkbase support
• New mode for FlexText "Split Once" option: Delimited (line starts with)
• Map to or from JSON files
New features in MapForce Version 2014 R2 include:

- New **sequence functions**: generate sequence, item-at, etc.
- Ability to define **CDATA** sections in output components
- Ability to define timeout values for **database execution** and **Web service calls**
- **Keeping** connections after deleting components
- **Bulk transfer** of database data (bulk Insert all)
- Automatic highlighting of **mandatory items** in target components

New features in MapForce Version 2014 include:

- Integration of RaptorXML validator and basic support for **XML Schema 1.1**
- Integration of new RaptorXML XSLT and XQuery engines
- **XML Schema Wildcard support**, xs:any and xs:anyAttribute
- Support for **Comments and Processing Instructions** in XML target components
- **Age** function
- Ability to always insert **quote character** for CSV files

New features in MapForce Version 2013 R2 SP1 include:

- New super-fast transformation engine **RaptorXML Server**

New features in MapForce Version 2013 R2 include:

- **MapForce Server** support.
- Ability to generate a **MapForce Server execution** file from the command line and File menu, to be executed by MapForce Server.
- Ability to **deploy** MapForce mappings to FlowForce Server.
- Support for Informix 11.7 databases, and **extended support** for other databases.
- User defined **end-of-line** settings for output files.
- Internal updates and optimizations.

New features in MapForce Version 2013 include:

- Ability to call **stored procedures** in mappings
- Support for database functions (functionally similar to stored procedures)
- Support for **SELECT statements** with parameters
- Internal updates and optimizations

New features in MapForce Version 2012 R2 include:

- New **Sort component** for XSLT 2.0, XQuery, and the Built-in execution engine
- User defined component names
- Extended SQL-Where functionality: **ORDER BY**
- MapForce supports logical files of the IBM iSeries database and shows logical files as views
- Support for IBM DB2 logical files. A logical file in IBM iSeries editions of the DB2
database represents one or more physical files. A logical file allows users to access data in a sequence or format that can be different from the physical file. Users who connect to IBM iSeries computers may encounter existing databases constructed with logical files. These were previously not accessible, but are now supported in Version 2012 Release 2.

New features in MapForce Version 2012 include:

- Data streaming for XML, CSV and fixed-length field files (when using the built-in execution engine)
- New database engine supports direct ODBC and JDBC connections
- Auto-alignment of components in the mapping window
- New functions: parse-date and parse-time
- Find items in Project tab/window
- Prompt to connect to target parent node
- Specific rules governing the sequence that components are processed in a mapping
- New Programming Languages examples section in MapForce API

New features in MapForce Version 2011R3 include:

- Intermediate variables
- Support for multiple row ranges in Excel components
- Support for EDI X12 - HIPAA Implementation Guides
- Generation of X12 EDI 999 Implementation Acknowledgement component
- XML Digital signatures
- Support for US-GAAP 2011
- Support for .NET Framework 4.0 assembly files
- Ability to output StyleVision formatted documents from the command line

New features in MapForce Version 2011R2 include:

- Built-in Execution Engine now supports streaming output
- Find function capability in Library window
- Reverse mapping
- Extendable IF-ELSE function
- Node Name and parsing functions in Core Library
- New EDI format IATA PADIS
- Ability to process multiple EDI messages per component
- Improved database table actions dialog with integrated key generation settings
- New option of using StyleVision Power Stylesheets when documenting a mapping

New features in MapForce Version 2011 include:

- Ability to preview intermediate components in a mapping chain of two or more components connected to a target component (pass-through preview).
- Formatting functions for dateTime and numbers for all supported languages
- Enhancement to auto-number function
- New timezone functions: remove-timezone and convert-to-utc
- Ability to preview target components using StyleVision Power Stylesheets containing StyleVision Charts
New features in MapForce Version 2010 Release 3 include:

- Support for generation of Visual Studio 2010 project files for C# and C++ added
- Ability to define a worksheet row as column names in an Excel component
- Support for MSXML 6.0 in generated C++ code
- Support for Nullable values, and xsi:nil attribute in XML instance files
- Ability to disable automatic casting to target types in XML documents
- Support for SAP IDocs

New features in MapForce Version 2010 Release 2 include:

- 64-bit MapForce Enterprise / Professional editions on 64-bit operating systems
- Support for Excel 2010
- Automatic connection of identical child connections when moving a parent connection
- Support for fields in the SQL Where component
- Ability to add compiled Java .class and .NET assembly files
- Ability to tokenize input strings for further processing
- UN/EDIFACT and ANSI X12 EDI source file validation
- Generation of X12 EDI 997 Functional Acknowledgement component

New features in MapForce Version 2010 include:

- Multiple input/output files per component
- Upgraded relative path support
- xsi:type support allowing use of derived types
- New internal data type system
- Improved user-defined function navigation (see Navigating User-Defined Functions)
- Validation of EDI output in generated code (see EDI Component Validation)
- Support of EDIFACT service messages CONTRL, AUTACK and KEYMAN
- Support for Web services defined using WSDL 2.0
- Enhanced handling of mixed content in XML elements

New features in MapForce Version 2009 SP1 include:

- Parameter order in user-defined functions can be user-defined (see Parameters in User-Defined Functions)
- Ability to process XML files that are not valid against XML Schema
- Regular (standard) user-defined functions now support complex hierarchical parameters (see Inline and Regular User-Defined Functions)
- Apache Xerces 3.x support when generating C++ code

New features in MapForce Version 2009 include:

- Support for XBRL and XBRL dimension instance files, as well as XBRL taxonomies as source and target components
- EDI HL7 versions 2.2 to 2.6 components as source and target components
- EDI HL7 versions 3.x XML as source and target components
- Documentation of mapping projects
• Native support for XML fields in SQL Server
• Grouping of nodes or node content
• Ability to filter data based on a nodes position in a sequence
• QName support
• Item/node search in components

New features in MapForce Version 2008 Release 2 include:

• Ability to automatically generate XML Schemas for XML files
• Office Open XML Excel 2007 and higher (*.xlsx) support as source and target components
• Support for stream objects as input/output in generated Java and C# code
• Generation of Visual Studio 2008 project files for C++ and C#
• Support for SOAP version 1.2 in Web services
• New Repeated split option " Starts with... " in FlexText
• Ability to strip database schema names from generated code
• SQL SELECT Statements as virtual tables in database components
• Local Relations - on-the-fly creation of primary/foreign key relationships
• Support for Altova Global Resources
• Performance optimizations

New features in MapForce Version 2008 include:

• Aggregate functions
• Value-Map lookup component
• Enhanced XML output options: pretty print XML output, omit XML schema reference and Encoding settings for individual components
• Various internal updates

New features in MapForce Version 2007 Release 3 include:

• XML data mapping to/from database fields (see Mapping XML Data to / from Database Fields )
• Direct querying of databases
• SQL-WHERE filter and SQL statement wizard
• Code generator optimization and improved documentation
• Full support for all EDI X12 releases from 3040 to 5030
• Full support for UN/EDIFACT messages of directories 93A to 06B
Chapter 2

Introduction
2 Introduction

This introduction includes an overview of the MapForce features and user interface, the basic concepts in MapForce, as well as the conventions used in this documentation.
2.1 Support Notes

MapForce® is a 32/64-bit Windows application that runs on the following operating systems:

- Windows 7 SP1 with Platform Update, Windows 8, Windows 10
- Windows Server 2008 R2 SP1 with Platform Update or newer

64-bit support is available for the Enterprise and Professional editions.

MapForce is optionally available as a plug-in to the following integrated development environments:

- Visual Studio 2010/2012/2013/2015/2017, see MapForce Plug-in for Visual Studio
- Eclipse 4.7 / 4.8 / 4.9 / 4.10, see MapForce Plug-in for Eclipse.

MapForce integrates with Microsoft Office products as follows:

- It can map data to or from Access databases. For supported versions, see Databases and MapForce
- It can generate mapping documentation in Word 2000 or later format, see Generating and Customizing Mapping Documentation.
- It can map data to or from Excel 2007 or later spreadsheets, see Microsoft OOOXML Excel 2007+

For support information applicable to program code generation, see Introduction to Code Generator.

For other technical information, see Technical Data.
2.2 What Is MapForce?

Altova website: Data mapping tool

MapForce is a Windows-based, multi-purpose IDE (integrated development environment) that enables you to transform data from one format to another, or from one schema to another, by means of a visual, "drag-and-drop"-style graphical user interface that does not require writing any program code. In fact, MapForce generates for you the program code which performs the actual data transformation (or data mapping). When you prefer not to generate program code, you can just run the transformation using the MapForce built-in transformation language (available in the MapForce Professional or Enterprise Editions).

Any data mapping designed for the built-in transformation language qualifies for automation with MapForce Server. With MapForce Server, you can run mappings on Windows, Linux, and macOS not only from the command-line, but also through API calls (.NET, Java, COM), or as scheduled or recurring jobs under the management of FlowForce Server. For more information, see Automation with MapForce Server.

Mappings designed with MapForce enable you to conveniently convert and transform data from and to a variety of file-based and other formats. Regardless of the technology you work with, MapForce determines automatically the structure of your data, or gives you the option to supply a schema for your data, or generate it automatically from a sample instance file. For example, if you have an XML instance file but no schema definition, MapForce can generate it for you, thus making the data inside the XML file available for mapping to other files or formats.

The technologies supported as mapping sources or targets are as follows.
What Is MapForce?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• XML and XML schema</td>
<td>• XML and XML schema</td>
<td>• XML and XML schema</td>
</tr>
<tr>
<td>• HL7 version 3.x (schema-based)</td>
<td>• Flat files, including comma-separated values (CSV) and fixed-length field (FLF) format</td>
<td>• Flat files, including comma-separated values (CSV) and fixed-length field (FLF) format</td>
</tr>
<tr>
<td></td>
<td>• Databases (all major relational databases, including Microsoft Access and SQLite databases)</td>
<td>• Databases (all major relational databases, including Microsoft Access and SQLite databases)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Data from legacy text files can be mapped and converted to other formats with MapForce FlexText</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• EDI family of formats (including UN/EDIFACT, ANSI X12, HL7, IATA PADIS, SAP IDoc, TRADACOMS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• JSON files</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Microsoft Excel 2007 and later files</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• XBRL instance files and taxonomies</td>
</tr>
</tbody>
</table>

Based on the MapForce edition, you can choose the preferred language for your data transformation as follows.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• XSLT 1.0</td>
<td>• MapForce built-In transformation language</td>
<td>• MapForce built-In transformation language</td>
</tr>
<tr>
<td>• XSLT 2.0</td>
<td>• XSLT 1.0</td>
<td>• XSLT 1.0</td>
</tr>
<tr>
<td></td>
<td>• XSLT 2.0</td>
<td>• XSLT 2.0</td>
</tr>
<tr>
<td></td>
<td>• XQuery</td>
<td>• XQuery</td>
</tr>
<tr>
<td></td>
<td>• Java</td>
<td>• Java</td>
</tr>
<tr>
<td></td>
<td>• C#</td>
<td>• C#</td>
</tr>
<tr>
<td></td>
<td>• C++</td>
<td>• C++</td>
</tr>
</tbody>
</table>

You can preview the result of all transformations, as well as the generated XSLT or XQuery code without leaving the graphical user interface. Note that, as you design or preview mappings, MapForce validates the integrity of your schemas or transformations and displays any validation errors in a dedicated window, so that you can immediately review and address them.

When you choose Java, C#, or C++ as transformation language, MapForce generates the required projects and solutions so that you can open them directly in Visual Studio or Eclipse, and run the generated data mapping program. For advanced data integration scenarios, you can also extend the generated program with your own code, using Altova libraries and the MapForce API.
In MapForce, you design all mapping transformations visually. For example, in case of XML, you can connect any element, attribute, or comment in an XML file to an element or attribute of another XML file, thus instructing MapForce to read data from the source element (or attribute), and write it to the target element (or attribute).

Likewise, when working with databases in MapForce Professional or Enterprise Editions, you can see any database column in the MapForce mapping area and map data to or from it by making visual connections. As with other Altova MissionKit products, when setting up a database connection from MapForce, you can flexibly choose the database driver and the connection type (ADO, ADO.NET, ODBC, or JDBC) according to your existing infrastructure and data mapping needs. Additionally, you can visually build SQL queries, use stored procedures, or query a database directly (support varies by database type, edition and driver).
Introduction

What Is MapForce?

Altova MapForce 2019 Enterprise Edition

In a very simple scenario, a mapping design created with MapForce could be resumed as “read data from X and write it to Y”. However, you can easily design MapForce scenarios such as “read data from X and write it to Y, and then read data from Y and write it to Z”. These are known as “pass-through”, or “chained” mappings, and enable you to access your data at an intermediary stage in the transformation process (in order to save it to a file, for example).

Note that the data mappings you can create in MapForce are not limited to single, predefined files. In the same transformation, you can process dynamically multiple input files from a directory and generate multiple output files. Therefore, you can have scenarios such as “read data from multiple X files and write it to a single Y file”, or “read file X and generate multiple files Y”, and so on.

Importantly, in the same transformation, you can mix multiple sources and multiple targets, which can be of any type supported by your MapForce edition. For example, in case of MapForce Professional or Enterprise, this makes it possible to merge data from two different databases into a single XML file. Or, you can merge data from multiple XML files, and write some of the data to one database, and some of the data to another database. You can preview the SQL statements before committing them to the database.

Direct conversion of data from a source to a target is not typically the only thing you want to
achieve. In many cases, you might want to process your data in a particular way (for example, sort, group or filter it) before it reaches the destination. For this reason, MapForce includes, on one hand, miscellaneous functional components that are simplified programming language constructs (such as constants, variables, SQL-WHERE conditions, Filter and Sort components). On the other hand, MapForce includes rich and extensible function libraries which can assist you with virtually any kind of data manipulation.

If necessary, you can extend the built-in library either with functions you design in MapForce directly (the so-called User-Defined Functions, or UDF), or with functions or libraries created externally in XSLT, XQuery, Java, or C# languages.
projects (available in MapForce Professional and Enterprise edition). This allows for easier access and management. Importantly, you can generate program code from entire projects, in addition to generating code for individual mappings within the project.

For advanced data processing needs (such as when running mapping transformations with the MapForce Server API), you can design a mapping so that you can pass values to it at run-time, or get a simple string value from it at run-time. This feature also enables you to quickly test the output of functions or entire mappings that produce a simple string value. The Professional and Enterprise editions of MapForce also include components that enable you to perform run-time string parsing and serialization, similar to how this works in many other programming languages.

With MapForce Enterprise Edition, you can visually design SOAP 1.0 and SOAP 2.0 Web services based on Web Service Language Definition (WSDL) files, see Implementing SOAP Web Services. You can also call WSDL-style or REST-style Web services from within a mapping. For more information, see Calling Web Services.

With MapForce Professional and Enterprise Editions, you can generate detailed documentation of your mapping design files, in HTML, Word, and RTF formats. Documentation design can be customized (for example, you can choose to include or exclude specific components from the documentation).

If you are using MapForce alongside other Altova MissionKit products, MapForce integrates with them as well as with the Altova server-based products, as shown in the following table.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>You can choose to run the generated XSLT directly in MapForce and preview the data transformation result immediately. When you need increased performance, you can process the mapping using RaptorXML Server, an ultra-fast XML transformation engine.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| If XMLSpy is installed on the same machine, you can conveniently open and edit any supported file types, by opening XMLSpy directly from the relevant MapForce contexts (for example, the Component | Edit Schema Definition in XMLSpy menu command is available when you click an XML component). | | |
|-----------------|-----------------|---------------------------|
| You can run data transformations either directly in MapForce, or deploy them to a different machine and even operating system for command-line or automated execution. More specifically, you can design mappings on Windows, and run them on a Windows, Linux, or Mac server machine which runs MapForce Server (either standalone or under FlowForce Server management). | | |
| If StyleVision is installed on the same machine, you can design or reuse existing StyleVision Power Stylesheets and preview the result of the mapping transformations as HTML, RTF, PDF, or Word 2007+ documents. | | |

MapForce Professional and Enterprise edition can be installed as a plug-in of Visual Studio and Eclipse integrated development environments. This way, you can design mappings and get access to MapForce functionality without leaving your preferred development environment.
In MapForce, you can completely customize not only the look and feel of the development environment (graphical user interface), but also various other settings pertaining to each technology and to each mapping component type, for example:

- When mapping to or from XML, you can choose whether to include a schema reference, or whether the XML declaration must be suppressed in the output XML files. You can also choose the encoding of the generated files (for example, UTF-8).
- When mapping to or from databases, you can define settings such as the time-out period for executing database statements, whether MapForce should use database transactions, or whether it should strip the database schema name from table names when generating code.
- In case of XBRL, you can select the structure views MapForce should display (such as the "Presentation and definition linkbases" view, the "Table Linkbase" View, or the "All concepts" view).

All editions of MapForce are available as a 32-bit application. The MapForce Professional and Enterprise editions are additionally available as a 64-bit application.
2.3 Basic Concepts

This section outlines the basic concepts that will help you get started with data mapping.

Mapping
A MapForce mapping design (or simply "mapping") is the visual representation of how data is to be transformed from one format to another. A mapping consists of components that you add to the MapForce mapping area in order to create your data transformations (for example, convert XML documents from one schema to another). A valid mapping consists of one or several source components connected to one or several target components. You can run a mapping and preview its result directly in MapForce. You can generate code and execute it externally. You can also compile a mapping to a MapForce execution file and automate mapping execution using MapForce Server or FlowForce Server. MapForce saves mappings as files with .mfd extension.

Component
In MapForce, the term "component" is what represents visually the structure (schema) of your...
data, or how data is to be transformed (functions). Components are the central building pieces of any mapping. On the mapping area, components appear as rectangles. The following are examples of MapForce components:

- Constants
- Databases
- Filters
- Conditions
- Flat files (CSV, fixed-length, and other text files)
- Function components
- EDI documents (UN/EDIFACT, ANSI X12, HL7)
- Excel 2007+ files
- Simple input components
- Simple output components
- Variables
- XBRL documents
- XML Schemas and DTDs

**Connector**
A connector is a small triangle displayed on the left or right side of a component. The connectors displayed on the left of a component provide data entry points to that component. The connectors displayed on the right of a component provide data exit points from that component.

**Connection**
A connection is a line that you can draw between two connectors. By drawing connections, you instruct MapForce to transform data in a specific way (for example, read data from an XML document and write it to another XML document).

**Source component**
A source component is a component from which MapForce reads data. When you run the mapping, MapForce reads the data supplied by the connector of the source component, converts it to the required type, and sends it to the connector of the target component.

**Target component**
A target component is a component to which MapForce writes data. When you run the mapping, a target component instructs MapForce to either generate a file (or multiple files) or output the result as a string value for further processing in an external program. A target component is the opposite of a source component.
2.4 User Interface Overview

The graphical user interface of MapForce is organized as an integrated development environment. The main interface components are illustrated below. You can change the interface settings by using the menu command Tools | Customize.

Use the ▼ □ ▫ buttons displayed in the upper-right corner of each window to show, hide, pin, or dock it. If you need to restore toolbars and windows to their default state, use the menu command Tools | Restore Toolbars and Windows.

MapForce graphical user interface (MapForce Professional Edition)

Menu Bar and Toolbars
The Menu Bar displays the menu items. Each toolbar displays a group of buttons representing MapForce commands. You can reposition the toolbars by dragging their handles to the desired locations.

Libraries window
The Libraries window lists the MapForce built-in functions, organized by library. The list of available functions changes based on the transformation language you select. If you have created
user-defined functions, or if you imported external libraries, they also appear in the Libraries window.

To search functions by name or by description, enter the search value in the text box at the bottom of the Libraries window. To find all occurrences of a function (within the currently active mapping), right-click the function, and select Find All Calls from the context menu. You can also view the function data type and description directly from the Libraries window. For more information, see Working with Functions.

**Project window**

MapForce supports the Multiple Document Interface, and allows you to group your mappings into mapping projects. The Project window shows all files and folders that have been added to the project. Project files have the extension *.mfp (MapForce Project). To search for mappings inside projects, click anywhere inside the Projects window, and press CTRL + F. For more information, see Working with Mapping Projects.
Mapping pane
The Mapping pane is the working area where you design mappings. You can add mapping components (such as files, schemas, constants, variables, and so on) to the mapping area from the Insert menu (see Adding Components to the Mapping). You can also drag into the Mapping pane functions displayed in the Libraries window (see Working with Functions).

XSLT (XSLT2) pane
The XSLT (or XSLT2) pane displays the XSLT 1.0 (or 2.0) transformation code generated from your mapping. To switch to this pane, select XSLT (or XSLT 2) as transformation language, and then
click the **XSLT** tab (or **XSLT2** tab, respectively).

This pane provides line numbering and code folding functionality. To expand or collapse portions of code, click the "+" and "−" icons at the left side of the window. Any portions of collapsed code are displayed with an ellipsis symbol. To preview the collapsed code without expanding it, move the mouse cursor over the ellipsis. This opens a tooltip that displays the code being previewed, as shown in the image below. Note that, if the previewed text is too big to fit in the tooltip, an additional ellipsis appears at the end of the tooltip.

To configure the display settings (including indentation, end of line markers, and others), right-click the pane, and select **Text View Settings** from the context menu. Alternatively, click the **Text View Settings** () toolbar button.

**XQuery pane**

The XQuery pane displays the XQuery transformation code generated from your mapping, when you click the **XQuery** button. This pane is available when you select XQuery as transformation language. This pane also provides line numbering and code folding functionality, which works in a similar way as in the XSLT pane (see above).

**DB Query pane**

The DB Query pane allows you to directly query any major database. You can work with multiple active connections to different databases.
For more information, see Browsing and Querying Databases.

Output pane
The Output pane displays the result of the mapping transformation (for example, an XML file), when you click the Output button. If the mapping generates multiple files, you can navigate sequentially through each generated file.
This pane also provides line numbering and code folding functionality, which works in a similar way as in the XSLT pane (see above).

**StyleVision Output buttons**

If you have installed Altova StyleVision ([https://www.altova.com/stylevision.html](https://www.altova.com/stylevision.html)), the StyleVision output buttons enable you to preview and save the mapping output in HTML, RTF, PDF, and Word 2007+ formats. This is possible by means of StyleVision Power Stylesheet (SPS) files designed in StyleVision and assigned to a mapping component in MapForce.

**Overview window**

The Overview window gives a bird's-eye view of the Mapping pane. Use it to navigate quickly to a particular location on the mapping area when the size of the mapping is very large. To navigate to a particular location on the mapping, click and drag the red rectangle.
Messages window

The Messages window shows messages, errors, and warnings when you execute a mapping (see Previewing the Output) or perform a mapping validation (see Validating Mappings).

![Messages window screenshot]

To highlight on the mapping area the component or structure which triggered the information, warning, or error message, click the underlined text in the Messages window.

The results of a mapping execution or validation operation is displayed in the Messages window with one of the following status icons:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="GreenCheckmark.png" alt="Green Checkmark" /></td>
<td>Operation completed successfully.</td>
</tr>
<tr>
<td><img src="YellowExclamation.png" alt="Yellow Exclamation" /></td>
<td>Operation completed with warnings.</td>
</tr>
<tr>
<td><img src="RedX.png" alt="Red X" /></td>
<td>Operation has failed.</td>
</tr>
</tbody>
</table>

The Message window may additionally display any of the following message types: information messages, warnings, and errors.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="InformationMessage.png" alt="Information Message" /></td>
<td>Denotes an information message. Information messages do not stop the mapping execution.</td>
</tr>
<tr>
<td><img src="WarningMessage.png" alt="Warning Message" /></td>
<td>Denotes a warning message. Warnings do not stop the mapping execution. They may be generated, for example, when you do not create connections to some mandatory input connectors. In such cases, output will still be generated for those component where valid connections exist.</td>
</tr>
<tr>
<td><img src="ErrorMessage.png" alt="Error Message" /></td>
<td>Denotes an error. When an error occurs, the mapping execution fails, and no output is generated. The preview of the XSLT or XQuery code is also not possible.</td>
</tr>
</tbody>
</table>

Other buttons in the Messages window enable you to take the following actions:
<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="icon.png" alt="Icon" /></td>
<td>Filter messages by severity (information messages, errors, warnings). Select <strong>Check All</strong> to include all severity levels (this is the default behaviour). Select <strong>Uncheck All</strong> to remove all severity levels from the filter. In this case, only the general execution or validation status message is displayed.</td>
</tr>
<tr>
<td><img src="icon.png" alt="Icon" /></td>
<td>Jump to next line.</td>
</tr>
<tr>
<td><img src="icon.png" alt="Icon" /></td>
<td>Jump to previous line.</td>
</tr>
<tr>
<td><img src="icon.png" alt="Icon" /></td>
<td>Copy the selected line to clipboard.</td>
</tr>
<tr>
<td><img src="icon.png" alt="Icon" /></td>
<td>Copy the selected line to clipboard, including any lines nested under it.</td>
</tr>
<tr>
<td><img src="icon.png" alt="Icon" /></td>
<td>Copy the full contents of the Messages window to clipboard.</td>
</tr>
<tr>
<td><img src="icon.png" alt="Icon" /></td>
<td>Find a specific text in the Messages window. Optionally, to find only words, select <strong>Match whole word only</strong>. To find text while preserving the upper or lower case, select <strong>Match case</strong>.</td>
</tr>
<tr>
<td><img src="icon.png" alt="Icon" /></td>
<td>Find a specific text starting from the currently selected line up to the end.</td>
</tr>
<tr>
<td><img src="icon.png" alt="Icon" /></td>
<td>Find a specific text starting from the currently selected line up to the beginning.</td>
</tr>
<tr>
<td><img src="icon.png" alt="Icon" /></td>
<td>Clear the Messages window.</td>
</tr>
</tbody>
</table>

When you work with multiple mapping files simultaneously, you might want to display information, warning, or error messages in individual tabs for each mapping. In this case, click the numbered tabs available on the left side of the Messages window before executing or validating the mapping.

**Application status bar**

The application status bar appears at the bottom of the application window, and shows application-level information. The most useful of this information are the tooltips that are displayed here when you move the mouse over a toolbar button. If you are using the 64-bit version of MapForce, the application name appears in the status bar with the suffix (x64). There is no suffix for the 32-bit version.
2.5 Conventions

Example files
Most of the data mapping design files (files with .mfd extension, as well as other accompanying instance files) illustrated or referenced in this documentation are available in the following folders:

- C:\Users\<username>\Documents\Altova\MapForce2019\MapForce Examples
- C:\Users\<username>\Documents\Altova\MapForce2019\MapForce Examples\Tutorials

The example mappings and instance files accompanying MapForce illustrate most aspects of how it works, and you are highly encouraged to experiment with them as you learn about MapForce. When in doubt about the possible effects of making changes to the MapForce original examples, create back-ups before changing them.

Graphical user interface
Some of the images (screen shots) accompanying this documentation depict graphical user interface elements that may not be applicable to your MapForce edition. In relevant contexts, images typically include the name of the source mapping design (*.mfd) file, as well as the edition of MapForce in which the graphic was produced.
3 Tutorials

The MapForce tutorials are intended to help you understand and use the basic data transformation capabilities of MapForce in a short amount of time. You can regard these tutorials as a "crash course" of MapForce. While the goal is not to illustrate completely all MapForce features, you will be guided through the MapForce basics step-by-step, so it is recommended that you follow the tutorials sequentially. It is important that you understand each concept before moving on to the next one, as the tutorials gradually grow in complexity. Basic knowledge of XML and XML schema will be advantageous.

Convert XML to New Schema
This tutorial shows you how to convert data from an XML structure to another using the XSLT 2.0 language, without writing any code. You will also learn about MapForce sequences and items, creating mapping connections, using a function, validating and previewing a mapping, as well as saving the resulting output to the disk.

Map Multiple Sources to One Target
This tutorial shows you how to read data from two XML files with different schema and merge it into a single target XML file. You will also learn how to change the name and instance files of each mapping component, and the concept of "duplicate inputs".

Work with Multiple Target Schemas
This tutorial shows you how to work with more complex mappings that produce two or more target outputs. More specifically, you will learn how to generate, in the same mapping, an XML file that stores a list of book records, and another XML file that contains only a subset of the books in the first file, filtered by a specific publication year. To support filtering data, you will use a Filter component, a function and a numeric constant.

Process and Generate Files Dynamically
This tutorial shows you how to read data from multiple XML instance files located in the same folder and write it to multiple XML files generated on the fly. You will also learn about stripping the XML and schema declarations and using functions to concatenate strings and extract file extensions.
3.1 Convert XML to New Schema

This tutorial shows you how to convert data between two XML files, while helping you learn the basics of the MapForce development environment. Both XML files store a list of books, but their elements are named and organized in a slightly different way (that is, the two files have different schemas).

Abstract model of the data transformation

The code listing below shows sample data from the file that will be used as data source (for the sake of simplicity, the XML and the namespace declarations are omitted).

```
<books>
  <book id="1">
    <author>Mark Twain</author>
    <title>The Adventures of Tom Sawyer</title>
    <category>Fiction</category>
    <year>1876</year>
  </book>
  <book id="2">
    <author>Franz Kafka</author>
    <title>The Metamorphosis</title>
    <category>Fiction</category>
    <year>1912</year>
  </book>
</books>

books.xml

This is how data should look in the target (destination) file:

```
<library>
  <last_updated>2015-06-02T16:26:55+02:00</last_updated>
  <publication>
    <id>1</id>
    <author>Mark Twain</author>
  </publication>
</library>

library.xml
```
As you may have noticed, some element names in the source and target XML are not the same. Our goal is to populate the `<author>`, `<title>`, `<genre>` and `<publish_year>` elements of the target file from the equivalent elements in the source file (<author>, `<title>`, `<category>`, `<year>`). The attribute `id` in the source XML file must be mapped to the `<id>` element in the target XML file. Finally, we must populate the `<last_updated>` element of the target XML file with the date and time when the file was last updated.

To achieve the required data transformation, let's take the following steps.

**Step 1: Select XSLT2 as transformation language**
You can do this in one of the following ways:
- Click the XSLT2 (XSLT2) toolbar button.
- On the Output menu, click XSLT 2.0.

**Step 2: Add the source XML file to the mapping**
The source XML file for this mapping is located at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\books.xml`. You can add it to the mapping in one of the following ways:
- Click the Insert XML Schema/File (Insert XML Schema/File) toolbar button.
- On the Insert menu, click XML Schema/File.
- Drag the XML file from Windows Explorer into the mapping area.

Now that the file has been added to the mapping area, you can see its structure at a glance. In MapForce, this structure is known as a mapping component, or simply component. You can expand elements in the component either by clicking the collapse (Collapse) and expand icons (Expand), or by pressing the + and - keys on the numeric keypad.
To move the component inside the mapping pane, click the component header and drag the mouse to a new position. To resize the component, drag the corner of the component. You can also double-click the corner so that MapForce adjusts the size automatically.

The top level node represents the file name; in this particular case, its title displays the name of the XML instance file. The XML elements in the structure are represented by the icon, while XML attributes are represented by the icon.

The small triangles displayed on both sides of the component represent data inputs (if they are on the left side) or outputs (when they are on the right side). In MapForce, they are called input connectors and output connectors, respectively.

**Step 3: Add the target XML schema to the mapping**

To generate the target XML, we will use an existing XML schema file. In a real-life scenario, this file may have been provided to you by a third party, or you can create it yourself with a tool such as XMLSpy. If you don't have a schema file for your XML data, MapForce prompts you to generate it whenever you add an XML file without an accompanying schema or schema reference.

For this particular example, we are using an existing schema file available at: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\library.xsd`. To add it to the mapping, follow the same steps as with the source XML file (that is, click the Insert XML Schema/File ( toolbar button). Click Skip when prompted by MapForce to supply an instance file.
At this stage, the mapping design looks as follows:

Step 4: Make the connections
For each `<book>` in the source XML file, we want to create a new `<publication>` in the target XML file. We will therefore create a mapping connection between the `<book>` element in the source component and the `<publication>` element in the target component. To create the mapping connection, click the output connector (the small triangle) to the right of the `<book>` element and drag it to the input connector of the `<publication>` element in the target.

When you do this, MapForce may automatically connect all elements which are children of `<book>` in the source file to elements having the same name in the target file; therefore, four connections are being created simultaneously. This behavior is called "Auto Connect Matching Children" and it can be disabled and customized if necessary.

You can enable or disable the "Auto Connect Matching Children" behavior in one of the following ways:

- Click the **Toggle auto connect of children** button.
On the **Connection** menu, click **Auto Connect Matching Children**.

Notice that some of the input connectors on the target component have been highlighted by MapForce in orange, which indicates that these items are mandatory. To ensure the validity of the target XML file, provide values for the mandatory items as follows:

- Connect the `<category>` element in the source with the `<genre>` element in the target
- Connect the `<year>` element in the source with the `<publish_year>` element in the target

Finally, you need to supply a value to the `<last_updated>` element. If you move the mouse over its input connector, you can see that the element is of type `xs:dateTime`. Note that, for tips to be displayed, the **Show tips** (INFO) toolbar button must be enabled.

You can also make the data type of each item visible at all times, by clicking the **Show Data Types** (RT) toolbar button.

You can get the current date and time (that is, the `xs:dateTime` value) by means of a date and time XSLT function. To find the XSLT function to the mapping, start typing "date" in the text box located in the lower part of the Libraries window. Alternatively, double-click an empty area on the mapping and start typing "current-date".
As shown above, if you move the mouse over the "result" part of the function, you can see its description. For tips to be displayed, make sure that the Show tips toolbar button is enabled.

To add the function to the mapping, drag the function into the mapping pane, and connect its output to the input of the `<last_updated>` element.

You have now created a MapForce mapping design (or simply a "mapping") which converts data from the `books.xml` instance file (having the `books.xsd` schema) to the new `library.xml` file (having the `library.xsd` schema). If you double-click the header of each component, you can view these and other settings in the Component Settings dialog box, as shown below.
Component settings for the source

Component settings for the target

Step 5: Validate and save the mapping
Validating a mapping is an optional step that enables you to see and correct potential mapping errors and warnings before you run the mapping. To check whether the mapping is valid, do one of the following:

- On the File menu, click **Validate Mapping**.
- Click the Validate ( ![validate](x.png)) toolbar button.

The Messages window displays the validation results:
At this point, you might also want to save the mapping to a file. To save the mapping, do one of the following:

- On the **File** menu, click **Save**.
- Click the **Save** button in the toolbar.

For your convenience, the mapping created in this tutorial is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\BooksToLibrary.mfd`. Therefore, from this point onwards, you can either continue with the mapping file you created, or with the `BooksToLibrary.mfd` file.

**Step 6: Preview the mapping result**

You can preview the result of the mapping directly in MapForce. To do this, click the **Output** button located in the lower part of the mapping pane. MapForce runs the transformation and displays the result of the mapping in the **Output** pane.
You can now see the result of the transformation in MapForce.

By default, the files displayed for preview in the **Output** pane are not written to the disk. Instead, MapForce creates temporary files. To save the file displayed in the **Output** pane to the disk, select the menu command **Output | Save Output File**, or click the **Save generated output** (💾) toolbar button.

To configure MapForce to write the output directly to final files instead of temporary, go to **Tools | Options | General**, and then select the **Write directly to final output files** check box. Note that enabling this option is not recommended while you follow this tutorial, because you may unintentionally overwrite the original tutorial files.

You can also preview the generated XSLT code that performs the transformation. To preview the code, click the **XSLT2** button located in the lower area of the mapping pane.
To generate and save the XSLT2 code to a file, select the menu item **File | Generate Code in | XSLT 2.0**. When prompted, select a folder where the generated code must be saved. After code generation completes, the destination folder includes the following two files:

1. An XSLT transformation file, named after the target schema (in this example, `MappingMaptolibrary.xslt`).
2. A `DoTransform.bat` file. The `DoTransform.bat` file enables you to run the XSLT transformation in RaptorXML Server (for more information, see [https://www.altova.com/raptorxml.html](https://www.altova.com/raptorxml.html)).
3.2 Map Multiple Sources to One Target

In the previous tutorial, you have converted data from a source file (books.xml) to a target file (library.xml). The target file (library.xml) did not exist before running the mapping; it was generated by the mapping transformation. Let's now imagine a scenario where you already have some data in the library.xml file, and you want to merge this data with data converted from the books.xml. The goal in this tutorial is to design a mapping that generates a file called merged_library.xml. The generated file will include data from two sources: the books.xml file and the library.xml file. Note that the files used as source (books.xml and library.xml) have different schemas. If the source files had the same schema, you could also merge their data using a different approach (see Process and Generate Files Dynamically).

Abstract model of the data transformation

To achieve the required goal, let's take the following steps.

Step 1: Prepare the mapping design file

This tutorial uses as starting point the BooksToLibrary.mfd mapping from the <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ folder. You have already designed this mapping in the Convert XML to New Schema tutorial. To begin, open the BooksToLibrary.mfd file in MapForce, and save it with a new name.

Make sure to save the new mapping in the <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ folder, because it references several files from it.
Step 2: Create a second source component

First, select the target component and copy it (press Ctrl + C), and then paste it (press Ctrl + V) into the same mapping. Click the header of the new component and drag it under the books component.
The mapping now has two source components: **books** and **library**, and one target component: **library**.

You can always move the mapping components in any direction (left, right, top, bottom). Nevertheless, placing a source component to the left of a target component will make your mapping easier to read and understand by others. This is also the convention for all mappings illustrated in this documentation, as well as in the sample mapping files accompanying your MapForce installation.

**Step 3: Verify and set the input/output files**

In the previous step, the new source component was copy-pasted from the target component, so it inherits the same settings. To ensure that the name input/output instance files are correctly set, double-click the header of each component, and, in the Component Settings dialog box, verify and change the name and the input/output files of each component as shown below.
Components settings for the first source (books)

Component settings for the second source (library)
As shown above, the first source component reads data from books.xml. The second source component reads data from library.xml. Finally, the target component outputs data to a file called merged_library.xml.

Step 4: Make the connections
To instruct MapForce to write data from the second source to the target, click the output connector (small triangle) of the publications item in the source library component and drag it to the input connector of the publications item in the target library component. Because the target input connector already has a connection to it, the following notification message appears.

In this particular tutorial, replacing the connection is not what we want to achieve; our goal is to map data from two sources. Therefore, click Duplicate Input. By doing so, you configure the target component to accept data from the new source as well. The mapping now looks as follows:
Notice that the publication item in the target component has now been duplicated. The new publication(2) node will accept data from the source library component. Importantly, even though the name of this node appears as publication(2) in the mapping, its name in the resulting XML file will be publication, which is the intended goal.

You can now click the Output button at the bottom of the mapping pane, and view the mapping result. You will notice that data from both library.xml and books.xml files has now been merged into the new merged_library.xml file.
3.3 Work with Multiple Target Schemas

In the previous tutorial, Map Multiple Sources to One Target, you have seen how to map data from multiple source schemas to a single target schema. You have also created a file called merged_library.xml, which stores book records from two sources. Now let's assume that someone from another department has asked you to provide a subset of this XML file. Specifically, you must deliver an XML file that includes only the books published after 1900.

For convenience, you can modify the existing MultipleSourcesToOneTarget.mfd mapping so that, whenever required, you can generate both the complete XML library, and the filtered library.

![Abstract model of the data transformation](image)

In the diagram above, the data is first merged from two different schemas (books.xsd and library.xsd) into a single XML file called merged_library.xml. Secondly, the data is transformed using a filtering function and passed further to the next component, which creates an XML file called filtered_library.xml. The "intermediate" component acts both as data target and source. In MapForce, this technique is known as "chaining mappings", which is also the subject of this tutorial.

Our goal is to make it possible to generate at any time both the merged_library.xml and the filtered_library.xml. To achieve the goal, let's take the following steps.

Step 1: Prepare the mapping design file

This tutorial uses as starting point the MultipleSourcesToOneTarget.mfd mapping from the <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ folder. You have already designed this mapping in the Map Multiple Sources to One Target tutorial. To begin, open the MultipleSourcesToOneTarget.mfd file in MapForce, and save it with a new name.

Make sure to save the new mapping in the <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ folder, because it references several files from it.
Step 2: Add and configure the second target component

To add the second target component, click the **Insert XML Schema/File** toolbar button, and open the `library.xsd` file located in the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\` folder. Click **Skip** when prompted to supply a sample instance file. The mapping now looks as follows:
As shown above, the mapping now has two source components: **books** and **library**, and two target components. To distinguish between the target components, we will rename the second one to **filtered_library**, and also set the name of the XML file that should be generated by it. To do this, double-click the header of the right-most component and edit the component settings as follows:

Notice that the new name of the component is **filtered_library**, and the output XML file is named **filtered_library.xml**.

**Step 3: Make the connections**

Create a connection from the item **publication** in the **merged_library** to the item **publication** in the **filtered_library**. When you do this, a notification message is displayed.
Click OK. Notice that new buttons are now available in the upper-right corner of both target components: Preview (before) and Pass-through (after). These buttons will be used and explained in the following steps.

Step 4: Filter data
To filter data before supplying it to the filtered_library, we will use a Filter component. To add a filter component, right-click the connection between merged_library and filtered_library, and select Insert Filter: Nodes/Rows from the context menu.
The filter component has now been added to the mapping.

As shown above, the bool input connector is highlighted in orange, which suggests that an input is required. If you move the mouse over the connector, you can see that an input of type `xs:boolean` is required. Note that, for tips to be displayed, the Show tips ( ) toolbar button must be enabled.
The filter component requires a condition that returns either `true` or `false`. When the Boolean condition returns `true`, data of the current `publication` sequence will be copied over to the target. When the condition returns `false`, data will not be copied.

In this tutorial, the required condition is to filter all books which were published after 1900. To create the condition, do the following:

1. Add a constant of numeric type having the value "1900" (On the **Insert** menu, click **Constant**). Choose **Number** as type.

2. In the Libraries window, locate the function `greater` and drag it to the mapping pane.

3. Make the mapping connections to and from the function `greater` as shown below. By doing this, you are instructing MapForce: "When `publish_year` is greater than 1900, copy the current `publication` source item to the `publication` target item".
Step 5: Preview and save the output of each target component

You are now ready to preview and save the output of both target components. When multiple target components exist in the same mapping, you can choose which one to preview by clicking the Preview button. When the Preview button is in a pressed state, it indicates that that specific component is currently enabled for preview (and this particular component will generate the output in the Preview pane). Only one component at a time can have the preview enabled.

Therefore, when you want to view and save the output of the merged_library (that is, the "intermediate") component, do the following:

1. Click the Preview button on the merged_library component.
2. Click the Output button at the bottom of the mapping pane.
3. On the Output menu, click Save Output File if you want to save the output to a file.

When you want to view and save the output of the filtered_library component:

1. Click the Pass-through button on the merged_library component.
2. Click the Preview button on the filtered_library component.
3. Click the Output button at the bottom of the mapping pane.
4. On the Output menu, click Save Output File if you want to save the output to a file.

Notice the Pass-through button—clicking or not clicking it makes a big difference in any mapping which has multiple target components, including this one. When this button is in a pressed state, MapForce lets data pass through the intermediate component, so that you can preview the result of the entire mapping.

Release the button if you want to preview only the portion of the mapping between the merged_library and the filtered_library. In the latter case, an error will be generated. This behavior is expected, because the intermediate component does not have a valid input XML file from which it should read data. To solve the problem, double-click the header of the component and edit so as to supply a valid input XML file, as shown below:
You have now finished designing a mapping which has multiple target components, and you can view and save the output of each target, which was the intended goal of this tutorial. For further information about working with pass-through components, see Chained mappings / pass-through components.
3.4 Process and Generate Files Dynamically

This tutorial shows you how to read data from multiple source XML files and write it to multiple target files in the same transformation. To illustrate this technique, we will now create a mapping with the following goals:

1. Read data from multiple XML files in the same directory.
2. Convert each file to a new XML schema.
3. For each source XML file, generate a new XML target file under the new schema.
4. Strip the XML and namespace declaration from the generated files.

Abstract model of the data transformation

We will use three source XML files as example. The files are located in the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\` folder, and they are named `bookentry1.xml`, `bookentry2.xml`, and `bookentry3.xml`. Each of the three files stores a single book.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="books.xsd">
  <book id="1">
    <author>Mark Twain</author>
    <title>The Adventures of Tom Sawyer</title>
    <category>Fiction</category>
    <year>1876</year>
  </book>
</books>
```

`bookentry1.xml`
The source XML files use the books.xsd schema available in the following folder: <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\>. To convert the source files to a new XML schema, we will use the library.xsd schema (available in the same folder). After the transformation, the mapping will generate three files according to this new schema (see the code listings below). We will also configure the mapping so that the name of the generated files will be: publication1.xml, publication2.xml, and publication3.xml. Notice that the XML declaration and the namespace declaration must be stripped.
To achieve the goals, let's take the following steps.

**Step 1: Prepare the mapping design file**

This tutorial uses as starting point the `BooksToLibrary.mfd` mapping from the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\` folder. You have already designed this mapping in the *Convert XML to New Schema* tutorial. To begin, open the `BooksToLibrary.mfd` file in MapForce, and save it with a new name, in the same folder.

Make sure to save the new mapping in the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\` folder, because it references several files from it.
Step 2: Configure the input
To instruct MapForce to process multiple XML instance files, double-click the header of the source component. In the Component Settings dialog box, enter `bookentry*.xml` as input file.

The asterisk (`*`) wildcard character in the file name instructs MapForce to use as mapping input all the files that have the `bookentry-` prefix. Because the path is a relative one, MapForce will look for all `bookentry-` files in the same directory as the mapping file. Note that you could also enter an absolute path if necessary, while still using the `*` wildcard character.

Step 3: Configure the output
To create the file name of each output file, we will use the `concat` function. This function concatenates (joins) all the values supplied to it as argument.
To build the file name using the `concat` function:

1. Search for the `concat` function in the Libraries window and drag it to the mapping area. By default, this function is added to the mapping with two parameters; however, you can add new parameters if necessary. Click the `Add parameter` symbol inside the function component and add a third parameter to it. Note that clicking the `Delete parameter` symbol deletes a parameter.

![Concat Function Diagram]

2. Insert a constant (on the `Insert` menu, click `Constant`). When prompted to supply a value, enter "publication" and leave the `String` option unchanged.

![Insert Constant Dialog]

3. Connect the constant with `value1` of the `concat` function.

![Constant Connection]

4. Connect the `id` attribute of the source component with `value2` of the `concat` function.
5. Search for the `get-fileext` function in the Libraries window and drag it to the mapping area. Create a connection from the top node of the source component (File: `books.xml`) to the `filepath` parameter of this function. Then create a connection from the result of the `get-fileext` function to `value3` of the `concat` function. By doing this, you are extracting only the extension part (in this case, `.xml`) from the source file name.

So far, you have provided as parameters to the `concat` function the three values which, when joined together, will create the generated file name (for example, `publication1.xml`):
The constant "publication" supplies the constant string value "publication".

The attribute id of the source XML file supplies a unique identifier value for each file. This is to prevent all files from being generated with the same name.

The get-fileext function returns the extension of the file name to be generated.

You can now instruct MapForce to actually build the file name when the mapping runs. To do this, click the File (File) or File/String (File/String) button of the target component and select **Use Dynamic File Names Supplied by Mapping**.

You have now instructed MapForce to generate the instance files dynamically, with whatever name will be provided by the mapping. In this particular example, the name is created by the concat function; therefore, we will connect the result of the concat function with the File: <dynamic> node of the target component.
If you double-click the target component header at this time, you will notice that the **Input XML File** and **Output XML File** text boxes are disabled, and their value shows `<File names supplied by the mapping>`.

This serves as an indication that you have supplied the instance file names dynamically from a mapping, so it is no longer relevant to define them in the component settings.

Finally, you need to strip the XML namespace and schema declaration from the target. To achieve this, clear the selection from the **Add schema/DTD reference...** and **Write XML Declaration** check boxes on the Component Settings dialog box.
You can now run the mapping and see the result, as well as the name of generated files. This mapping generates multiple output files. You can navigate through the output files using the left and right buttons in the upper left corner of the output pane, or by picking a file from the adjacent drop-down list.
Chapter 4

Common Tasks
4 Common Tasks

This section describes common MapForce tasks and concepts, such as working with mappings, components, connections, and mapping projects.
4.1 Working with Mappings

A MapForce mapping design (or simply "mapping") is the visual representation of how data is to be transformed from one format to another. A mapping consists of components that you add to the MapForce mapping area in order to create your data transformations (for example, convert XML documents from one schema to another). A valid mapping consists of one or several source components connected to one or several target components. You can run a mapping and preview its result directly in MapForce. You can generate code and execute it externally. You can also compile a mapping to a MapForce execution file and automate mapping execution using MapForce Server or FlowForce Server. MapForce saves mappings as files with .mfd extension.

To create a new mapping:

1. Do one of the following:
   - On the File menu, click New.
   - Click the New ( ) toolbar button.
2. Click Mapping, and then click OK.

Your mapping is now created; however, it does not yet do anything because it is empty. A mapping requires at least two connected components to become valid, so the next step is to add components to the mapping (see Adding Components to the Mapping) and draw connections between components (see Working with Connections).

4.1.1 Adding Components to the Mapping

In MapForce, the term "component" is what represents visually the structure (schema) of your data, or how data is to be transformed (functions). Components are the central building pieces of any mapping. On the mapping area, components appear as rectangles. The following are examples of MapForce components:

- Constants
- Databases
- Filters
- Conditions
- Flat files (CSV, fixed-length, and other text files)
- Function components
- EDI documents (UN/EDIFACT, ANSI X12, HL7)
- Excel 2007+ files
Common Tasks Working with Mappings

- Simple input components
- Simple output components
- Variables
- XBRL documents
- XML Schemas and DTDs

To add a component to the mapping, do one of the following:

- On the **Insert** menu, click the option relevant for the component type you wish to add (for example, **XML Schema/File**).
- Drag a file from Windows File Explorer onto the mapping area. Note that this operation is possible only for compatible file-based components.
- Click the relevant button on the Insert Component toolbar.

---

**Insert Component toolbar (MapForce Enterprise Edition)**

Each component type has specific purpose and behavior. For component types where that is necessary, MapForce walks you through the process by displaying contextual wizard steps or dialog boxes. For example, if you are adding an XML schema, a notification dialog box prompts you to optionally select an instance file as well.

For an introduction to components, see [Working with Components](#). For specific information about each technology supported as mapping source or target, see [Data Sources and Targets](#). For information about MapForce built-in components used to store data temporarily or transform it (such as filtering or sorting), see [Designing Mappings](#).

### 4.1.2 Adding Components from a URL

In addition to adding local files as mapping components, you can also add files from a URL. Note that this operation is supported when you add a component as source component (that is, your mapping reads data from the remote file). The supported protocols are HTTP, HTTPS, and FTP.

To add a component from a URL:

1. On the **Insert** menu, select the type of the component type you wish to add (for example, **XML Schema/File**).
2. On the **Open** dialog box, click **Switch to URL**.
3. Enter the URL of the file in the **File URL** text box, and click **Open**.
Common Tasks Working with Mappings

Make sure that the file type in the File URL text box is the same as the file type you specified in step 1.

If the server requires password authentication, you will be prompted to enter the user name and password. If you want the user name and password to be remembered next time you start MapForce, enter them in the Open dialog box and select the Remember password between application starts check box.

The Open As setting defines the grammar for the parser when opening the file. The default and recommended option is Auto.

If the file you are loading is not likely to change, select the Use cache/proxy option to cache data and speed up loading the file. Otherwise, if you want the file to be reloaded each time when you open the mapping, select Reload.

For servers with Web Distributed Authoring and Versioning (WebDAV) support, you can browse files after entering the server URL in the Server URL text box and clicking Browse. Although the preview shows all file types, make sure that you choose to open the same file type as specified in step 1 above; otherwise, errors will occur.

If the server is a Microsoft SharePoint Server, select the This is a Microsoft SharePoint Server check box. Doing so displays the check-in or check-out state of the file in the preview area. If you want to make sure that no one else can edit the file on the server while you are using it in
MapForce to read data from it, right-click the file and select **Check Out**. To check in any file that was previously checked out by you, right-click the file and select **Check In**.

Open dialog box (in Switch to URL mode)

### 4.1.3 About Data Streaming

Data streaming is a MapForce built-in mechanism that allows you to use arbitrarily large data sources as input or output to your mappings. Data streaming should not be confused with stream objects in MapForce generated code. (The latter represent a possible way of handling data if you integrate MapForce generated code with a custom C# and Java application.)

Data streaming applies to the following data sources:

- XML files
- CSV files
- Fixed-length field files
- Databases

When you use any of the above data sources as input or output in your mappings, MapForce treats the data source as an open stream of data, and processes its contents sequentially, instead of loading all data into the memory.
Note: Data streaming is possible only if you have selected BUILT-IN as transformation language (see Selecting a transformation language).

Memory usage considerations
When you work with mapping inputs and outputs that are data streaming candidates, “Out of memory” errors can occur if your mapping requires random access to the input source.

For example, let’s assume that your mapping contains a component that applies a group-by function on the source data. If you apply the group-by function on the entire tree structure of the input file, this would require the entire source file to be loaded into memory, and, consequently, file streaming would no longer be possible. The same is true for any operation which would require the whole contents of the mapping source to be loaded into memory, such as sorting.

When situations such as the one described above occur, the transformation will nevertheless complete successfully if there is enough virtual memory and disk space available on your system.

4.1.4 Selecting a Transformation Language

To meet your data mapping needs, MapForce provides the ability to choose between various transformation languages.

By default, MapForce provides a robust, built-in engine capable of performing the same transformations supported in other languages. When you deploy MapForce mappings to MapForce Server, the built-in engine executes them without the need for any external processors. Furthermore, if you require minimal or no manual intervention in your data transformation process, you can use FlowForce Server to automate mapping processes by means of scheduled jobs.

Consider choosing the transformation language after testing several approaches and determining what works best for your data. The available transformation languages are as follows:

- BUILT-IN (This is the default native transformation engine used by MapForce.)
- C++
- C#
- Java
- XQuery
- XSLT 1.0
- XSLT 2.0

To select a transformation language, do one of the following:

- On the Output menu, click the name of the language you wish to use for transformation.
- Click the name of the language in the Language Selection toolbar.

Note: Some mapping inputs and outputs are not supported by certain languages. For example, if you use a database as mapping input or output, you cannot generate XSLT code. Therefore, if you attempt to generate the code or preview the output of a mapping that has
sources or targets not supported by the selected language, MapForce displays a relevant notification message.

The BUILT-IN transformation language

When you select BUILT-IN (\[\text{BUILT-IN}\]) as transformation language for your mapping, MapForce uses its internal transformation engine to execute the data mapping. MapForce also uses this option implicitly, whenever you want to preview the output of a mapping where the selected transformation language is Java, C#, or C++.

It is recommended to set the transformation language to BUILT-IN in the following cases:

- As default option, when you do not necessarily need to use a specific language to transform data. The BUILT-IN transformation language supports most MapForce features. As stated subsequently in this documentation, certain MapForce features are not supported when the transformation language is Java, C#, or C++.
- If you intend to deploy and execute the mapping on a different Linux, Mac, or Windows machine with MapForce Server, see Automating Mappings and MapForce.
- If you are processing large files and memory usage is a concern.
- If you need to generate digital signatures in the output XML file.

4.1.5 Validating Mappings

MapForce validates mappings automatically, when you click the Output tab to preview the transformation result. You can also validate a mapping explicitly, before attempting to preview its result. This helps you identify and correct potential mapping errors and warnings before the mapping is run. Note that running a mapping may generate additional runtime errors or warnings depending on the processed data, for example, when values mapped to attributes are overwritten.

To validate a mapping explicitly, do one of the following:

- On the File menu, click Validate Mapping.
- Click the Validate (\[\text{\textbullet\textperiodcentered}\]) toolbar button.

The Messages window displays the validation results, for example:

![Messages window]

When you validate a mapping, MapForce checks for the validity of the mapping (such as incorrect
or missing connections, unsupported component kinds), and the validation result is then displayed in the Messages window with one of the following status icons:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>Validation has completed successfully.</td>
</tr>
<tr>
<td>☢️</td>
<td>Validation has completed with warnings.</td>
</tr>
<tr>
<td>⚪️</td>
<td>Validation has failed.</td>
</tr>
</tbody>
</table>

The Message window may additionally display any of the following message types: information messages, warnings, and errors.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>📣</td>
<td>Denotes an information message. Information messages do not stop the mapping execution.</td>
</tr>
<tr>
<td>🚨</td>
<td>Denotes a warning message. Warnings do not stop the mapping execution. They may be generated, for example, when you do not create connections to some mandatory input connectors. In such cases, output will still be generated for those component where valid connections exist.</td>
</tr>
<tr>
<td>⚠️</td>
<td>Denotes an error. When an error occurs, the mapping execution fails, and no output is generated. The preview of the XSLT or XQuery code is also not possible.</td>
</tr>
</tbody>
</table>

To highlight on the mapping area the component or structure which triggered the information, warning, or error message, click the underlined text in the Messages window.

For components that transform data (such as functions or variables), MapForce validation works as follows:

- If a mandatory **input connector** is unconnected, an error message is generated and the transformation is stopped.
- If an **output connector** is unconnected, then a warning is generated and the transformation process continues. The offending component and its data are ignored and are not mapped to the target document.

To display the result of each validation in an individual tab, click the numbered tabs available on the left side of the Messages window. This may be useful, for example, if you work with multiple mapping files simultaneously.

Other buttons in the Messages window enable you to take the following actions:

- Filter the message by types (for example, to show only errors or warnings)
- Move up or down through the entries
- Copy the message text to the clipboard
- Find a specific text in the window
- Clear the Messages window.

For general information about the Messages window, see [User Interface Overview](#).
4.1.6 Validating the Mapping Output

After you click the Output tab to preview the mapping, the resulting output becomes available in the Output pane. You can validate this output against the schema associated with it. For example, if the mapping transformation generates an XML file, then the resulting XML document can be validated against the XML schema. If the target component is an EDI file, then the output is validated against the EDI specification (see EDI Component Validation). Likewise, if the target is a JSON object, the output is validated against the JSON schema.

For XML files, you can specify the schema associated with the instance file in the Add Schema/DTD reference field of the Component Settings dialog box (see XML Component Settings). The path specifies where the schema file referenced by the produced XML output is to be located. This ensures that the output instance can be validated when the mapping is executed. You can enter an http:// address in this field, as well as an absolute or relative path. If you do not select the Add Schema/DTD reference field, then the validation of the output file against the schema is not possible. If you select this check box but leave it empty, then the schema filename of the Component Settings dialog box is generated into the output and the validation is done against it.

To validate the mapping output, do one of the following:

- Click the Validate Output toolbar button.

  ![Validate Output Tool Button](image)

- On the Output menu, click Validate Output File.

Note: The Validate Output button and its corresponding menu command (Output | Validate Output File) are enabled only if the output file supports validation against a schema.

The result of the validation is displayed in the Messages window, for example:

![Validating Output Result](image)

If the validation was not successful, the message contains detailed information on the errors that occurred.
The validation message contains a number of hyperlinks you can click for more detailed information:

- Clicking the file path opens the output of the transformation in the Output tab of MapForce.
- Clicking `<ElementName>` link highlights the element in the Output tab.
- Clicking the ![icon](image) icon opens the definition of the element in XMLSpy (if installed).
- Clicking the hyperlinks in the Details subsection (e.g., cvc-model-group) opens a description of the corresponding validation rule on the https://www.w3.org/ website.

### 4.1.7 Previewing the Output

When working with MapForce mappings, you can preview the resulting output without having to run and compile the generated code with an external processor or compiler. In general, it is a good idea to preview the transformation output within MapForce before attempting to process the generated code externally.

When you choose to preview the mapping results, MapForce executes the mapping and populates the Output pane with the resulting output.

Once data is available in the Output pane, you can validate and save it if necessary (see Validating the Mapping Output). You can also use the Find command (Ctrl + F key combination) to quickly locate a particular text pattern within the output file (see also Searching in Text View).

Any errors, warning, or information messages related to the mapping execution are displayed in the Messages window (see User Interface Overview).

To preview the transformation output:

- Click the Output tab under the Mapping window. MapForce executes the mapping using the transformation language selected in the Language toolbar and populates the Output pane with the resulting output.

**Note:** If you select C++, C#, or Java as transformation language, MapForce executes the mapping using its built-in transformation engine. The result that appears in the Output pane is the same as if the Java, C++, or C# code had been generated, compiled and executed.

To save the transformation output, do one of the following:

- On the Output menu, click **Save Output File**.
- Click the **Save Generated Output** toolbar button.
Partial output preview

When you are previewing large output files, MapForce limits the amount of data displayed in the Output pane. More specifically, MapForce displays only a part of the file in the Output pane, and a Load more... button appears in the lower area of the pane. Clicking the Load more... button appends the next file part to the currently visible data, and so on.

![File size and load options](image)

**Note:** The Pretty-print button becomes active when the complete file has been loaded into the Output pane.

You can configure the preview settings from the General tab of the Options dialog box (see Changing the MapForce Options).

### 4.1.8 Text View Features

The Output pane, the XSLT pane, as well as the XQuery pane have multiple visual aids to make the display of text easier. These include:

- Line Numbers
- Syntax Coloring
- Bookmarks
- Source Folding
- Indentation Guides
- End-of-Line and Whitespace Markers
- Zooming
- Pretty-printing
- Word wrapping
- Text highlighting

Where applicable, you can toggle or customize the features above from the Text View Settings dialog box. Settings in the Text View Settings dialog box apply to the entire application—not only to the active document.
To open the **Text View settings** dialog box, do one of the following:

- On the **Output** menu, select **Text View Settings**.
- Click the **Text View Settings** toolbar button.
- Right-click the Output pane, and select **Text View Settings** from the context menu.

Some of the navigation aids can also be toggled from the Text View toolbar, the application menu, or keyboard shortcuts.

For reference to all applicable shortcuts, see the "Key Map" section of the **Text View Settings** dialog box illustrated above.

**Line numbers**

Line numbers are displayed in the line numbers margin, which can be toggled on and off in the **Text View Settings** dialog box. When a section of text is collapsed, the line numbers of the collapsed text are also hidden.
Syntax coloring
Syntax coloring is applied according to the semantic value of the text. For example, in XML documents, depending on whether the XML node is an element, attribute, content, CDATA section, comment, or processing instruction, the node name (and in some cases the node’s content) is colored differently.

Bookmarks
Lines in the document can be bookmarked for quick reference and access. If the bookmarks margin is toggled on, bookmarks are displayed in the bookmarks margin.

```
<?xml version="1.0" encoding="UTF-8"?>
<CompletePO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="CompletePO.xsd">
    <Customer>
        <Number>3</Number>
        <FirstName>Ted</FirstName>
        <LastName>Little</LastName>
        <Address>
            <Street>Long Way</Street>
            <City>Los-Angeles</City>
            <ZIP>34424</ZIP>
            <State>CA</State>
        </Address>
    </Customer>
    <LineItems>
        <LineItem>....</LineItem>
        <LineItem>....</LineItem>
    </LineItems>
    <Total>
        <TotalSum>596</TotalSum>
        <TotalItems>2</TotalItems>
    </Total>
</CompletePO>
```

Otherwise, bookmarked lines are highlighted in cyan.
The bookmarks margin can be toggled on or off in the **Text View Settings** dialog box.

You can edit and navigate bookmarks using the following commands:

- **Insert/Remove Bookmark (Ctrl + F2)**
- **Go to Next Bookmark (F2)**
- **Go to Previous Bookmark (Shift + F2)**
- **Delete All Bookmarks (Ctrl + Shift + F2)**

The commands above are available in the **Output** menu. Bookmark commands are also available through the context menu, when you right-click the **Output** (or **XSLT**, or **XQuery**) pane.

**Source folding**

Source folding refers to the ability to expand and collapse nodes and is displayed in the source folding margin. The margin can be toggled on and off in the Text View Settings dialog box. To expand or collapse portions of text, click the “+” and “-” nodes at the left side of the window. Any portions of collapsed code are displayed with an ellipsis symbol. To preview the collapsed code without expanding it, move the mouse cursor over the ellipsis. This opens a tooltip that displays the code being previewed, as shown in the image below. Note that, if the previewed text is too big to fit in the tooltip, an additional ellipsis appears at the end of the tooltip.
Indentation guides

Indentation guides are vertical dotted lines that indicate the extent of a line's indentation. They can be toggled on and off in the Text View Settings dialog box.

Note: The Insert tabs and Insert spaces options take effect when you use the Output | Pretty-Print XML text option.

End-of-line markers, whitespace markers

End-of-line (EOL) markers and whitespace markers can be toggled on in the Text View Settings dialog box. The image below shows a document where both end-of-line and whitespace markers are visible. An arrow represents a tab character, a "CR" is a carriage return, and a dot represents a space character.

Zooming in and out

You can zoom in and out by scrolling (with the scroll-wheel of the mouse) while holding the Ctrl key pressed. Alternatively, press the "-" or "+" keys while holding the Ctrl key pressed.

Pretty-printing

The Pretty-Print XML Text command reformats the active XML document in Text View to give a
structured display of the document. By default, each child node is offset from its parent by four space characters. This can be customized from the Text View Settings dialog box.

To pretty-print an XML document, select the Output | Pretty-Print XML Text menu command, or click the Pretty Print toolbar button.

Word wrapping
To toggle word wrapping in the currently active document, select the Output | Word Wrap menu command, or click the Word Wrap toolbar button.

Text highlighting
When you select text, all matches in the document of the text selection that you make are highlighted automatically. The selection is highlighted in pale blue, and matches are highlighted in pale orange. The selection and its matches are indicated in the scroll bar by gray marker-squares. The current cursor position is given by the blue cursor-marker in the scroll bar.

To switch text highlighting on, select Enable auto-highlighting in the Text View Settings dialog box. A selection can be defined to be an entire word or a fixed number of characters. You can also specify whether casing should be taken into account or not.

For a character selection, you can specify the minimum number of characters that must match, starting from the first character in the selection. For example, you can choose to match two or more characters. In this case, one-character selections will not be matched, but a selection consisting of two or more characters will be matched. So, in this case, if you select "e", then no matches will be shown; selecting "ey" will show all "ey" matches; selecting "eyp" will show all "eyp" matches; and so on.

For word searches, the following are considered to be separate words: element names (without angular brackets), the angular brackets of element tags, attribute names, and attribute values without quotes.

4.1.9 Searching in Text View

The text in the Output pane, the XQuery pane, as well as the XSLT pane can be searched using an extensive set of options and visual aids.

To start a search, press Ctrl+F (or select the menu command Edit | Find). You can then search in the entire document or within a text selection for a search term that you enter in the dialog.

- Enter a string to find, or use the combo box to select a string from one of the last 10 strings.
- When you enter or select a string to find, all matches are highlighted and the positions of the matches are indicated by beige markers in the scroll bar.
- The currently selected match has a different highlight color than the other matches, and its position is indicated in the scroll bar by the dark blue cursor-marker.
- The total number of matches is listed below the search term field, together with the index position of the currently selected match. For example, 2 of 4 indicates that the second of four matches is currently selected.
- You can move from one match to the next, in both directions, by selecting the Previous
Working with Mappings

(Shift+F3) and Next (F3) buttons at bottom right.

To close the Find dialog, click the Close button at top right, or press Esc.

Note the following points:

- The Find dialog is *modeless*. This means that it can remain open while you continue to use Text View.
- If text is selected prior to opening the dialog box, then the selected text is automatically inserted into the search term field.
- To search within a selection, do the following: (i) Mark the selection; (ii) Toggle on the **Find in Selection** option to lock the selection; (iii) Enter the search term. To search within another selection, unlock the current selection by toggling off the **Find in Selection** option, then make the new selection and toggle on the **Find in Selection** option.
- After the Find dialog is closed, you can repeat the current search by pressing **F3** for a forward search, or **Shift-F3** for a backward search. The Find dialog will appear again in this case.

**Find options**

Find criteria can be specified via buttons located below the search term field. When an option is toggled on, its button color changes to blue. You can select from the following options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Match case</td>
<td>Aa</td>
<td>Performs a case-sensitive search when toggled on (&quot;Address&quot; is not the same as &quot;address&quot;).</td>
</tr>
</tbody>
</table>
### Option | Icon | Description
--- | --- | ---
Match whole word | ![ABC](https://via.placeholder.com/15) | Only the exact words in the text will be matched. For example, for the input string *fit*, with **Match whole word** toggled on, only the word *fit* will match the search string; the *fit* in *fitness*, for example, will not.

Regular expression | ![.*](https://via.placeholder.com/15) | If toggled on, the search term will be read as a regular expression. See “Using regular expressions” below.

Find anchor | ![T](https://via.placeholder.com/15) | When a search term is entered, the matches in the document are highlighted and one of these matches will be marked as the current selection. The **Find anchor** toggle determines whether that first current selection is made relative to the cursor position or not. If **Find anchor** is toggled on, then the first currently selected match will be the next match from the current cursor location. If **Find anchor** is toggled off, then the first currently selected match will be the first match in the document, starting from the top.

Find in selection | ![E](https://via.placeholder.com/15) | When toggled on, locks the current text selection and restricts the search to the selection. Otherwise, the entire document is searched. Before selecting a new range of text, unlock the current selection by toggling off the **Find in Selection** option.

**Using regular expressions**

You can use regular expressions (regex) to find a text string. To do this, first, switch the **Regular expression** option on. This specifies that the text in the search term field is to be evaluated as a regular expression. Next, enter the regular expression in the search term field. For help with building a regular expression, click the **Regular Expression Builder** button, which is located to the right of the search term field. Click an item in the Builder to enter the corresponding regex metacharacter/s in the search term field. The screenshot below shows a simple regular expression to find email addresses.
The following custom set of regular expression metacharacters are supported when finding and replacing text.

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Matches any character. This is a placeholder for a single character.</td>
</tr>
</tbody>
</table>
| (abc)     | The ( and ) metacharacters mark the start and end of a tagged expression. Tagged expressions may be useful when you need to tag ("remember") a matched region for the purpose of referring to it later (back-reference). Up to nine sub-expressions can be tagged (and then back-referenced later).

For example, \(\texttt{the}\) \(\texttt{the}\) matches the string \(\texttt{the the}\). This expression can be literally explained as follows: match the string "the" (and remember it as a tagged region), followed by a space character, followed by a back-reference to the tagged region matched previously.

| \n | Where \(n\) is 1 through 9, \(n\) refers to the first through ninth tagged region (see above). |
| \< | Matches the start of a word. |
| \> | Matches the end of a word. |
| \ \ | Escapes the character following the backslash. In other words, the expression \(\texttt{\textbackslash x}\) allows you to use the character \(x\) literally. For example, \(\texttt{\textbackslash i}\) would be interpreted as i and not as the start of a character set. |
| [...] | Matches any characters in this set. For example, \(\texttt{[abc]}\) matches any of the characters \(a\), \(b\) or \(c\). You can also use ranges: for example \(\texttt{[a-z]}\) for any lower case character. |
| [...^] | Matches any characters not in this set. For example, \(\texttt{[^A-Za-z]}\) matches any...
character except an alphabetic character.

^ Matches the start of a line (unless used inside a set, see above).

$ Matches the end of a line. For example, $A+$ matches one or more A’s at end of line.

* Matches zero or more occurrences of the preceding expression. For example, $Sa+m$ matches Sm, Sam, Saam, Saaam and so on.

+ Matches one or more occurrences of the preceding expression. For example, $Sa+m$ matches Sam, Saam, Saaam and so on.

Finding special characters
You can search for any of the following special characters within text, provided that the Regular expression option is enabled:
- \t (Tab)
- \r (Carriage Return)
- \n (New line)
- \ (Backslash)

For example, to find a tab character, press Ctrl + F, select the option, and then enter \t in the Find dialog box.

4.1.10 Previewing the XSLT Code

You can preview the XSLT code generated by MapForce if you selected XSLT 1.0 or XSLT 2.0 as data transformation language (see Selecting a transformation language).

To preview the generated XSLT 1.0 (or XSLT 2.0) code, do one of the following:

- To preview the XSLT 1.0 code, click the XSLT tab under the Mapping window.
- To preview the XSLT 2.0 code, click the XSLT2 tab under the Mapping window.

Note: The XSLT (or XSLT2) tab becomes available if you have selected XSLT (or XSLT2, respectively) as transformation language.

4.1.11 Generating XSLT Code

To generate XSLT code:

1. Select the menu item File | Generate code in | XSLT 1.0 (XSLT 2.0).
2. Select the folder you want to save the generated XSLT file, and click OK. MapForce generates the code and displays the result of the operation in the Messages window.

The name of the generated .xslt file has the form <A>MapTo<B>.xslt, where:

- "<A>" is the value of the Application Name field in mapping settings (see Changing the Mapping Settings).
- "<B>" is the name of the target mapping component. To change this value, open the
settings of the target component and edit the value of the **Component Name** field (see [Changing the Component Settings](#)).

The folder where the .xslt file is saved also contains a batch file called **DoTransform.bat** which can be run with RaptorXML Server to transform the data (see [Automation with RaptorXML Server](#)).

**To run the transformation with RaptorXML Server:**

1. Download and install RaptorXML from the download page ([https://www.altova.com/download#server](https://www.altova.com/download#server)).
2. Start the **DoTransform.bat** batch file located in the previously designated output folder.

Note that you might need to add the RaptorXML installation location to the **path** variable of the Environment Variables. You can find the RaptorXML documentation on the website documentation page ([https://www.altova.com/documentation](https://www.altova.com/documentation)).

### 4.1.12 Previewing the XQuery Code

You can preview the XQuery code generated by MapForce if you selected **XQuery** as data transformation language (see [Selecting a transformation language](#)).

To preview the generated XQuery code:

- Click the **XQuery** tab under the Mapping window.

**Note:** The XQuery tab becomes available if you have selected **XQuery** as transformation language.

### 4.1.13 Working with Multiple Mapping Windows

MapForce uses a Multiple Document Interface (MDI). Each mapping file you open in MapForce has a separate window. This enables you to work with multiple mapping windows and arrange or resize them in various ways inside the main (parent) MapForce window. You can also arrange all open windows using the standard Windows layouts: Tile horizontally, Tile vertically, Cascade.

When multiple mappings are open in MapForce, you can quickly switch between them using the tabs displayed in the lower part of the Mapping pane.
Window management options are available both on the **Window** menu and on the **Windows** dialog box. From the **Windows** dialog box, you can take actions against any or all currently open mapping windows (including saving, closing, or minimizing them).

You can open the Windows dialog box using the menu command **Window | Windows**. To select multiple windows in the Windows dialog box, click the required entries while holding the **Ctrl** key pressed.
4.1.14 Changing the Mapping Settings

You can change the document-specific settings of the currently active mapping design file from the Mapping Settings dialog box. This information is stored in the *.mfd file.

To open the Mapping Settings dialog box:

- On the File menu, click Mapping Settings.

![Mapping Settings dialog box](image)

The available settings are as follows.
<table>
<thead>
<tr>
<th><strong>Application Name</strong></th>
<th>Defines the XSLT1.0/2.0 file name prefix or the Java, C# or C++ application name for the generated transformation files.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base Package Name</strong></td>
<td>Defines the base package name for the Java output.</td>
</tr>
<tr>
<td><strong>Make paths absolute in generated code</strong></td>
<td>Defines whether the file paths should be relative or absolute in the generated program code, as well as in MapForce Server Execution files (mfx) and in mapping functions deployed to FlowForce Server. For more information, see <a href="#">About Paths in Generated Code</a>.</td>
</tr>
<tr>
<td><strong>Ensure Windows path convention for file path</strong></td>
<td>The &quot;Ensure Windows path convention...&quot; check box makes sure that Windows path conventions are followed. When outputting XSLT2 (and XQuery), the currently processed file name is internally retrieved using the document-uri function, which returns a path in the form file:// URI for local files. When this check box is active, a file:// URI path specification is automatically converted to a complete Windows file path (e.g. &quot;C:...&quot;) to simplify further processing.</td>
</tr>
<tr>
<td><strong>Line ends</strong></td>
<td>This combo box allows you to specify the line endings of the output files. &quot;Platform default&quot; is the specific default for the target operating system, e.g. Windows (CR+LF), macOS (LF), or Linux (LF). You can also select a specific line ending manually. The settings you select here are crucial when you deploy a mapping to FlowForce Server running on a different operating system.</td>
</tr>
</tbody>
</table>
| **XML Schema Version** | Lets you define the XML Schema Version used in the mapping file. You can define if you always want to load the Schemas conforming to version 1.0 or 1.1. Note that not all version 1.1 specific features are currently supported.  
If the xs:schema vc:minVersion="1.1" declaration is present, then version 1.1 will be used; if not, version 1.0 will be used.  
If the XSD document has no vc:minVersion attribute or the value of the vc:minVersion attribute is other than 1.0 or 1.1, then XSD 1.0 will be the default mode. |

Note: Do not confuse the vc:minVersion attribute with the xsd:version attribute. The former holds the XSD version number, while the latter holds the document version number.
### Common Tasks

| Web Service Operation Settings | The WSDL-Definitions, Service, Port and Operation fields are automatically filled if the mapping document is part of a Web service implementation. |

Changing this setting in an existing mapping causes a reloading of all schemas of the selected XML schema version, and might also change its validity.
4.2 Working with Components

Components are the central elements of any mapping design in MapForce. Generally, the term "component" is a convenient way to call any object which acts as a data source, or as a data target, or represents your data in the mapping at an intermediary processing stage.

There are two main categories of components: structure components and transformation components.

The structure components represent the abstract structure or schema of your data. For example, when you add an XML file to the mapping area (using the menu command Insert | XML Schema/File), it becomes a mapping component. For further information about structure components and their specifics, see Data Sources and Targets. With a few exceptions, structure components consist of items and sequences. An item is the lowest level mapping unit (for example, a single attribute in the XML file, or an element of simple type). A sequence is a collection of items.

The transformation components either transform data (for example, functions), or assist you in transformations (for example, constants or variables). For information on how you can use these components to achieve various data transformation tasks, see Designing Mappings.

With the help of structure components, you can either read data from files or other sources, write data to files or other sources, or store data at some intermediary stage in the mapping process (for example, in order to preview it). Consequently, structure components can be of the following types:

- **Source.** You declare a component as source by placing it on the left of the mapping area, and, thus, instructing MapForce to read data from it.
- **Target.** You declare a component as target by placing on the right of the mapping area, and, thus, instructing MapForce to write data to it.
- **Pass-through.** This is a special component type which acts both as a source and target (for further information, see Chained mappings / pass-through components).

On the mapping area, components appear as rectangles. The following sample mapping illustrates three source components, one target XML component, and various transformation components (functions and filters) through which data goes before being written to the source.
4.2.1 Searching within Components

To search for a specific node/item in a component:

1. Click the component you want to search in, and press the CTRL+F keys.
2. Enter the search term and click Find Next.
Use the Advanced options to define which items (nodes) are to be searched, as well as restrict the search options based on the specific connections.

### 4.2.2 Aligning Components

When you move components in the mapping pane, MapForce displays auto-alignment guide lines. These guide lines help you align a component to any other component in the mapping window.

In the sample mapping below, the lower component is being moved. The guide lines show that it can be aligned to the component on the left side of the mapping.
To enable or disable this option:

1. On the **Tools** menu, click **Options**.
2. In the **Editing** group, select the **Align components on mouse dragging** check box.

### 4.2.3 Changing the Component Settings

After you add a component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component settings dialog box in one of the following ways:

- Select the component and, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component header, and then click **Properties**.

Note that the available options depend on the type of the component. For reference to the settings applicable to each component type, see:

- XML Component Settings
- Database Component Settings
- CSV Component Settings
- Fixed-Length Field Component Settings
- FlexText Component Settings
- JSON Component Settings
- Excel 2007+ Component Settings
- XBRL Component Settings
- EDI Component Settings
- Binary Component Settings

For any file-based component, such as XML, a **File/String** button appears next to the root node. This button specifies advanced options applicable if you want to process or generate multiple files in a single mapping (see Processing Multiple Input or Output Files Dynamically). Additionally, it enables advanced options for parsing strings or serializing data to strings (see Parsing and Serializing Strings).

### 4.2.4 Duplicating Input

Sometimes, you may need to configure a component to accept data from more than one source. For example, you may need to convert data from two different XML schemas into a single schema. To make the destination schema accept data from both source schemas, you can duplicate any of the input items in the component. Duplicating input is meaningful only for a component which is a target component. On any given target component, you can duplicate as many items as required.

To duplicate a particular input item, right-click it and select **Add Duplicate Input After/Before** from the context menu.
In the image above, the item LineItem is being duplicated in order to provide the ability to map data from a second source.

Once you duplicate an input, you can make connections both to the original input and to the duplicate input. For example, this would enable you to copy data from source A to original input, and data from source B to the duplicate input.

**Note:** Duplication of XML attributes is not allowed, as it would make the resulting XML instance invalid. In case of XML elements, duplicating input is allowed regardless of the value of the element's `maxOccurs` attribute in the schema. This behaviour is intentional, since the schema could change later, or the source data could be optional. For example, a mapping could generate a single XML element, even if the input is duplicated on the mapping.

For a step-by-step example, see [Map Multiple Sources to One Target](#).
4.3 Working with Connections

A mapping is ultimately about transforming data from one format or structure into another. In a very basic mapping scenario, you add to the mapping area the components which represent your source and your target data (for example, a source XML schema and a destination one), and then draw visually the mapping connections between the two structure. A connection is, therefore, the visual representation of how data is mapped from a source to a destination.

Components have inputs and outputs which appear on the mapping as small triangles, called connectors. Input connectors are positioned to the left of any item to which you can draw a connection. Output connectors are positioned to the right of any item from which you can draw a connection.

To draw a connection between two items:

- Click the output connector of a source item and drag it to a destination item. When the drop action is allowed, a link tooltip appears next to the text cursor.

An input connector accepts only one incoming connection. If you try to add a second connection to the same input, a message box appears asking if you want to replace the connection with a new one or duplicate the input item. An output connector can have several connections, each to a different input.

To move a connection to a different item:

- Click the stub of the connection (the straight section closer to the target) and drag it to the destination.

To copy a connection to a different item:

- Click the stub of the connection (the straight section closer to the target), and drag it to the destination while holding down the Ctrl key.
To view the item(s) at the other end of a connection:

- Point to the straight section of a connection (close to the input/output connector). A tooltip appears which displays the name(s) of the item(s) at the other end of the connection. If multiple connections have been defined from the same output, then a maximum of ten item names are displayed. In the sample below, the two target items are **SinglePrice** and **value2** of the multiply function.

![Diagram showing connections and tooltips](image)

To change the connection settings, do one of the following:

- On the **Connection** menu, click **Properties** (this menu item becomes enabled when you select a connection).
- Double-click the connection.
- Right-click the connection, and then click **Properties**.

See also [Connection Settings](#).

To delete a connection, do one of the following:

- Click the connection, and then press the **Delete** key.
- Right-click the connection, and then click **Delete**.

### 4.3.1 About Mandatory Inputs

To aid you in the mapping process, MapForce highlights in orange the mandatory inputs in target components:

- In XML and EDI components, these are items where the minOccurs parameter is equal/greater than 1.
- In databases, these are fields that have been defined as "not null"
- WSDL calls and WSDL response (all nodes)
- XBRL nodes that have been defined as mandatory
- In functions, these are the specific mandatory parameters such that once one parameter has been mapped, then the other mandatory ones will be highlighted to show that a connection is needed. E.g. once one of the filter input parameters is mapped, then the other one is automatically highlighted.
- Worksheet names in MS Excel sheets
Example
When creating a mapping like CompletePO.mfd, available in the ...\MapForceExamples folder, the inserted XML Schema files exist as shown below.

The Number element of the Customers component is then connected to the Number element of the CompletePO component. As soon as the connection has been made, the mandatory items/nodes of the CompletePO component are highlighted. Note that the collapsed "Article" node/icon is also highlighted.
4.3.2 Changing the Connection Display Preferences

You can selectively view the connections in the mapping window.

- **Show selected component connectors** switches between showing:
  - all mapping connectors in black, or
  - those connectors relating to the currently selected component in black. Other connectors appear dimmed.

- **Show connectors from source to target** switches between showing:
  - connectors that are directly connected to the currently selected component, or
  - connectors linked to the currently selected component, originating from source and terminating at the target components.

4.3.3 Annotating Connections

Individual connections can be labeled allowing you to comment your mapping in great detail. This option is available for all connection types.

**To annotate to a connection:**

1. Right-click the connection, and select **Properties** from the context menu.
2. Enter the name of the currently selected connection in the **Description** field. This enables all the options in the Annotation Settings group.
3. Use the remaining groups to define the starting location, alignment and position of the label.
4. Activate the **Show annotations** icon in the View Options toolbar to see the annotation text.

**Note:** If the **Show annotations** icon is inactive, you can still see the annotation text if you place the mouse cursor over the connection. The annotation text will appear in a callout if the **Show tips** toolbar button is active in the View Options toolbar.

4.3.4 Connection Settings

Right-clicking a connection and selecting **Properties** from the context menu, or double-clicking a connection, opens the Connection Settings dialog box in which you can define the settings of the current connection. Note that unavailable options are disabled.
For items of `complexType`, you can choose one of the following connection types for mapping (note that these settings also apply to `complexType` items which do not have any text nodes):

<table>
<thead>
<tr>
<th>Target Driven (Standard)</th>
<th>Changes the connection type to &quot;Target-driven&quot; (see Target-driven / Standard mapping).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy-all (Copy child items)</td>
<td>Changes the connection type to &quot;Copy-all&quot; and automatically connects all identical items in the source and target components (see Copy-all connections).</td>
</tr>
<tr>
<td>Source Driven (mixed content)</td>
<td>Changes the connection type to &quot;Source-driven&quot;, and enables the selection of additional elements to be mapped. The additional elements must be child items of the mapped item in the XML source file, to qualify for mapping. Activating the Map Processing Instructions or Map Comments check boxes enables you to include these data groups in the output file.</td>
</tr>
</tbody>
</table>
The Annotation Settings group enables you to annotate the connection (see Annotating Connections).

### 4.3.5 Connection Context Menu

When you right-click a connection, the following context commands are available.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connect Matching Children</strong></td>
<td>Opens the &quot;Connect Matching Children&quot; dialog box (see Connecting Matching Children). This command is enabled when the connection is eligible to have matching children.</td>
</tr>
<tr>
<td><strong>Delete</strong></td>
<td>Deletes the selected connection.</td>
</tr>
<tr>
<td><strong>Go to source:</strong> <code>&lt;item name&gt;</code></td>
<td>Selects the source connector of the current connection.</td>
</tr>
<tr>
<td><strong>Go to target:</strong> <code>&lt;item name&gt;</code></td>
<td>Selects the target connector of the current connection.</td>
</tr>
<tr>
<td><strong>Target Driven (Standard)</strong></td>
<td>Changes the connection type to &quot;Target-driven&quot; (see Target-driven connections).</td>
</tr>
</tbody>
</table>

![Connection Context Menu Diagram]
Copy-All (Copy Child Items) | Changes the connection type to "Copy-all" and automatically connects all identical items in the source and target components (see Copy-all connections).

   This command is enabled (and meaningful) when both the source item and the target item have children items.

Source Driven (Mixed Content) | Changes the connection type to "Source-driven" (see Source-driven connections).

   This command is enabled (and meaningful) when both the source item and the target item have children items.

Insert Sort: Nodes/Rows | Adds a Sort component between the source and the target item (see Sorting Data).

Insert Filter: Nodes/Rows | Adds a Filter component between the source and the target item (see Filters and Conditions).

Insert SQL-Where Condition | Adds a SQL-Where component between the source and the target item (see SQL WHERE / ORDER Component).

Insert Value-Map | Adds a Value-Map component between the source and the target item (see Using Value-Maps).

Properties | Opens the Connections Settings dialog box (see Connection Settings).

4.3.6 Connecting Matching Children

You can create multiple connections between items of the same name in both the source and target components. Note that a "Copy-all" connection (see Copy-all connections) is created by default.

To toggle the "Auto Connect Matching Children" option on or off, do one of the following:

- Click the Auto Connect Matching Children ( toolbar button.
- On the Connection menu, click Auto Connect Matching Children.

To change the settings for "Connect Matching Children":

1. Connect two (parent) items that share identically named child items in both components.
2. Right click the connection and select the Connect matching child elements option.
3. Select the required options (see the table below), and click OK. Connections are created for all the child items that have identical names and adhere to the settings defined in the dialog box.

**Note:** The settings you define here are applied when connecting two items if the **Toggle auto connect of children** ( ) toolbar button is active.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ignore Case</td>
<td>Ignores the case of the child item names.</td>
</tr>
<tr>
<td>Ignore Namespaces</td>
<td>Ignores the namespaces of the child items.</td>
</tr>
<tr>
<td>Recursive</td>
<td>Creates new connections between any matching items recursively. That is, a connection is created no matter how deep the items are nested in the hierarchy, as long as they have the same name.</td>
</tr>
<tr>
<td>Mix Attributes and Elements</td>
<td>When enabled, allows connections to be created between attributes and elements which have the same name. For example, a connection is created if two &quot;Name&quot; items exist, even though one is an element, and the other is an attribute.</td>
</tr>
<tr>
<td>Create copy-all connections</td>
<td>This setting is active by default. It creates (if possible) a connection of type &quot;Copy-all&quot; between source and target items.</td>
</tr>
<tr>
<td>Ignore existing output connections</td>
<td>Creates additional connections for any matching items, even if they already have outgoing connections.</td>
</tr>
<tr>
<td>Retain</td>
<td>Retains existing connections.</td>
</tr>
<tr>
<td>Overwrite</td>
<td>Recreates connections according to the settings defined. Existing connections are discarded.</td>
</tr>
</tbody>
</table>
Common Tasks

Working with Connections

Delete all existing

Deletions all existing connections, before creating new ones.

Deleting connections

Connections that have been created using the Connect Matching Children dialog, or during the mapping process, can be removed as a group.

To delete connections:

1. Right-click the item name in the component, not the connection itself ("Person" in this example).
2. Select Delete Connections | Delete all connections.

| Delete all direct connections | Deletes all connections directly mapped to, or from, the current component to any other source or target components. |
| Delete all incoming child connections | Only active if you have right clicked an item in a target component. Deletes all incoming child connections. |
| Delete all outgoing child connections | Only active if you have right clicked an item in a source component. Deletes all outgoing child connections. |

4.3.7 Notifications on Missing Parent Connections

When you create connections between source and target items manually, MapForce automatically analyzes the possible mapping outcomes. If you are mapping two child items, a notification message can appear suggesting that you also connect the parent of the source item with the parent in the target item.

This notification message helps you prevent situations where a single child item appears in the Output window when you preview the mapping. This will generally be the case if the source node supplies a sequence instead of a single value.

To understand how this works, open the sample mapping Tut-OrgChart.mfd available in the <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ folder. If you connect the
source text() item to the target text() item, a notification message appears, stating that the parent item "para" is not connected and will only be generated once in the output.

4.3.8 Moving Connections and Child Connections

When you move a connection to a different component, MapForce automatically matches identical child connections and will prompt you whether it should move them to the new location as well. A common use of this feature is if you have an existing mapping and then change the root element of the target schema. Normally, when this happens, you would need to remap all descending connections manually. This feature helps you prevent such situations.

This example uses the Tut-ExpReport.mfd file available in the <Documents>\Altova
To understand how it works, do the following:

1. Open the **Tut-ExpReport.mfd** sample mapping.
2. Edit the **ExpReport-Target.xsd** schema outside MapForce so as to change the Company root element of the target schema to Company-EU. You do not need to close MapForce.
3. After you have changed the Company root element of the target schema to Company-EU, a "Changed files" prompt appears in MapForce.
4. Click the **Reload** button to reload the updated Schema. Since the root element was deleted, the component displays multiple missing nodes.

5. Click **Select new root element** at the top of the component. (You can also change the root element by right clicking the component header and selecting **Change Root Element** from the context menu.)

6. Select **Company-EU** as new root element and click OK to confirm. The **Company-EU** root element is now visible at the top of the component.
7. Click the target stub of the connection that exists between the expense-report item of the source component and the Company item of the target component, and then drag-and-drop it on the Company-EU root element of the target component.

A notification dialog box appears.

8. Click Include descendent connections. This instructs MapForce to re-map the correct child items under the new root element, and the mapping becomes valid again.

**Note:** If the node to which you are mapping has the same name as the source node but is in a different namespace, then the notification dialog box will contain an additional button: "Include descendants and map namespace". Clicking this button moves the child connections of the same namespace as the source parent node to the same child nodes under the different namespace node.

### 4.3.9 Keeping Connections After Deleting Components

You can decide what happens when you delete a component that has multiple (child) connections to another component, e.g. a filter or sort component. This is very useful if you want to keep all the child connections and not have to restore each one individually.

You can opt to keep/restore the child connections after the component is deleted, or to delete all child connections immediately.

Select Tools | Options | Editing (tab) to see the current setting. The default setting for the check box is **inactive**, i.e. "Smart component deletion (keep useful connections)" is disabled.
E.g. using the CompletePO.mfd mapping in the ...\MapForceExamples folder, and the check box is active, the Customer filter is a **copy-all** connection with many connected child items, as shown below.

Deleting the Customer filter opens a prompt asking if you really want to delete it. If you select Yes, then the filter is deleted but all the child connectors remain.
Note that the remaining connectors are still selected (i.e. shown in red). If you want to delete them as well, hit the Del. key.

Clicking anywhere in the mapping area deselects the connectors.

If the "Smart component deletion..." check box is inactive, then deleting the filter will delete all child connectors immediately.

**Note:** If a filter component has both "on-true" and "on-false" outputs connected, then the connectors for both outputs will be retained.

## 4.3.10 Dealing with Missing Items

Over time, it is likely that the structure of one of the components in a mapping may change e.g. elements or attributes are added/deleted to an XML schema. MapForce uses placeholder items to retain all the connectors, and any relevant connection data between components, when items have been deleted.

**Example**

Using the `MFCompany.xsd` schema file as an example. The schema is renamed to `MyCompany.xsd` and a connector is created between the Company item in both schemas. This creates connectors for all child items between the components, if the Autoconnect Matching Children is active.
While editing MyCompany.xsd, in XMLSpy, the First and Last items in the schema are deleted. Returning to MapForce opens a Changed Files notification dialog box, prompting you to reload the schema. Clicking **Reload** updates the components in MapForce.

The deleted items and their connectors are now marked in the MyCompany component. You could now reconnect the connectors to other items if necessary, or delete the connectors.

Note that you can still preview the mapping (or generate code), but warnings will appear in the Messages window if you do so at this point. All connections to, and from, missing items are ignored during preview or code-generation.

Clicking one of the highlighted connectors and deleting it, removes the "missing" item from the component, e.g. Last, in MyCompany.
Renamed items
If a parent item is renamed e.g. Person to ZPerson, then the original parent item connector is retained and the child items and their connectors are deleted.

"Copy all" connectors and missing items
Copy all connections are treated in the same way as normal connections, with the only difference being that the connectors to the missing child items are not retained or displayed.
Renamed or deleted component sources

If the data source of a component i.e. schema, database etc. has been renamed or deleted, then all items it contained are highlighted. The red frame around the component denotes that there is no valid connection to a schema or database file and prevents preview and code generation.

Placing the mouse cursor over the highlighted component, opens a popup containing pertinent information.
Double-click the title bar of the highlighted component to open the Component Settings dialog box and select a different schema, see also Changing the Component Settings.

Clicking the **Change** button in the dialog box that opens if the component is a database, allows you to select a different database, or change the tables that appear in the database component. Connectors to tables of the same name will be retained.

All valid/correct connections (and relevant database data, if the component is a database) will be retained if you select a schema or database of the same structure.
4.4 Working with Mapping Projects

In addition to creating standalone mappings, you can also create mapping projects that include multiple mappings. Mappings added to a project are easily accessible from the Project window.

![Project window (MapForce Enterprise Edition)](image)

The main advantage of projects is that you can define common code generation settings (such as the target language and the output directory) for all the mapping files included in that particular project. You can also create folders inside projects, and specify custom code generation settings for each individual folder in a project. For more information about the MapForce-generated program
code (in C++, C#, and Java), see Code Generator.

In MapForce Enterprise edition, you can additionally create Web Service projects. Such projects enable you to generate Java or C# program code that implements SOAP Web services, based on existing Web Services Description Language (WSDL) files. For further information about Web Service projects, see Implementing Web Services.

4.4.1 Opening, Searching, and Closing Projects

MapForce project files have the *.mfp extension. You can open existing MapForce projects in the same way as you open mappings (on the File menu, click Open).

When a mapping project is opened in MapForce, the Project window shows all files and folders that have been added to the project. By default, when you run MapForce for the first time, it loads the MapForceExamples.mfp project in the Project window.

To search for files within a project:

1. In the Project window, click the project or the folder to be searched.
2. Press Ctrl + F.
3. Optionally, select your search options. For example, if you want to include folder names in the search, select the Find in folder names option.

To close a project:

- On the Project menu, click Close Project.

4.4.2 Creating a New Project

To create a new project:

1. On the File menu, click New.
2. Select Project File, and then click OK.
3. Enter the project name in the **Save Project As** dialog box, and click **Save**. The new project is now displayed in the Project window.

You can now add mappings to the project.

**To add the currently active mapping to the project, do one of the following:**

- On the **Project** menu, click **Add Active File to Project**.
- Right-click the project, and select **Add Active File to Project**.

**To add existing mapping files to the project, do one of the following:**

- On the **Project** menu, click **Add Files to Project**.
- Right-click the project, and select **Add Files to Project**.

**Tip:** To open multiple files, hold the **Ctrl** key while selecting the files in the Open dialog box.

**To remove a file or folder from a project, do one of the following:**

- Right-click the file in the Project window, and select **Delete** from the context menu.
- Select the file in the Project window, and press **Delete**.
### 4.4.3 Setting the Code Generation Settings

For any project, you can specify code generation settings that will affect all the mappings inside a project. To open the **Project Settings** dialog box, do one of the following:

- Right-click the project name in the **Project** window and choose **Properties** from the context menu.
- On the **Project** menu, click **Properties**.

![Project Settings dialog box](image)

The available settings are as follows. Note that the project name and the project directory cannot be changed after the project has been created.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output name</strong></td>
<td>The value entered here determines the name of the generated project or solution, as well as other object names in the generated code.</td>
</tr>
<tr>
<td><strong>Output directory</strong></td>
<td>Defines the Windows folder where the generated code (from all mappings in this project) will be saved. By default, output is saved to the <code>output/</code> directory located in the project directory.</td>
</tr>
<tr>
<td><strong>Language</strong></td>
<td>Defines the code generation language for all mapping files in this project.</td>
</tr>
<tr>
<td><strong>Base package name</strong></td>
<td>This setting is applicable if you selected Java as transformation language. It defines the name of the base package in the generated Java project.</td>
</tr>
</tbody>
</table>
4.4.4 Managing Project Folders

If you want to organize the mappings inside a project into folders, you can create as many folders as required, and add mappings to (or drag mappings into) them. Such folders are “virtual” and meaningful only inside a MapForce project; they do not correspond to actual folders on your operating system. One of the advantages of creating folders is that you can define common code generation settings (such as the target language and the output directory) for all the mapping files under that particular folder.

To create a folder inside a MapForce project:

1. Do one of the following:
   - On the Project menu, click Create Folder.
   - Right-click the project, and select Create Folder.
2. In the Properties dialog box, enter the required code generation settings, and click OK.

The settings you can define in the Folder Properties dialog box are as follows.

<table>
<thead>
<tr>
<th>Name</th>
<th>The name of the folder.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use default project settings</td>
<td>This is the default option and it means that the code generation settings in the current folder are the same as for the entire project. Therefore, when you generate code from you project, MapForce will use the code generation settings defined at the project level, not at the folder level. If your folder requires custom code generation settings (other than those set at the project level), select Use the</td>
</tr>
</tbody>
</table>
Following settings and specify the code output directory and language as required.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output directory</td>
<td>Defines the Windows folder where the generated code (from all mappings in this folder) will be saved.</td>
</tr>
<tr>
<td>Language</td>
<td>Defines the code generation language for all mapping files in this folder.</td>
</tr>
</tbody>
</table>
Chapter 5
Designing Mappings
5  Designing Mappings

Altova website: Data integration tool

This section describes how to design data mappings, and ways in which you can transform data on the mapping area. It also includes various considerations applicable to mapping design. Use the following roadmap for quick access to specific tasks or concepts:

<table>
<thead>
<tr>
<th>I want to...</th>
<th>Read this topic...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create or edit path references to miscellaneous schema, instance, and other files used by a mapping.</td>
<td>Using Relative and Absolute Paths</td>
</tr>
<tr>
<td>Fine-tune the data mapping for specific needs (for example, influence the sequence of items in a target component).</td>
<td>Connection Types</td>
</tr>
<tr>
<td>Use the output of a component as input of another component.</td>
<td>Chained mappings / pass-through components</td>
</tr>
<tr>
<td>Process multiple files (for example, all files within a directory) in the same mapping, either as a source or a target.</td>
<td>Processing Multiple Input or Output Files Dynamically</td>
</tr>
<tr>
<td>Pass an external value (such as a string parameter) to the mapping.</td>
<td>Supplying Parameters to the Mapping</td>
</tr>
<tr>
<td>Get a string value out of the mapping, instead of a file.</td>
<td>Returning String Values from a Mapping</td>
</tr>
<tr>
<td>Store some mapping data temporarily for later processing (similar to variables in a programming language).</td>
<td>Using Variables</td>
</tr>
<tr>
<td>Sort data in ascending or descending order.</td>
<td>Sorting Data</td>
</tr>
<tr>
<td>Filter nodes/rows based on specific criteria, or process values conditionally.</td>
<td>Filters and Conditions</td>
</tr>
<tr>
<td>Merge or join data from multiple sources with different schema.</td>
<td>Joining Data</td>
</tr>
<tr>
<td>Merging Data from Multiple Schemas</td>
<td>Merging Data from Multiple Schemas</td>
</tr>
<tr>
<td>Process key-value pairs, for example, to convert months from numerical representation (01, 02, and so on) to text representation (January, February, and so on).</td>
<td>Using Value-Maps</td>
</tr>
<tr>
<td>Configure a mapping to return an error when a specific condition occurs.</td>
<td>Adding Exceptions</td>
</tr>
<tr>
<td>Convert complex mapping structures to string data type, and vice versa.</td>
<td>Parsing and Serializing Strings</td>
</tr>
</tbody>
</table>
I want to... | Read this topic...
---|---
Learn how to avoid undesired results when designing complex mappings. | Mapping rules and strategies

Importantly, MapForce additionally includes an extensive built-in function library (see Function Library Reference) to help you with a wide array of processing tasks. When the built-in library is not sufficient, you can always build your own custom functions in MapForce, or re-use external XSLT files, as well as .dll or Java .class libraries. For further information, see Using Functions.
5.1 Using Relative and Absolute Paths

A mapping design file (*.mfd) may have references to several schema and instance files. The schema files are used by MapForce to determine the structure of the data to be mapped, and to validate it. The instance files, on the other hand, are required to read, preview, and validate the source data against the schema.

Mappings may also include references to StyleVision Power Stylesheets (*.sps) files, used to format data for outputs such as PDF, HTML and Word. Also, mappings may have references to file-based databases such as Microsoft Access or SQLite.

All references to files used by a mapping design are created by MapForce when you add a component to the mapping. However, you can always set or change such path references manually if required.

This section provides instructions for setting or changing the path to miscellaneous file types referenced by a mapping, and the implications of using relative versus absolute paths.

5.1.1 Using Relative Paths on a Component

The Component Settings dialog box (illustrated below for an XML component) provides the option to specify either absolute or relative paths for various files which may be referenced by the component:

- Input files (that is, files from which MapForce reads data)
- Output files (that is, files to which MapForce writes data)
- Schema files (applicable to components which have a schema)
- Structure files (applicable to components which may have a complex structure, such as input or output parameters of user-defined functions, or variables)
- StyleVision Power Stylesheet (*.sps) files, used to format data for outputs such as PDF, HTML and Word.
- Database trace files, in case of database components

You can enter relative paths directly in the relevant text boxes (shown enclosed in a red frame in the image below).

Before entering relative file paths, make sure to save the mapping file (.mfd) first. Otherwise, all relative paths are resolved against the personal application folder of Windows (Documents \Altova\MapForce2019), which may not be the intended behavior.

You can also instruct MapForce to save all above-mentioned file paths relative to the mapping .mfd file. In the sample image below, notice the option Save all file paths relative to MFD file. If the check box is enabled (which is the default and recommended option), the paths of any files referenced by the component will be saved relative to the path of the mapping design file (.mfd). This affects all files referenced by the component (shown enclosed in a red frame in the image).
Although the component illustrated above is an XML component, the setting **Save all file paths relative to MFD file** works in the same way for the following files:

- Structure files used by complex input or output parameters of user-defined functions and
variables of complex type
- Input or output flat files *
- Schema files referenced by database components which support XML fields *
- Database trace files *
- Input or output XBRL, FlexText, EDI, Excel 2007+, JSON files **

* MapForce Professional and Enterprise Edition
** MapForce Enterprise Edition only

Taking the component above as an example, if the .mfd file is in the same folder as the books.xsd and books.xml files, the paths will be changed as follows:

C:\Users\altova\Documents\MyMapping\books.xsd will change to books.xsd
C:\Users\altova\Documents\MyMapping\books.xml will change to books.xml

Paths that reference a non-local drive or use a URL will not be made relative.

When the check box is selected, MapForce will also keep track of the files referenced by the component if you save the mapping to a new folder using the Save as menu command. Also, if all files are in the same directory as the mapping, path references will not be broken when you move the entire directory to a new location on the disk.

Using relative paths (and, therefore, enabling the Save all file paths relative to MFD file check box) may be important in many cases, for example:

- The location of the mapping on your operating system is likely to change in future.
- The mapping is stored in a directory which is under source control (using a version control system such as TortoiseSVN, for example).
- You intend to deploy the mapping for execution with MapForce Server to a different machine or even to a different operating system.

If the Save all file paths relative to MFD file check box is not selected, saving the mapping does not modify the file paths (that is, they remain as they appear in the Component Settings dialog box).

### 5.1.2 Setting the Path to File-Based Databases

When you add a database file such as Microsoft Access or SQLite to the mapping (see Starting the Database Connection Wizard), you can use a relative path instead of an absolute one. To use a relative path, enter the required relative path instead of clicking Browse in the Database Connection Wizard.

Before entering relative file paths, make sure to save the mapping file (.mfd) first. Otherwise, all relative paths are resolved against the personal application folder of Windows (Documents \Altova\MapForce2019), which may not be the intended behavior.
If the database is a SQLite database, the Connect button becomes enabled if the following is true:

- The path points to a file that can be resolved relatively to the mapping (.mfd) file
- The referenced file is a SQLite database.

To change the path of a database component which is already in the mapping, do the following:

1. Right-click the header of the database component, and select Properties (see also Changing the Component Settings). Alternatively, double-click the component title bar.
2. On the Component Settings dialog box, click Change.
This re-opens the Database Connection Wizard, from where you can change the database connection properties (including the path) as already shown above.

**Note:** When you generate program code, or when you compile MapForce Server execution files (.mfx), or when you deploy the mapping to FlowForce Server, a relative path will be converted to an absolute path if the check box **Make paths absolute in generated code** is selected from the mapping settings (see **About Paths in Generated Code**).

### 5.1.3 Fixing Broken Path References

When you add or change a file reference in a mapping, and the path cannot be resolved, MapForce displays a warning message. This way, MapForce diminishes the chance for broken path references to happen. Nevertheless, broken path references may still occur in cases such as:

- You use relative paths, and then move the mapping file to a new directory without moving the schema and instance files.
- You use absolute paths to files in the same directory as the mapping file, and then move the directory to another location.

When this happens, MapForce highlights the component in red, for example:

![Broken path reference](image)

The solution in this case is to double-click the component header and update any broken path references in the **Component Settings** dialog box (see also **Changing the Component Settings**).
5.1.4 Paths in Various Execution Environments

If you generate code from mappings, compile mappings to MapForce Server execution files (.mfx), or deploy mappings to FlowForce Server, the generated files are no longer run by MapForce. Instead, the mappings are run by the target environment you have chosen (for example, RaptorXML Server, MapForce Server, or a C# application). The implication is that, for the mapping to run successfully, any relative paths must be meaningful in the environment where the mapping runs.

Consequently, when the mapping uses relative paths to instance or schema files, consider the base path to be as follows for each target language:

<table>
<thead>
<tr>
<th>Target language</th>
<th>Base path</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLT/XSLT2</td>
<td>Path of the XSLT file.</td>
</tr>
<tr>
<td>XQuery*</td>
<td>Path of the XQuery file.</td>
</tr>
<tr>
<td>C++, C#, Java*</td>
<td>Working directory of the generated application.</td>
</tr>
<tr>
<td>BUILT-IN* (when previewing the mapping in MapForce)</td>
<td>Path of the mapping (.mfd) file.</td>
</tr>
<tr>
<td>BUILT-IN* (when running the mapping with MapForce Server)</td>
<td>The current working directory.</td>
</tr>
<tr>
<td>BUILT-IN* (when running the mapping with MapForce Server under FlowForce Server control)</td>
<td>The working directory of the job or the working directory of FlowForce Server.</td>
</tr>
</tbody>
</table>

* Languages available in MapForce Professional and Enterprise editions

If required, you can instruct MapForce to convert all paths from relative to absolute when generating code for a mapping. This option might be useful if you run the mapping code (or the MapForce Server execution file) on the same operating system, or perhaps on another operating system where any absolute path references used by the mapping can still be resolved.

To convert all paths to absolute in the generated code, right-click the mapping, and select the **Make paths absolute in generated code** check box. See also [Changing the Mapping Settings](#).

When you generate code and the check box is selected, MapForce resolves any relative paths based on the directory of the mapping file (.mfd), and makes them absolute in the generated code. This setting affects the path of the following files:

- Input and output instance files for all file-based component kinds
- Access and SQLite database files used as mapping components

When the check box is not selected, the file paths will be preserved as they are defined in the component settings.
5.1.5 Copy-Paste and Relative Paths

When you copy a component from a mapping and paste it into another, a check is performed to ensure that relative paths of schema files can be resolved against the folder of the destination mapping. If the path cannot be resolved, you will be prompted to make the relative paths absolute by means of the folder of the source mapping. It is recommended to save the destination mapping first, otherwise relative paths are resolved against the personal application folder.
5.2 Connection Types

When you create a mapping connection (and both the source and the target item have child items), you can optionally choose the type of the connection to be one of the following.

- Target Driven (Standard)
- Source Driven (Mixed Content)
- Copy-All (Copy Child Items).

The connection type determines the sequence of children items in the output generated by the mapping. This section provides information about each connection type and the scenarios when they are useful.

5.2.1 Target-driven connections

When a connection is "target-driven" (or "standard"), the sequence of child nodes in the mapping output is determined by the sequence of nodes in the target schema. This connection type is suitable for most mapping scenarios and is the default connection type used in MapForce.

On a mapping, target-driven connections are shown with a solid line.

Target-driven connections might not be suitable when you want to map XML nodes that contain mixed context (character data as well as child elements), for example:

<
<p>This is our <i>best-selling</i> product.</p>

With mixed content, it is likely that you want to preserve the sequence of items as they appear in the source file, in which case a source-driven connection is recommended (see Source-driven connections ).

5.2.2 Source-driven connections

Source-driven (Mixed Content) mapping enables you to automatically map text and child nodes in the same sequence that they appear in the XML source file.

- Mixed content text node content is supported/mapped.
- The sequence of child nodes is dependent on the source XML instance file.
Mixed content mappings are shown with a dotted line.

Source-driven / mixed content mapping can also be applied to XML schema **complexType** items. Child nodes will then be mapped according to their sequence in the XML source file.

Source-driven / mixed content mapping supports:

Mappings from

- As **source** components:
  - XML schema complexTypes (including mixed content, i.e. mixed=true)
  - XML schema complexTypes (including mixed content) in embedded schemas of a database field
- As **target** components:
  - XML schema complexTypes (including mixed content),
  - XML schema complexTypes (including mixed content) in embedded schemas of a database field

Note: CDATA sections are treated as text.

### 5.2.2.1 Mapping mixed content

The files used in the following example (Tut-OrgChart.mfd, Tut-OrgChart.mfd.xml, Tut-OrgChart.mfd.xsd, Tut-Person.xsd) are available in the `...\MapForceExamples\Tutorial\` folder.

Source XML instance

A portion of the **Tut-OrgChart.xml** file used in this section is shown below. Our area of concern is the mixed content element "para", along with its child nodes "bold" and "italic".

The **para** element also contains a Processing Instruction (`<?sort alpha-ascending?>`) as well as Comment text (`<!--Company details... -->`) which can also be mapped, as shown below.
Note the sequence of the text and bold/italic nodes in the XML instance file:

<para> The company...
   <bold>Vereno</bold> in 1995 ...
   <italic>multi-core</italic> February 1999
   <bold>Nano-grid.</bold>The company ...
   <italic>offshore</italic> to drive ...
</para>

Initial mapping

The initial state of the mapping when you open Tut-Orgchart.mfd is shown below.

Output of above mapping

The result of the initial mapping is shown below: Organization Chart as well as the individual office names have been output.
Mapping the para element

The image below shows an example of mixed content mapping. The para element is of mixed content, and the connector is shown as a dotted line to highlight this. The text() node contains the textual data and needs to be mapped for the text to appear in the target component.

To annotate (add a label to) any connection, right-click it and select Properties (see Annotating Connections).

The image below shows the content model of the Description element (Desc) of the Tut-OrgChart.xsd schema file. This definition is identical in both the source and target schemas used in this example.

Note the following properties of the para element in the Content model:

- para is a complexType with mixed="true", of type "TextType"
- bold and italic elements are both of type "xs:string", they have not been defined as recursive in this example, i.e. neither bold, nor italic are of type "TextType"
- bold and italic elements can appear any number of times in any sequence within para
- any number of text nodes can appear within the para element, interspersed by any
To create mixed content connections between items:

1. Select the menu option **Connection | Auto Connect Matching Children** to activate this option, if it is not currently activated.
2. Connect the *para* item in the source schema, with the *para* item in the target schema. A message appears, asking if you would like MapForce to define the connectors as source driven.

![MapForce interface](image)

3. Click Yes to create a mixed content connection.

**Note:** *Para* is of mixed content, and makes the message appear at this point. The mixed-content message also appears if you only map the para items directly, without having the autoconnect option activated.

All child items of *para* have been connected. The connector joining the *para* items is displayed as a dotted line, to show that it is of type mixed content.

4. Click the Output tab to see the result of the mapping.

```xml
1  <xml version="1.0" encoding="UTF-8">  
2  <Company-Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames="true">  
3      <Name>Organization Chart</Name>  
4      <Office>  
5          <Name>Nanonull, Inc.</Name>  
6              <Desc>  
7                  <para>The company was established in <bold>Veneto</bold> in 1995. Nanonull develops</para>  
8                      <para>White papers and further information will be made available in the near future.</para>  
9                  </Desc>  
10              </Office>  
11      </Office>  
12      <Office>  
13          <Name>Nanonull Europe, A0</Name>  
14              <Desc>  
15                  <para>In May 2000, Nanonull <italic>Europe</italic> was set up in Vienna. The team of</para>  
16                      <Desc>  
17                  </Desc>  
18              </Office>  
19      </Office-Person>  
20  </Company-Person>
```

5. Click the word **Wrap** icon in the Output tab icon bar, to view the complete text in the Output window.
The mixed content text of each office description has been mapped correctly; the text, as well as the bold and italic tag content, have been mapped as they appear in the XML source file.

6. Switch back to the Mapping view.

To remove text nodes from mixed content items:

1. Click the text() node connector and press Del. to delete it.

2. Click the Output tab to see the result of the mapping.
Result:
- all **text** nodes of the para element have been removed.
- mapped bold and italic text content remain
- the bold and italic item **sequence** still follows that of the source XML file.

**To map the Processing Instructions and Comments:**

1. Right-click the mixed content connection, and select **Properties**.
2. Under **Source-Drive (Mixed content)**, select the **Map Processing Instructions** and **Map Comments** check boxes.

### 5.2.2.2 Mixed content example

The following example is available as "ShortApplicationInfo.mfd" in the ...\MapForceExamples folder.

A snippet of the XML source file for this example is shown below.

```xml
<Page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="SectionedPage.xsd">
  <Item>
    <Title>XMLSpy</Title>
    <MainSection author="altova">Altova <Trademark>XMLSpy</Trademark> 2005 Inter
is the industry standard <Keyword>XML</Keyword> development environment editing, debugging and transforming all <Keyword>XML</Keyword> technolo
gically generating runtime code in multiple programming languages</MainSection>
  </Item>
</Page>
```

The mapping is shown below. Please note the following:
Designing Mappings Connection Types

- The "SubSection" item connector is of mixed content, and is mapped to the Description item in the target XML/schema.
- The text() nodes are mapped to each other
- Trademark text is mapped to the Bold item in the target
- Keyword text is mapped to the Italic item in the target

**Mapping result**
The mixed content text of each description has been mapped correctly; the text, as well as the bold and italic tag content, have been mapped as they appear in the XML source file.

```xml
<!?xml version="1.0" encoding="UTF-8"?>
<ShortInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:NoNamespaceSchemaLocation="C:\PROGRA~1\Altova\MapForce2005\MapForceExamples\ShortInfo.xsd">
  <Info>
    <Title>XMLSpy</Title>
    <Description>Altova XMLSpy</Description>
  </Info>
</ShortInfo>
```

5.2.2.3 **Using standard connections on mixed content items**

As mentioned before, source-driven (not standard) connections are normally used when mapping data from mixed content nodes. Otherwise, the resulting output may be undesirable. To see the consequences of using a standard (target-driven) connection when mapping data from a mixed content node, follow the steps below:

1. Open the mapping **Tut-OrgChart.mfd** from the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial` folder.
2. Create a connection between the `para` node in the source and the `para` node in the
target. A message appears, asking if you would like MapForce to define the connections as source-driven. Click No (this disregards the MapForce suggestion and creates a standard connection).

![Diagram of connection types]

**Note:** Make sure that the connection is standard (target-driven), as shown above. If a Copy-All connection is created automatically, right-click the connection, and select **Target Driven (Standard)** from the context menu.

3. Click the **Output** tab to see the result of the mapping.

```xml
<Office>
  <Name>Nanonull, Inc.</Name>
  <Desc>
    <para>The company was established in 1995. Nanonull develops nanoelectronic technology. An unveiling of the first prototype in 1995, Nanonull has expanded its operations to drive down operations through innovation in technology. It features a bold Verreno Nano-grid. It also features multi-core processors and offshore production.</para>
    <para>White papers and further information will be made available in the near future.</para>
  </Desc>
</Office>
```

As illustrated above, mapping mixed content nodes using standard connections produces the following result:

- The content of the `text()` source item is copied to the target; however, the sequence of child nodes (bold and italic, in this case) in the output corresponds to the sequence in the target XML schema. In other words, the child nodes (bold and italic, in this case) appear after the mixed content node text.
- For each `para` element, MapForce has mapped the `text()` node first, then all bold items, and, finally, all italic items. As a result, multiple bold and italic items appear stacked on each other. Note that the content of each item is mapped if a connection exists to it from the source.

### 5.2.3 Copy-All Connections

Copy-All connections map data between complex structures (nodes with children items) that are very similar or identical. The main benefit of "Copy-All" connections is that they simplify the mapping workspace (one "thick" connection is created instead of multiple).
On the mapping, a "Copy-All" connection appears as a single bold line (with input and output "forks" for each child item) that connects two identical or similar structures.

When you draw a mapping connection between two structures on the mapping, MapForce creates a "Copy-All" connection automatically if it detects that the source and target structure are assignment compatible (that is, when both structures are either of the same type, or the target is a subtype of the source type). At mapping runtime, all instance data will be copied from the source to the target recursively, including children.

To create a "Copy-All" connection manually, right-click an existing connection between two similar nodes with child items, and select Copy-All (Copy Child Items) from the context menu.

Note the following:

- In contexts where a "Copy-All" connection is not meaningful or not supported, it is not possible to create this kind of connection manually.
- A "Copy-All" connection cannot be created to the root element of an XML/Schema component.
- When creating "Copy-All" connections between a schema and a parameter of a user-defined function, the two components must be based on the same schema. It is not necessary that they both have the same root elements, however.

For an example of a "Copy-All" connection created manually, take the following steps:

1. Create a new mapping.
2. On the Insert menu, click XML Schema/File and browse for the books.xml file located in the folder `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\`.
3. On the Insert menu, click XML Schema/File and browse for the library.xsd file located in the folder `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\`.
4. Draw a mapping connection between the book node of the "books" component to the publication node of the "library" component.
5. Right-click the new connection, and select Copy-All (Copy Child Items) from the context menu.
If there are slight differences between the source and the target structures, the "Copy-All" connection will enumerate, at mapping runtime, the source items (such as elements and attributes) and will copy only those that exist in the target type. This is repeated recursively.

For example, in the mapping above, only two child items are identical between the two structures (author and title) and thus they are mapped to the target. The item id is not included automatically because it is an attribute in the source and an element in the target. If you need to map, for example, category to genre, the "Copy-All" connection is no longer possible, because these are different items.

When an input connector (the small triangle to the side of the component) receives a "Copy-All" connection, it cannot accept any other connections. In the example above, if you attempt to create a connection between category and genre, MapForce prompts you to either replace it, or duplicate the input.

Duplicating input is meaningful only if you want the target to accept data from more than one input, which is not required here (see also Duplicating Input). If you choose to replace the "Copy-All" connection, a message box prompts you again to either resolve or delete the "Copy-All" connection.

Click Resolve copy-all connection if you want to replace the "Copy-All" connection by standard
individual target-driven connections to corresponding child items. If you prefer to remove the "Copy-All" connection completely, click **Delete child connections**.
5.3 Chained Mappings

MapForce supports mappings that consist of multiple components in a mapping chain. Chained mappings are mappings where at least one component acts both as a source and a target. Such a component creates output which is later used as input for a following mapping step in the chain. Such a component is called an "intermediate" component.

For example, the mapping illustrated below shows an expense report (in XML format) that is being processed in two stages. The part of the mapping from A to B filters out only those expenses that are marked as "Travel". The mapping from B to C filters out only those "Travel" expenses that have a travel cost less than 1500. Component B is the "intermediate" component, as it has both input and output connections. This mapping is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ChainedReports.mfd`.

![ChainedReports.mfd](image)

Chained mappings introduce a feature called "pass-through". "Pass-through" is a preview capability allowing you to view the output produced at each stage of a chained mapping in the Output window. For example, in the mapping above, you can preview and save the XML output resulting from A to B, as well as the XML output resulting from B to C.

**Note:** The "pass-through" feature is available only for file-based components (for example, XML, CSV, and text). Database components can be intermediate, but the pass-through button is not shown. The intermediate component is always regenerated from scratch when previewing or generating code. This would not be feasible with a database as it would have to be deleted prior to each regeneration.

If the mapping is executed by MapForce Server, or by generated code, then the full mapping chain is executed. The mapping generates the necessary output files at each step in the chain, and the output of a step of a mapping chain is forwarded as input to the following mapping step.

It is also possible for intermediate components to generate dynamic file names. That is, they can accept connections to the "File:" item from the mapping, provided that the component is
configured correspondingly. For more information, see Processing Multiple Input or Output Files Dynamically.

Preview button
Both the component B and the component C have preview buttons. This allows you to preview in MapForce the intermediate mapping result of B, as well as the final result of the chained mapping. Click the preview button of the respective component, then click Output to see the mapping result.

"Intermediate" components with the pass-through button active cannot be previewed. Their preview button is automatically disabled, because it is not meaningful to preview and let data pass through at the same time. To see the output of such a component, first click the "pass-through" button to deactivate it, and then click the preview button.

Pass-through button
The intermediate component B has an extra button in the component title bar called "pass-through".

If the pass-through button is active, MapForce maps all data into the preview window in one go; from component A to component B, then on to component C. Two result files will be created:

- the result of mapping component A to intermediate component B
- the result of the mapping from the intermediate component B, to target component C.

If the pass-through button is inactive, MapForce will execute only parts of the full mapping chain. Data is generated depending on which preview buttons are active:

- If the preview button of component B is active, then the result of mapping component A to component B is generated. The mapping chain actually stops at component B. Component C is not involved in the preview at all.
- If the preview button of component C is active, then the result of mapping intermediate component B to the component C is generated. Because pass-through is inactive, automatic chaining has been interrupted for component B. Only the right part of the mapping chain is executed. Component A is not used.

As previously mentioned, if the mapping is executed by MapForce Server, or by generated code, then the output of all components is generated. In this case, the settings of the pass-through button of component B, as well as the currently selected preview component, are disregarded. Taking the mapping above as example, two result files will be generated, as follows:

1. The output file resulting from mapping component A to B
2. The output file resulting from mapping component B to C.
The following sections, Example: Pass-Through Active and Example: Pass-Through Inactive, illustrate in more detail how the source data is transferred differently when the pass-through button is active or inactive.

### 5.3.1 Example: Pass-Through Active

The mapping used in this example (ChainedReports.mfd) is available in the `<Documents>` \Altova\MapForce2019\MapForceExamples\Tutorial\ folder. This mapping processes an XML file called ReportA.xml that contains travel expenses and looks as shown below. For simplicity, the namespace declaration and some expense-item elements have been omitted:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<expense-report currency="USD" detailed="true">
  <Person>
    <First>Fred</First>
    <Last>Landis</Last>
    <Title>Project Manager</Title>
    <Phone>123-456-78</Phone>
    <Email>f.landis@nanonull.com</Email>
  </Person>
  <expense-item type="Travel" expto="Development">
    <Date>2003-01-02</Date>
    <Travel Trav-cost="337.88"></Travel>
    <description>Biz jet</description>
  </expense-item>
  <expense-item type="Lodging" expto="Sales">
    <Date>2003-01-01</Date>
    <Lodging Lodge-cost="121.2"></Lodging>
    <description>Motel mania</description>
  </expense-item>
  <expense-item type="Travel" expto="Marketing">
    <Date>2003-02-02</Date>
    <Travel Trav-cost="2000"></Travel>
    <description>Hong Kong</description>
  </expense-item>
</expense-report>
```

ReportA.xml

The goal of the mapping it to produce, based on the file above, two further reports:

- **ReportB.xml** - this report should contain only those travel expenses that are of type "Travel".
- **ReportC.xml** - this report should contain only those travel expenses that are of type "Travel" and do not exceed 1500.
To achieve this goal, the intermediate component of the mapping (component B) has the pass-through button active, as shown below. This causes the mapping to be executed in stages: from A to B, and then from B to C. The output created by the intermediate component will be used as input for the mapping between B and C.

The names of generated output files at each stage in the mapping chain is determined by the settings of each component. (To open the component settings, right-click it, and then select Properties from the context menu). Namely, the first component is configured to read data from an XML file called ReportA.xml. Because this is a source component, the Output XML File field is irrelevant and it was left empty.

As shown below, the second component (ReportB) is configured to create an output file called ReportB.xml. Notice that the Input XML File field is grayed out. When pass-through is active (as in this example), the Input XML File field of the intermediate component is automatically deactivated. An input file name need not exist for the mapping to execute, because the output created at this stage in the mapping is stored in a temporary file and reused further in the mapping. Also, if an Output XML File is defined (as illustrated below), then it is used for the file name of the intermediate output file. If no Output XML File is defined, a default file name will be automatically used.
Finally, the third component is configured to produce an output file called `ReportC.xml`. The **Input XML File** field is irrelevant here, because this is a target component.

To select which of the two generated output files should be displayed in the Output window, either click the arrow buttons, or select the desired entry from the dropdown list.
When the mapping is executed by MapForce, the setting "Write directly to final output file" (configured from Tools | Options | General) determines whether the intermediate files are saved as temporary files or as physical files. Note that this is only valid when the mapping is previewed directly in MapForce. Had this mapping been executed by MapForce Server or by generated code, actual files would be produced at each stage in the mapping chain.

If StyleVision is installed, and if a StyleVision Power Stylesheet (SPS) file has been assigned to the target component (as in this example), then the final mapping output can be viewed (and saved as) HTML, RTF, PDF, or Word file. To generate and view this output in MapForce, click the tab with the corresponding name.
Note that only the output of the final target component in the mapping chain is displayed. To display StyleVision output of intermediary components, you would need to deactivate the pass-through button, and preview the intermediate component (as shown in Example: Pass-Through Inactive).

### 5.3.2 Example: Pass-Through Inactive

The mapping used in this example (*ChainedReports.mfd*) is available in the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\` folder. This example illustrates how output is generated differently when the pass-through button is deactivated on the intermediate component.
As explained in Example: Pass-Through Active, the goal of the mapping is to produce two separate reports. In the previous example, the pass-through button was active, and both reports were generated as expected and could be viewed in the Output tab. However, if you want to preview only one of the reports (either ReportB.xml or ReportC.xml), then the pass-through button must be deactivated. More precisely, deactivating the pass-through button may be useful if you want to achieve the following:

- Preview only output generated from A to B, and disregard the portion of the mapping from B to C
- Preview only output generated from B to C, and disregard the portion of the mapping from A to B.

When you deactivate the pass-through button as shown above, you can choose whether to preview either ReportB or ReportC (notice that both have preview buttons).

Deactivating the pass-through button also lets you to choose what input file should be read by the intermediate component. In most cases, this should be the same file as defined in Output XML File field (as in this example).

Settings of the intermediate component

Having the same input and output file on the intermediate component is particularly important if you intend to generate code from the mapping, or run the mapping with MapForce Server. As previously mentioned, in these environments, all outputs created by each component in the mapping chain are generated. So, it usually makes sense for the intermediate component to receive one file for processing (in this case ReportB.xml) and forward the same file to the subsequent mapping, rather than look for a different file name. Be aware that, not having the same input and output file names on the intermediate component (when the pass-through button is inactive) might cause errors such as “The system cannot find the file specified” in generated code or in MapForce Server execution.

If you click the preview button on the third component (ReportC), and attempt to preview the mapping in MapForce, you will notice that an execution error occurs. This is expected, since,
according to the settings above, a file called `ReportB.xml` is expected as input. However, the mapping did not produce yet such a file (because the pass-through button is not active, and only the portion of the mapping from B to C is executed). You can easily fix this problem as follows:

1. Click the preview button on the intermediate component.
2. Click the **Output** tab to preview the mapping.
3. Save the resulting output file as `ReportB.xml`, in the same folder as the mapping (`<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial`).

Now, if you click again the preview button on the third component (ReportC), the error is no longer shown.

When the pass-through button is inactive, you can also preview the StyleVision-generated output for each component that has an associated StyleVision Power StyleSheet (SPS) file. In particular, you can view the HTML version of the intermediate report as well (in addition to that of the final report):
Personal Expense Report

Employee Information

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>Landis</td>
<td>Project Manager</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E-Mail</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:f.landis@nanonull.com">f.landis@nanonull.com</a></td>
<td>123-456-78</td>
</tr>
</tbody>
</table>

Expense List

<table>
<thead>
<tr>
<th>Type</th>
<th>Expense To</th>
<th>Date (yyyy-mm-dd)</th>
<th>Expenses $</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Travel</td>
<td>Development</td>
<td>2003-01-02</td>
<td>Travel 337.88</td>
<td>Biz jet</td>
</tr>
<tr>
<td>Travel</td>
<td>Accounting</td>
<td>2003-07-07</td>
<td>Travel 1014.22</td>
<td>Ambassador class</td>
</tr>
<tr>
<td>Travel</td>
<td>Marketing</td>
<td>2003-02-02</td>
<td>Travel 2000</td>
<td>Hong Kong</td>
</tr>
</tbody>
</table>

*HTML output of the intermediate component*
5.4 Processing Multiple Input or Output Files Dynamically

You can configure MapForce to process multiple files (for example, all files in a directory) when the mapping runs. Using this feature, you can solve tasks such as:

- Supply to the mapping a list of input files to be processed
- Generate as mapping output a list of files instead of a single output file
- Generate a mapping application where both the input and output file names are defined at runtime
- Convert a set of files to another format
- Split a large file (or database) into smaller parts
- Merge multiple files into one large file (or load them into a database)

You can configure a MapForce component to process multiple files in one of the following ways:

- Supply the path to the required input or output file(s) using wildcard characters instead of a fixed file name, in the component settings (see Changing the Component Settings). Namely, you can enter the wildcards * and ? in the Component Settings dialog box, so that MapForce resolves the corresponding path when the mapping runs.
- Connect to the root node of a component a sequence which supplies the path dynamically (for example, the result of the `replace-fileext` function). When the mapping runs, MapForce will read dynamically all the input files or generate dynamically all the output files.

Depending on what you want to achieve, you can use either one or both of these approaches on the same mapping. However, it is not meaningful to use both approaches at the same time on the same component. To instruct MapForce which approach you want to use for a particular component, click the **File** or **File/String** button available next to the root node of a component. This button enables you to specify the following behavior:

<table>
<thead>
<tr>
<th>Use File Names from Component Settings</th>
<th>If the component should process one or several instance files, this option instructs MapForce to process the file name(s) defined in the Component Settings dialog box.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If you select this option, the root node does not have an input connector, as it is not meaningful.</td>
</tr>
<tr>
<td><strong>File</strong></td>
<td>Articles</td>
</tr>
<tr>
<td><strong>File/String</strong></td>
<td>Articles</td>
</tr>
<tr>
<td><strong>File</strong></td>
<td>Article</td>
</tr>
<tr>
<td><strong>File/String</strong></td>
<td>Article</td>
</tr>
</tbody>
</table>

If you did not specify yet any input or output files in the Component Settings dialog box, the name of the root node is **File: (default)**. Otherwise, the root node displays the name of the input file, followed by
a semi-colon ( ;), followed by the name of the output file.

If the name of the input is the same with that of the output file, it is displayed as name of the root node.

Note that you can select either this option or the *Use Dynamic File Names Supplied by Mapping* option.

<table>
<thead>
<tr>
<th>Use Dynamic File Names Supplied by Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>This option instructs MapForce to process the file name(s) that you define on the mapping area, by connecting values to the root node of the component.</td>
</tr>
<tr>
<td>If you select this option, the root node gets an input connector to which you can connect values that supply dynamically the file names to be processed during mapping execution. If you have defined file names in the Component Settings dialog box as well, those values are ignored.</td>
</tr>
<tr>
<td>When this option is selected, the name of the root node is displayed as <code>File: &lt;dynamic&gt;</code>.</td>
</tr>
<tr>
<td>This option is mutually exclusive with the <em>Use File Names from Component Settings</em> option.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parse Strings to XML, Parse Strings to JSON, Parse Strings to CSV, Parse Strings to FLF, Parse Strings to EDI</th>
</tr>
</thead>
<tbody>
<tr>
<td>When switched on, this option enables the component to accept a string value as input to the root node, and convert it to an XML, JSON, CSV, FLF, or EDI structure, respectively. For more information, see Parsing and Serializing Strings.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Serialize XML to Strings, Serialize JSON to Strings, Serialize CSV to Strings,</th>
</tr>
</thead>
<tbody>
<tr>
<td>When switched on, this option enables the component to accept a structure as input, and</td>
</tr>
</tbody>
</table>
Serialize FLF to Strings, Serialize EDI to Strings

Multiple input or output files can be defined for the following components:

- XML files
- Text files (CSV*, FLF* files and FlexText** files)
- EDI documents**
- Excel spreadsheets**
- XBRL documents**

* Requires MapForce Professional Edition
** Requires MapForce Enterprise Edition

The following table illustrates support for dynamic input and output file and wildcards in MapForce languages.

<table>
<thead>
<tr>
<th>Target language</th>
<th>Dynamic input file name</th>
<th>Wildcard support for input file name</th>
<th>Dynamic output file name</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLT 1.0</td>
<td>*</td>
<td>Not supported by XSLT 1.0</td>
<td>Not supported by XSLT 1.0</td>
</tr>
<tr>
<td>XSLT 2.0</td>
<td>*</td>
<td>*(1)</td>
<td>*</td>
</tr>
<tr>
<td>XQuery</td>
<td>*</td>
<td>*(1)</td>
<td>Not supported by XQuery</td>
</tr>
<tr>
<td>C++</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>C#</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Java</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>BUILT-IN</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Legend:

* Supported

(1) Uses the fn:collection function. The implementation in the Altova XSLT 2.0 and XQuery engines resolves wildcards. Other engines may behave differently. For details on how to transform XSLT 1.0/2.0 and XQuery code using the RaptorXML Server engine, see [Generating XSLT 1.0, or 2.0 code](#) and [Generating XQuery 1.0 code](#).

## 5.4.1 Mapping Multiple Input Files to a Single Output File

To process multiple input files, do one of the following:
- Enter a file path with wildcards (* or ?) as input file in the Component Settings dialog box. All matching files will be processed. The example below uses the * wildcard character in the Input XML file field to supply as mapping input all files whose name begins with "Nanonull"-. Multiple input files are being merged into a single output file because there is no dynamic connector to the target component, while the source component accesses multiple files using the wildcard *. Notice that the name of the root node in the target component is File: <default>, indicating that no output file path has been defined in the Component Settings dialog box. The multiple source files are thus appended in the target document.

![Diagram](image1.png)

MergeMultipleFiles.mfd (MapForce Basic Edition)

- Map a sequence of strings to the File node of the source component. Each string in the sequence represents one file name. The strings may also contain wildcards, which are automatically resolved. A sequence of file names can be supplied by components such as an XML file, database text fields, text files (CSV or fixed length), or an Excel sheet.
5.4.2 Mapping Multiple Input Files to Multiple Output Files

To map multiple files to multiple target files, you need to generate unique output file names. In some cases, the output file names can be derived from strings in the input data, and in other cases it is useful to derive the output file name from the input file name, e.g. by changing the file extension.

In the following mapping, the output file name is derived from the input file name, by adding the prefix “Persons-” with the help of the `concat` function.
Note: Avoid simply connecting the input and output root nodes directly, without using any processing functions. Doing this will overwrite your input files when you run the mapping. You can change the output file names using functions such as the `concat` function, as shown above.

The menu option File | Mapping Settings allows you to define globally the file path settings used by the mapping (see Changing the mapping settings).

### 5.4.3 Supplying File Names as Mapping Parameters

To supply custom file names as input parameters to the mapping, do the following:

1. Add an Input component to the mapping (On the Function menu, click Insert Input). For more information about such components, see Simple Input.
2. Click the File (File) or File/String (File/String) button of the source component and select Use Dynamic File Names Supplied by Mapping.
3. Connect the Input component to the root node of the component which acts as mapping source.

For a worked example, see Example: Using File Names as Mapping Parameters.

### 5.4.4 Previewing Multiple Output Files

Click the Output tab to display the mapping result in a preview window. If the mapping produces multiple output files, each file has its own numbered pane in the Output tab. Click the arrow.
buttons to see the individual output files.

To save the generated output files, do one of the following:

- On the Output menu, click Save All Output Files ( ).
- Click the Save all generated outputs ( ) toolbar button.

5.4.5 Example: Split One XML File into Many

This example shows you how to generate dynamically multiple XML files from a single source XML file. The accompanying mapping for this example is available at the following path:

<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\Tut-ExpReport-dyn.mfd.

The source XML file (available in the same folder as the mapping) consists of the expense report for a person called "Fred Landis" and contains five expense items of different types. The aim of the example is to generate a separate XML file for each of the expense items listed below.
As the `type` attribute defines the specific expense item type, this is the item we will use to split up the source file. To achieve the goal of this example, do the following:

1. Insert a `concat` function (you can drag it from the `core | string functions` library of the Libraries pane).
2. Insert a constant (on the Insert menu, click Constant) and enter ".xml" as its value.
3. Insert the `auto-number` function (you can drag it from the `core | generator functions` library of the Libraries pane).
4. Click the File (File) or File/String (File/String) button of the target component and select Use Dynamic File Names Supplied by Mapping.
5. Create the connections as shown below and then click the Output tab to see the result of the mapping.
Note that the resulting output files are named dynamically as follows:

- The `type` attribute supplies the first part of the file name (for example, "Travel").
- The `auto-number` function supplies the sequential number of the file (for example, "Travel1", "Travel2", and so on).
- The constant supplies the file extension, which is ".xml", thus "Travel1.xml" is the file name of the first file.

### 5.4.6 Example: Split Database Table into Many XML Files

This example shows you how to generate dynamically multiple XML files, one for each record of a database table. The accompanying mapping for this example is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\PersonDB-dyn.mfd`.

The source database file (available in the same folder as the mapping) includes a Person table
which contains 21 records. The aim of the example is to generate a separate XML file for each record in the Person table.

As the "PrimaryKey" field uniquely identifies each person in the table, this is the item we will use to split up the source database into separate files. To achieve the goal of this example, do the following:

1. Insert a `concat` function (you can drag it from the `core | string functions` library of the Libraries pane).
2. Insert a constant (on the `Insert` menu, click `Constant`) and enter ".xml" as its value.
3. Click the `File` (File) or `File/String` (File/String) button of the target component and select `Use Dynamic File Names Supplied by Mapping`.
4. Create the connections as shown below and then click the `Output` tab to see the result of the mapping.
Note that the resulting output files are named dynamically as follows:

- The **PrimaryKey** field supplies the first part of the file name (for example, "1").
- The constant supplies the file extension (".xml"), thus "1.xml" is the file name of the first file.

### 5.4.7 Multiple XML files from Excel rows

The content of the `altova.xlsx` spreadsheet file, available in the `...\MapForceExamples\Tutorial` folder, is shown below. It consists of two worksheets: Admin with 10, and Development with 11, rows of data. This example is available as `Excel-Mapping-dyn.mfd` in the `...\Tutorial` folder.

MapForce is able to display and map Excel components in two different ways, depending on the component options. The default settings are shown in the dialog box below. The "Show Worksheets by name" check box is selected when you first insert the Excel component.
To access a Workbook as if it were a single Worksheet:

1. Click the icon next to **Admin** in the Excel component.

2. Click the “Show Worksheets by name” check box to deselect it. The named Worksheets are not visible anymore, as shown in the screenshot below, but a Worksheet Name item is now available.
Aim 1: To generate separate files for each Worksheet containing the person records of each. The "Worksheet Name" item determines the specific worksheets in the workbook, so this is the item we will use to split up the source workbook into separate files.

1. Drag the `concat` function from the Libraries window into the mapping, and insert a constant.

2. Create the connections as shown above: Worksheet Name to value1 and the constant to value2.

3. Connect the `result` parameter of the `concat` function to the `File: <dynamic>` item of the target component. Note that `File: <dynamic>` is now displayed.

4. Define the remaining connections as needed.

5. Click the Output tab to see the result of the mapping.
Each record is now visible in its own Preview tab, the first one is shown above. Clicking the Next/Previous arrows allows you to see each of the files in the Output tab.

Notes:

- The **Worksheet** name field supplies the first part of the file name e.g. Admin.
- The **constant** component supplies the file extension i.e. `.xml`, thus Admin.xml is the file name of the first file. Admin.xml contains all the rows of that Excel tab. Development.xml contains the other rows.
- Clicking the Save All icon allows you to save the individual files directly from the Output tab, without having to generate code.

**Aim 2:** To generate separate files for each Person in each Worksheet

1. Click the icon next to Worksheets if you followed the section above, and activate the "Show Worksheets by name" check box.
   Each of the separate Worksheets are now visible in the component: Admin and Development.
Designing Mappings

Processing Multiple Input or Output Files Dynamically

1. Insert the concat and constant components as shown.
2. Insert a second XML Schema file of the same name and create the connections as shown.

As the "Row number" element determines the specific person rows in the worksheet, this is the item we will use to split up each worksheet into separate files.

For the **Admin** worksheet item in the Altova XLSX component:

1. Create the connections as shown above: Admin constant to value1, **Row Number** to value2, and .xml constant to value3.
2. Connect the **result** parameter of the concat function to the **File**: item of the target component. Note that File: <dynamic> is now displayed.
3. Define the remaining connections as needed.
4. Click the Output tab to see the result of the mapping.
Each row is now visible in its own Preview tab, the first one is shown above. All 10 records of the Admin worksheet have been split into separate files. Clicking the Next/Previous arrows allows you to see each of the files in the Output tab.

Notes:

- The Admin constant supplies the first part of the file name e.g. Admin.
- The Row Number item supplies the row number from the Excel worksheet e.g. 1.
- The .xml constant supplies the file name extension used when saving the file which is Admin1.xml, as shown above.
- Clicking the Save All icon allows you to save the individual files directly from the Output tab, without having to generate code.

To see the output of the Development worksheet, click the Preview button of that component, then click the Output tab to see the result of the mapping. 11 records have been split into separate files.
5.5 Supplying Parameters to the Mapping

You can pass simple values to a mapping by means of simple input components. On the mapping area, simple input components play the role of a source component which has a simple data type (for example, string, integer, and so on) instead of a structure of items and sequences. Consequently, you can create a simple input component instead of (or in addition to) a file-based source component.

You can use simple input components in any the following MapForce transformation languages:

- BUILT-IN (when you preview the mapping transformation directly in MapForce, from the Preview tab)
- BUILT-IN (when you run a compiled MapForce Server execution file)
- XSLT 1.0, XSLT 2.0
- XQuery
- C++
- C#
- Java

In case of mappings executed with MapForce Server or by means of generated code, simple input components become command line parameters. In case of mappings generated as XSLT transformations, simple input components correspond to stylesheet parameters in the generated XSLT file.

You can create each simple input component (or parameter) as optional or mandatory (see Input Component Settings). If necessary, you can also create default values for the mapping input parameters (see Creating a Default Input Value). This enables you to safely run the mapping even if you do not explicitly supply a parameter value at mapping execution time.

Input parameters added on the main mapping area should not be confused with input parameters in user-defined functions (see User-Defined Functions). There are some similarities and differences between the two, as follows.

<table>
<thead>
<tr>
<th>Input parameters on the mapping</th>
<th>Input parameters of user-defined functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Added from Function</td>
<td>Insert Input menu.</td>
</tr>
<tr>
<td>Can have simple data types (string, integer, and so on).</td>
<td>Can have simple as well as complex data types.</td>
</tr>
<tr>
<td>Applicable to the entire mapping.</td>
<td>Applicable only in the context of the function in which they were defined.</td>
</tr>
</tbody>
</table>

When you create a reversed mapping (using the menu command Tools | Create Reversed Mapping), a simple input component becomes a simple output component.

For an example, see Example: Using File Names as Mapping Parameters.
5.5.1 Adding Simple Input Components

To add a simple input to the mapping:

1. Make sure that the mapping window displays the main mapping (not a user-defined function).
2. On the Function menu, click Input.
3. Enter a name and select the data type required for this input. If the input should be treated as a mandatory mapping parameter, select the Input is required check box. For a complete list of settings, see Simple Input Component Settings.

Note: The parameter name can contain only letters, digits, and underscores; no other characters are allowed. This makes it possible for a mapping to work across all code generation languages.

4. Click OK.

![Create Input dialog box](image)

You can change later any of the settings defined here (see Simple Input Component Settings).

5.5.2 Simple Input Component Settings

You can define the settings applicable to a simple input component when adding it to the mapping area. You can also change the settings at a later time, from the Edit Input dialog box.
Edit Input dialog box

To open the Edit Input dialog box, do one of the following:

- Select the component, and, on the Component menu, click Properties.
- Double-click the component.
- Right-click the component, and then click Properties.

The available settings are as follows.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter a descriptive name for the input parameter corresponding to this component. At mapping execution time, the value entered in this text box becomes the name of the parameter supplied to the mapping; therefore, no spaces or special characters are allowed.</td>
</tr>
<tr>
<td>Datatype</td>
<td>By default, any input parameter is treated as string data type. If the parameter should have a different data type, select the respective value from the list. When the mapping is executed, MapForce casts the input parameter to the data type selected here.</td>
</tr>
<tr>
<td>Input is required</td>
<td>When enabled, this setting makes the input parameter mandatory (that is, the mapping cannot be executed unless you supply a parameter value). Disable this check box if you want to specify a default value for the input parameter (see Creating a Default Input Value).</td>
</tr>
<tr>
<td>Specify value</td>
<td>This setting is applicable only if you execute the mapping during design time, by clicking the Preview tab. It allows you to enter directly in the component the value to use as mapping input.</td>
</tr>
<tr>
<td>Value</td>
<td>This setting is applicable only if you execute the mapping during design time, by clicking the Preview tab. To enter a value to be used by MapForce as mapping input, select the Specify Value check box, and then...</td>
</tr>
</tbody>
</table>
5.5.3 Creating a Default Input Value

After you add an Input component to the mapping area, notice the default item to the left of the component.

The default item enables you to connect an optional default value to this input component, as follows:

1. Add a constant component (on the Insert menu, click Constant), and then connect it to the default item of the input component.

2. Double click the input component and make sure that the Input is required check box is disabled. When you create a default input value, this setting is not meaningful and causes mapping validation warnings.
3. Click OK.

Note: If you click the Specify value check box and enter a value in the adjacent box, the entered value takes precedence over the default value when you preview the mapping (that is, at design-time execution). However, the same value has no effect in the generated code.

5.5.4 Example: Using File Names as Mapping Parameters

This example walks you through the steps required to execute a mapping that takes input parameters at runtime. The mapping design file used in this example is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\FileNamesAsParameters.mfd`.

The mapping uses two input components: InputFileName and OutputFileName. These supply the input file name (and the output file name, respectively) of the source and target XML file. For this reason, they are connected to the File: `<dynamic>` item.

Both the InputFileName and OutputFileName components are simple input components in the mapping, so you can supply them as input parameters when executing the mapping. The following sections illustrate how to do this in the following transformation languages:

- **XSLT 2.0**, using RaptorXML Server
- Built-in (MapForce Server Execution File), using MapForce Server
- **Java**

XSLT 2.0

If you generate code in XSLT 1.0 or XSLT 2.0, the input parameters are written to the
DoTransform.bat batch file, for execution by RaptorXML Server (see Automation with RaptorXML Server). To use a different input (or output) file, you can either pass the required parameters at command line, when calling the DoTransform.bat file, or edit the latter to include the required parameters.

To supply a custom input parameter in the DoTransform.bat file:

1. Generate the XSLT 2.0 code (File | Generate Code In | XSLT 2.0) from the FileNamesAsParameters.mfd sample.
2. Copy the Altova_Hierarchical.xml file from the <Documents>\Altova\MapForce2019 \MapForceExamples directory to the directory where you generated the XSLT 2.0 code (in this example, c:\codegen\examples\xslt2). This file will act as custom parameter.
3. Edit DoTransform.bat to include the custom input parameter either before or after %* (as highlighted below). Note that the parameter value is enclosed with single quotes. The available input parameters are listed in the rem (Remark) section.

```batch
@echo off
RaptorXML xslt --xslt-version=2 --
input="MappingMapToAltova_Hierarchical.xslt" --
param=InputFileName:'Altova_Hierarchical.xml' %*
"MappingMapToAltova_Hierarchical.xslt"
rem --param=InputFileName:
rem --param=OutputFileName:
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```

When you run the DoTransform.bat file, RaptorXML Server completes the transformation using Altova_Hierarchical.xml as input parameter.

MapForce Server Execution File

To supply custom input parameters to a MapForce Server execution file:

1. Compile the FileNamesAsParameters.mfd to a MapForce Server execution file (see Compiling Mappings to MapForce Server Execution Files). When prompted, save the .mfx execution file to a directory on your computer (in this example, c:\codegen\examples \mfx).
2. Copy the Altova_Hierarchical.xml file from the <Documents>\Altova\MapForce2019 \MapForceExamples directory to the directory where you saved the .mfx file. This file will act as the custom parameter supplied to the mapping execution file.
3. Run MapForce Server with the following command:

```
MapForceServer.exe run "C:\codegen\examples\mfx
```
In the MapForce Server command above, `-p=InputFileName` and `-p=OutputFileName` are the input parameters to the mapping. You can use any file name as the value of `-OutputFileName`. However, the file name supplied in `-InputFileName` parameter must exist as a physical file; otherwise, the mapping will fail.

**Note:** If you see the message "MapForceServer.exe is not recognized as an internal or external command, operable program, or batch file", change the current directory to the one where the MapForce Server executable is installed. To avoid changing path every time when you run a mapping, add to your operating system's PATH environment variable the path of the directory where MapForce Server executable is installed (for example, `C:\Program Files (x86)\Altova\MapForceServer2019\bin`).

### Java

To supply a custom input parameter to a Java .jar application:

1. Generate the Java code ([File | Generate Code In | Java](#)) from the `FileNamesAsParameters.mfd` sample.
2. Compile the Java code into an executable JAR file (for instructions on how to do this in Eclipse, see [Example: Build a Java application with Eclipse and Ant](#)).
3. Copy the `Altova_Hierarchical.xml` file from the `<Documents>\Altova\MapForce2019\MapForceExamples` directory to the directory where the .jar file is. This file will act as the custom parameter supplied to the Java mapping application.
4. At the command line, enter: `java -jar Mapping.jar /InputFileName "InputFile.xml"`

If you use wildcards when passing parameters to .jar files, place the wildcard parameters in quotes, for example:

```
java -jar Mapping.jar /InputFileName "altova-*.xml"
```
5.6 Returning String Values from a Mapping

Use a simple output component when you need to return a string value from the mapping. On the mapping area, simple output components play the role of a target component which has a string data type instead of a structure of items and sequences. Consequently, you can create a simple output component instead of (or in addition to) a file-based target component. For example, you can use a simple output component to quickly test and preview the output of a function (see Example: Testing Function Output). This technique is also useful for mappings which use string serialization (see String Parsing and Serialization). The main purpose of a simple output component is, however, to get back a string when calling the MapForce Server API, without writing any files.

Simple output components should not be confused with output parameters of user-defined functions (see User-Defined Functions). There are some similarities and differences between the two, as follows.

<table>
<thead>
<tr>
<th>Output components</th>
<th>Output parameters of user-defined functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Added from Function</td>
<td>Insert Output menu.</td>
</tr>
<tr>
<td>Have &quot;string&quot; as data type.</td>
<td>Can have simple as well as complex data types.</td>
</tr>
<tr>
<td>Applicable to the entire mapping.</td>
<td>Applicable only in the context of the function in which they were defined.</td>
</tr>
</tbody>
</table>

If necessary, you can add multiple simple output components to a mapping. You can also use simple output components in combination with file-based and database target components. When your mapping contains multiple target components, you can preview the data returned by a particular component by clicking the Preview (ckeck) button in the component title bar, and then clicking the Output tab on the Mapping window.

You can use simple output components as follows in MapForce transformation languages:

<table>
<thead>
<tr>
<th>Language</th>
<th>How it works</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUILT-IN (when previewing the mapping transformation)</td>
<td>You can preview Output components in the same way as you would preview a file-based mapping output—by clicking the Output tab on the Mapping window.</td>
</tr>
</tbody>
</table>
| BUILT-IN (when running the MapForce Server execution file) | When you run a compiled MapForce Server execution file (see Compiling a MapForce mapping), the mapping output is returned in the standard output stream (stdout), so you can view it or redirect to a file. For example, assuming that the name of the MapForce server execution file is MyMapping.mfx, use the following syntax to redirect the mapping output to output.txt file and any errors to the log.txt file:  

```
MapForceServer.exe run MyMapping.mfx >output.txt
```
### XSLT 1.0, XSLT 2.0

If the generated XSLT files, a simple output component is defined in the mapping becomes the output of the XSLT transformation.

If you are using RaptorXML Server, you can instruct RaptorXML Server to write the mapping output to the file passed as value to the `--output` parameter.

To write the output to a file, add or edit the `--output` parameter in the `DoTransform.bat` file. For example, the following `DoTransform.bat` file has been edited to write the mapping output to the `Output.txt` file (see highlighted text).

```
RaptorXML xslt --xslt-version=2 --
input="MappingMapToResult1.xslt" --
output="Output.txt" %* "MappingMapToResult1.xslt"
```

If an `--output` parameter is not defined, the mapping output will be written to the standard output stream (stdout) when the mapping is executed.

### C++, C#, Java

In the generated C++, C#, and Java code, the mapping output is written to the standard output of the generated application.

If the mapping contains multiple target components, the generated application concatenates the standard output of each target component and returns it as one unified standard output.

---

When you create a reversed mapping (using the menu command **Tools** | **Create Reversed Mapping**), the simple output component becomes a simple input component.

### 5.6.1 Adding Simple Output Components

To add an Output component to the mapping area:

1. Make sure that the mapping window displays the main mapping (not a user-defined function).
2. On the **Function** menu, click **Output**.
3. Enter a name for the component.
4. Click **OK**.
You can change the component name at any time later, in one of the following ways:

- Select the component, and, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component header, and then click **Properties**.

### 5.6.2 Example: Previewing Function Output

This example illustrates how to preview the output returned by MapForce functions with the help of simple output components. You will make the most of this example if you already have a basic understanding of functions in general, and of MapForce functions in particular. If you are new to MapForce functions, you may want to refer to [Using Functions](#) before continuing.

Our aim is to add a number of functions to the mapping area, and learn how to preview their output with the help of simple output components. In particular, the example uses a few simple functions available in the core library. Here is a summary of their usage:

- **string-length** Returns the number of characters in the string provided as argument. For example, if you pass to this function the value "Lorem ipsum", the result is "11", since this is the number of characters that the text "Lorem ipsum" takes.

- **substring-after** Returns the part of the string that occurs after the separator provided as argument. For example, if you pass to this function the value "Lorem ipsum" and the space character (" "), the result is "ipsum".

- **substring-before** Returns the part of the string that occurs before the separator provided as argument. For example, if you pass to this function the value "Lorem ipsum" and the space character (" "), the result is "Lorem".

To test each of these functions against a custom text value ("Lorem ipsum", in this example), follow the steps below:

1. Add a constant with the value "Lorem ipsum" to the mapping area (use the menu command **Insert | Constant**). The constant will be the input parameter for each of the
functions to be tested.

2. Add the string-length, substring-after, and substring-before functions to the mapping area, by dragging them to the mapping area from the core library, string functions section.

3. Add a constant with an empty space (" ") as value. This will be the separator parameter required by the substring-after and substring-before functions.

4. Add three simple output components (use the menu command Function | Insert Output). In this example, they have been named Result1, Result2, and Result3, although you can give them another title.

5. Connect the components as illustrated below.

![Diagram of function output with simple output components]

As shown in the sample above, the "Lorem ipsum" string acts as input parameter to each of the string-length, substring-after, and substring-before functions. In addition to this, the substring-after and substring-before functions take a space value as second input parameter. The Result1, Result2, and Result3 components can be used to preview the result of each function.

To preview the output of any function:

- Click the Preview ( ) button in the component title bar, and then click the Output tab on the Mapping window.
5.7 Using Variables

Variables are a special type of component used to store an intermediate mapping result for further processing. They might be necessary in situations where you want to temporarily “remember” some data on the mapping and process it in some way (for example, filter it, or apply some functions) before it is copied to the target component.

Variables can be of simple type (for example, string, integer, boolean, etc) or complex type (a tree structure).

Simple variable

You can create a variable of complex type by supplying an XML schema which expresses the structure of the variable. If the schema defines any elements globally, you can choose which one should become the root node of the variable structure. Note that a variable does not have any associated instance XML file; the data of the variable is computed at mapping runtime.

Complex variable created from an XML schema

It is also possible to create variables of complex type from databases, EDI files, or FlexText templates (for more information about the latter, see MapForce FlexText). In case of databases, you can choose a specific database table as root item for the variable structure.
Using Variables

Complex variable created from a database table

In the images above, you may notice that each variable has an item called `compute-when`. Connecting this item is optional; this enables you to control how the variable value should be computed on the mapping (see Changing the Context and Scope of Variables).

When necessary, items of a variable structure can be duplicated to accept data from more than one source connection, similar to how this is done for standard components (see Duplicating Input). This does not apply, however, to variables created from database tables.

Simple variable with duplicated inputs

One of the most important things about variables is that they are sequences, and can be used to create sequences. The term "sequence" here means a list of zero or more items (see also Mapping Rules and Strategies). This makes it possible for a variable to process multiple items for the duration of the mapping lifetime. If, however, you want to assign a value once to a variable and keep it the same for the rest of the mapping, it is also possible (see Changing the Context and Scope of Variables).

To some extent, variables can be compared to intermediate components of a chained mapping (see Chained Mappings). However, they are more flexible and convenient if you don’t need to produce intermediary files at each stage in the mapping. The following table outlines differences between variables and chained mappings.

<table>
<thead>
<tr>
<th>Chained mappings</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chained mappings involve two totally independent steps. For example, let’s assume a mapping that has three components A, B, and C. Running the mapping involves two stages: executing the mapping from A to B, and then executing the mapping from B to C.</td>
<td>While the mapping is executed, variables are evaluated according to their context and scope. Their context and scope can be influenced (see Changing the Context and Scope of Variables).</td>
</tr>
</tbody>
</table>
Designing Mappings Using Variables

<table>
<thead>
<tr>
<th>Chained mappings</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>When the mapping is executed, intermediate results are stored externally in files.</td>
<td>When the mapping is executed, intermediate results are stored internally. No external files containing a variable's results are produced.</td>
</tr>
<tr>
<td>The intermediate result can be previewed using the preview button.</td>
<td>A variable's result cannot be previewed, since it is computed at mapping runtime. You can, however, use the debugger to see the data produced by the variable (see Debugging Mappings).</td>
</tr>
</tbody>
</table>

**Note:** Variables are not supported if the mapping transformation language is set to XSLT 1.0.

### 5.7.1 Adding Variables

There are several ways to add variables to a mapping, as shown below.

**Using a menu or toolbar command**

1. On the **Insert** menu, click **Variable**. (Alternatively, click the **Variable** toolbar button).

   ![Create Variable](image.png)

2. Select the type of variable you want to insert (simple or complex type).

   If you select "Complex type", there are a few additional steps:

   3. Click **Choose** to select the source which should provide the structure of the variable (for example, an XML Schema, database, EDI file, or FlexText template).
4. When prompted, specify a root item of the structure. In case of XML Schemas, the root item can be any element defined globally. In case of databases, the root item can be any table.
Using a context menu

- Right-click the output connector of a component (in this example, "Article") and select **Create Variable from Source node**.

This creates a complex variable using the same source schema and automatically connects all items with a Copy-All connection.
Right-click the input connector of a target component and select **Create Variable for Target Node**. This creates a complex variable using the same schema as the target, and automatically connects all items with a Copy-All connection.

Right-click the output connector of a filter component (on-true/on-false) and select **Create Variable from Source Node**. This creates a complex component using the source schema, and automatically uses the item linked to the filter input as the root element of the intermediate component.

### 5.7.2 Changing the Context and Scope of Variables

Every variable has a **compute-when** input item which allows you to control the scope of the variable; in other words, when and how often the variable value is computed when the mapping is executed. You do not have to connect this input in many cases, but it can be essential to override the default context, or to optimize the mapping performance.

In the following text, a **subtree** means the set of an item/node in a target component and all of its descendants, for example, a `<Person>` element with its `<FirstName>` and `<LastName>` child elements.

A **variable value** means the data that is available at the output side of the variable component.

- For simple variables, it is a sequence of atomic values that have the datatype specified in the component properties.
- For complex variables, it is a sequence of root nodes (of the type specified in the component properties), each one including all its descendant nodes.

The sequence of atomic values (or nodes) may contain one or even zero elements. This depends on what is connected to the input side of the variable, and to any parent items in the source and target components.

**"Compute-when" is not connected (default)**

If the **compute-when** input item is not connected (to an output node of a source component), the variable value is computed **whenever it is first used in a target subtree** (either directly via a
connector from the variable component to a node in the target component, or indirectly via functions). The same variable value is also used for all target child nodes inside the subtree.

The actual variable value depends on any connections between parent items of the source and target components.

This default behavior is the same as that of complex outputs of regular user-defined functions and Web service function calls.

If the variable output is connected to multiple unrelated target nodes, the variable value is computed separately for each of them. This can produce different results in each case, because different parent connections influence the context in which the variable’s value is evaluated.

"Compute-when" is connected
By connecting an output connector of a source component to compute-when, the variable is computed whenever that source item is first used in a target subtree.

The variable actually acts as if it were a child item of the item connected to compute-when. This makes it possible to bind the variable to a specific source item. That is, at runtime the variable is re-evaluated whenever a new item is read from the sequence in the source component. This relates to the general rule governing connections in MapForce: “for each source item, create one target item”. With compute-when, it means "for each source item, compute the variable value" (see Mapping Rules and Strategies).

"Compute-once"
If necessary, you can choose to compute the variable value once before each of the target components, making the variable essentially a global constant for the rest of the mapping. To do this, right-click the compute-when item and select Compute Once from the context menu:

When you change the scope of a variable to compute-when=once, the input connector is removed from the compute-when item, since such a variable is only evaluated once.

In a user-defined function, the compute-when=once variable is evaluated each time the function is called, before the actual function result is evaluated.

Parent-context
Adding a parent-context may be necessary, for example, if your mapping uses multiple filters and you need an additional parent node to iterate over, see also Overriding the Mapping Context.
To add a parent-context to a variable, right-click the root node (in this example, "PersonList") and select Add Parent Context from the context menu. This adds a new node, parent-context, to the existing hierarchy.

![Diagram of a mapping with a parent-context node added to the hierarchy.]

The parent context adds a virtual "parent" node to the hierarchy within the component. This allows you to iterate over an additional node in the same, or in a different source component.

### 5.7.3 Example: Counting Database Table Rows

The mapping illustrated in this example is available as DB_UserList.mfd in the `<Documents>`\Altova\MapForce2019\MapForceExamples\ folder. This mapping extracts user records from a database table called "Users" and writes them to an XML file. The database column "Username" contains both the first name and the surname of a person (for example, "Vernon Callaby"). This mapping has the following goals:

1. For each record in the "Users" table, create a new Person element in the XML file.
2. Split the value extracted from the database field "Username" into two separate fields in the XML file ("First" and "Last").
3. For each record, find its sequential number compared to the number of total records present in the database (for example, "Record 1 of 4") and write this information to the Details element.

![Diagram of the DB_UserList.mfd mapping example.]

As illustrated above, in order to achieve the first goal, a connection is drawn between the source
"Users" table and the Person element of the target XML file. This ensures that, for each record in the source table, a new Person element will be created in the target.

The value of the field "Username" is supplied to the substring-before and substring-after functions. These two functions extract the text before and after the space character (" "), respectively, which takes care of the second mapping goal.

Finally, to achieve the third goal, the mapping uses the count function. The result of the count function is passed on to a variable. The variable ensures that this result is stored on the mapping and available when writing the "Details" element of each person to the target XML. Note that, for efficiency reasons, database records should be counted only once, so the variable scope is set to compute-when=once (see Changing the Context and Scope of Variables).

5.7.4 Example: Filtering and Numbering Nodes

The mapping illustrated in this example is available as PositionInFilteredSequence.mfd in the <Documents>\Altova\MapForce2019\MapForceExamples\ folder.

This mapping reads an XML file which contains contact data of several people, filters them, and writes them to a target XML file. The goal of the mapping is to filter from the source XML file only those people whose last name begins with letter "M" or a subsequent letter. Secondly, the extracted contacts must be numbered. The number is going to act as the unique identifier of each contact in the target XML file.

To achieve the goal above, the following component types were added to the mapping:

- A filter (see Filters and Conditions)
- A complex variable (see Adding Variables)
- The functions greater and position (see Working with Functions)
- A constant (To add a constant, select the menu command Insert | Constant).

The variable uses the same schema as the source component. If your right-click the variable and select Properties from the context menu, notice that the node BranchOffices/Office/Contact is selected as root node for this variable structure.
First, data of the source component is passed on to the filter. The filter passes onwards to the variable only those records that meet the filter condition. Namely, the filter is configured to get only those Contact nodes where the first name is equal or greater than M". To achieve this, the function greater compares each last item with the constant value "M".

The variable has the compute-when input connected to the root item of the source component (BranchOffices). At runtime, this causes the variable to be re-evaluated whenever a new item is read from the sequence in the source component. In this mapping, however, connecting or not connecting the compute-when item does not make a difference. The reason is that the variable is connected to the Contact source item (indirectly through the filter), and it would compute as many times as there are instances of Contact which meet the filter condition.

The position functions returns, for each iteration of the variable, the number of the current sequence. Only eight contacts meet the filter condition; therefore, if you preview the mapping and look at the output, notice how IDs 1 through 8 were written to the ID element of the target component.

In case you were wondering why the variable was necessary at all, it is because of the requirement to number all records. Had we connected the filter result directly to the target component, there would have been no way to number each occurrence of Contact. The purpose of the variable in this mapping is, therefore, to store each instance of Contact temporarily on the mapping, so that it can be numbered before it is written to the target.

### 5.7.5 Example: Grouping and Subgrouping Records

The mapping illustrated in this example is available as DividePersonsByDepartmentIntoGroups.mfd in the <Documents>\Altova\MapForce2019\MapForceExamples\ folder.

This mapping processes an XML file that contains employee records of a fictitious company. The company has two offices: "Nanonull, Inc." and "Nanonull Partners, Inc". Each office has several departments (for example, “IT”, “Marketing”, and so on), and each department has one or more employees. The goal of the mapping is to create groups of maximum three people from each department, regardless of the office. The size of each group is three by default; however, it should be easy to change if necessary. Each group must be saved as a separate XML file, with the name having the format "<Department Name>_GroupN" (for example, Marketing_Group1.xml, Marketing_Group2.xml, and so on).
As illustrated above, in order to achieve the mapping goal, a complex variable was added to the mapping, and a few other component types (primarily functions). The variable has the same structure as a Department item in the source XML. If you right-click the variable in order to view its properties, you will notice that it uses the same XML schema as the source component, and has Department as root element. Importantly, the variable has two nested parent-context items, which ensure that the variable is computed first in the context of each department, and then in the context of each group within each department (see also Changing the Context and Scope of Variables).

Initially, the mapping iterates through all departments in order to obtain the name of each department (this will be subsequently required to create the file name corresponding to each group). This is achieved by connecting the group-by function to the Department source item, and by supplying the department name as grouping key.

Next, within the context of each department, a second grouping takes place. Namely, the mapping calls the group-into-blocks function in order to create the required groups of people. The size of each group is supplied by a simple input component which has a default value of "3". The default value is supplied by a constant. In this example, in order to change the size of each group, one can easily modify the constant value as required. However, the "size" input component can also be modified so that, if the mapping is run by generated code or with MapForce Server, the size of each group could be conveniently supplied as a parameter to the mapping. For more information, see Supplying Parameters to the Mapping.

Next, the value of the variable is supplied to the target PersonList XML component. The file name for each created group was computed by concatenating the following parts, with the help of the concat function:

1. The name of each department
2. The string "_Group"
3. The number of the group in the current sequence (for example, "1" if this is the first group
for this department)

4. The string ".xml"

The result of this concatenation is stored in the Name item of the variable, and then supplied as a dynamic file name to the target component. This causes a new file name to be created for each received value. In this example, the variable computes eight groups in total, so eight output files are created when the mapping runs, as required. For more information about this technique, see Processing Multiple Input or Output Files Dynamically.
5.8 Sorting Data

To sort input data based on a specific sort key, use a Sort component. The Sort component supports the following target languages: XSLT2, XQuery, and Built-in. When the transformation language is "Built-in", the Sort component can be used to sort database table data. Better performance is, however, achieved using an SQL-WHERE/ORDER component. For more details, see Filtering and Sorting Database Data (SQL WHERE/ORDER).

To add a sort component to the mapping, do one of the following:

- Right-click an existing connection, and select Insert Sort: Nodes/Rows from the context menu. This inserts the Sort component and automatically connects it to the source and target components. For example, in the mapping below, the Sort component was inserted between a variable and an XML component. The only thing that remains to be connected manually is the sorting key (the field by which you want to sort).

![Mapping diagram showing Sort component](image1)

- On the Insert menu, click Sort (alternatively, click the Sort toolbar button). This inserts the Sort component in its "unconnected" form.

![Sort component in unconnected form](image2)

As soon as a connection is made to the source component, the title bar name changes to that of the item connected to the nodes/rows item.

To define the item by which you want to sort:

- Connect the item by which you want to sort to the key parameter of the Sort component. For example, in the mapping below, the Person nodes/rows are sorted by the field Last.

![Mapping diagram showing Sort component connected to Person nodes/rows](image3)
To change the sort order:

- Click the A→Z icon in the Sort component. It changes to Z→A to show that the sort order has been reversed.

To sort input data consisting of simple type items:

- Connect the item to both the nodes/rows and key parameters of the sort component. In the mapping below, the element of simple type first is being sorted.

To sort strings using language-specific rules:

- Double-click the header of the Sort component to open the Sort Properties dialog box.
Unicode codepoint collation: This (default) option compares/orders strings based on code point values. Code point values are integers that have been assigned to abstract characters in the Universal Character Set adopted by the Unicode Consortium. This option allows sorting across many languages and scripts.

Language-specific collation: This option allows you to define the specific language and country variant you want to sort by. This option is supported when using the BUILT-IN execution engine. For XSLT, support depends on the specific engine used to execute the code.

5.8.1 Sorting by Multiple Keys

After you add a Sort component to the mapping, one sorting key called key is created by default.

If you want to sort by multiple keys, adjust the Sort component as follows:

- Click the Add Key ( ) icon to add a new key (for example, key2 in the mapping below).
- Click the Delete Key ( ) icon to delete a key.
- Drop a connection onto the icon to add a key and also connect to it.

A mapping which illustrates sorting by multiple key is available at the following path: <Documents>\Altova\MapForce2019\MapForceExamples\SortByMultipleKeys.mfd.
In the mapping above, Person records are sorted by three sorting keys:

1. Shares (number of shares a person holds)
2. Last (last name)
3. First (first name)

Note that the position of the sorting key in the Sort component determines its sort priority. For example, in the mapping above, records are initially sorted by the number of shares. This is the sorting key with the highest priority. If the number of shares is the same, people are then sorted by their last name. Finally, when multiple people have the same number of shares and the same last name, the person's first name is taken into account.

The sort order of each key can be different. In the mapping above, the key Shares has a descending sort order (Z-A), while the other two keys have ascending sort order (A-Z).

5.8.2 Sorting with Variables

In some cases, it may be necessary to add intermediate variables to the mapping in order to achieve the desired result. This example illustrates how to extract records from an XML file, and sort them, with the help of intermediate variables. The example is accompanied by a mapping sample located at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Altova_Hierarchical_Sort.mfd`. 
This mapping reads data from a source XML file called **Altova_Hierarchical.xml** and writes it to a target XML file. As shown above, the source XML contains information about a fictitious company. The company is divided into offices. Offices are sub-divided into departments, and departments are further divided into people.

The target XML component, **PersonList**, contains a list of **Person** records. The **Details** item is meant to store information about the office and department where the person belongs.

The aim is to extract all persons from the source XML and sort them alphabetically by last name. Also, the office and department name where each person belongs must be written to the **Details** item.

To achieve this goal, this example makes use of the following component types:

1. **The concat function.** In this mapping, this function returns a string in the format **Office(Department)**. It takes as input the office name, the department name, and two constants which supply the start and end brackets. See also Working with Functions.
2. **An intermediate variable.** The role of the variable is to bring all data relevant to a person into the same mapping context. The variable causes the mapping to look up the department and office of each person, in the context of each person. To put it differently, the variable "remembers" the office and department name to which a person belongs. Without the variable, the context would be incorrect, and the mapping would produce...
unwanted output (for more information about how a mapping is executed, see *Mapping Rules and Strategies*). Notice that the variable replicates the structure of the target XML file (it uses the same XML schema). This makes it possible to connect the sort result to the target, through a Copy-All connection. See also *Using Variables* and *Copy-All Connections*.

3. A Sort component, which performs the actual sorting. Notice that the key input of the Sort component is connected to the Last item of the variable, which sorts all person records by their last name.
5.9 Grouping Data

When your mapping must group nodes or rows, you can achieve this with the help of the following MapForce built-in functions:

- `group-by`
- `group-adjacent`
- `group-into-blocks`
- `group-starting-with`
- `group-ending-with`

To use any of these functions on the mapping, drag them from the Libraries window onto the mapping area. See also Add a Built-in Function to the Mapping.

Note: Grouping functions are not available if your mapping uses XSLT 1.0 as transformation language.

The following sections provide typical examples of use for grouping functions. These examples are accompanied by the following demo mapping: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\GroupingFunctions.mfd`. Note that the demo mapping contains multiple transformations, one for each function. Since only one output can be previewed at a time, remember to click the Preview button applicable to the desired transformation before clicking the Output tab.

**group-by**

The `group-by` function creates groups of records according to some grouping key that you specify. For example, in the abstract transformation illustrated below, the grouping key is "Department". Since there are three unique departments in total, applying the `group-by` function would create three groups:

For more information, see the reference to the `group-by` function.

**group-adjacent**

The `group-adjacent` function requires a grouping key as argument, similar to `group-by` function. Unlike `group-by`, this function creates a new group whenever the next key is different. If two adjacent records have the same key, they will be placed into the same group.
For example, in the abstract transformation illustrated below, the grouping key is “Department”. The left side of the diagram shows the input data while the right side shows the output data after grouping. The following takes place when the transformation runs:

- Initially, the first key, “Administration”, creates a new group.
- The next key is different, so a second group is created, “Marketing”.
- The third key is also different, so another group is created, “Engineering”.
- The fourth key is the same as the third; therefore, this record is placed in the already existing group.
- Finally, the fifth key is different from the fourth, and this creates the last group.

As illustrated below, “Michelle Butler” and “Fred Landis” were grouped together because they have the same key and are adjacent. However, “Vernon Callaby” and “Frank Further” are in separate groups because they are not adjacent, even though they have the same key.

For more information, see the reference to the group-adjacent function.

**group-into-blocks**

The group-into-blocks function creates equal groups that contain exactly N items, where N is the value you supply to the block-size argument. Note that the last group may contain N items or less, depending on the number of items in the source. In the example below, block-size is 2. Since there are five items in total, each group contains exactly two items, except for the last one.
The `group-starting-with` function takes a Boolean condition as argument. If the Boolean condition is true, a new group is created, starting with the record that satisfies the condition. In the example below, the condition is that "Key" must be equal to "heading". This condition is true for the first and fourth records, so two groups are created as a result:

For more information, see the reference to the `group-starting-with` function.

The `group-ending-with` function takes a Boolean condition as argument. If the Boolean condition is true, a new group is created, ending with the record that satisfies the condition. In the example below, the condition is that "Key" must be equal to "trailing". This condition is true for the third and fifth records, so two groups are created as a result:

For more information, see the reference to the `group-ending-with` function.
Note: One additional group is created if records exist after the last one that satisfies the condition. For example, if there were more "line" records after the last "trailing" record, these would all be placed into a new group.

For more information, see the reference to the group-ending-with function.

5.9.1 Example: Grouping Records by Key

This example shows you how to group records with the help of the group-by function, and also illustrates how to aggregate data. This example is accompanied by a demo mapping available at the following path: \Documents\Altova\MapForce2019\MapForceExamples\GroupTemperaturesByYear.mfd. This mapping reads data from an XML file that contains a log of monthly temperatures, as illustrated in the code listing below:

```
<Temperatures>
  <data temp="-3.6" month="2006-01" />
  <data temp="-0.7" month="2006-02" />
  <data temp="7.5" month="2006-03" />
  <data temp="12.4" month="2006-04" />
  <data temp="16.2" month="2006-05" />
  <data temp="19" month="2006-06" />
  <data temp="22.7" month="2006-07" />
  <data temp="23.2" month="2006-08" />
  <data temp="18.7" month="2006-09" />
  <data temp="11.2" month="2006-10" />
  <data temp="9.1" month="2006-11" />
  <data temp="0.8" month="2006-12" />
  <data temp="-3.2" month="2007-01" />
  <data temp="-0.3" month="2007-02" />
  <data temp="6.5" month="2007-03" />
  <data temp="10.6" month="2007-04" />
  <data temp="19" month="2007-05" />
  <data temp="20.3" month="2007-06" />
  <data temp="22.3" month="2007-07" />
  <data temp="20.7" month="2007-08" />
  <data temp="19.2" month="2007-09" />
  <data temp="12.9" month="2007-10" />
  <data temp="8.1" month="2007-11" />
</Temperatures>
```
The business requirement of this mapping is two-fold:

1. Group temperatures of each year together.
2. Find out the minimum, maximum, and the average temperature of each year.

To achieve the first goal, the mapping calls the group-by function. To achieve the second goal, it calls the min, max, and avg aggregation functions. All of these functions are MapForce built-in functions, and you can add them to any mapping by dragging them from the Libraries window, see How to Add a Built-in Function to the Mapping.

The way MapForce executes a mapping (and the recommended approach to start reading one) is by looking at the topmost item of the target component. In this example, an YearlyStats item will be created for each group returned by the group-by function. The group-by function takes as first argument all data items from the source and groups them by whatever is connected to the key input. Since the requirement is to group temperatures by year, the year must be obtained first. To achieve this, the substring-before function extracts the year part from the month attribute of each data element. Namely, it takes as argument the value of month and returns the part before the first occurrence of \texttt{substr}. As illustrated above, in this example, \texttt{substr} is set to the dash character; therefore, if given the value "2006-01", the function will return "2006".

Finally, the values of \texttt{MinimumTemp}, \texttt{MaximumTemp}, and \texttt{AverageTemp} are obtained by connecting these items with the respective aggregate functions: min, max, and avg. All three functions take as input the sequence of temperatures read from the source component. These functions do not need a parent-context argument, because they already work in the context of each group. In other words, there is a parent connection—from data to YearlyStats—which provides the context for each aggregation function to work on.

To preview the mapping output, click the Output tab. Notice that the number of groups coincides with the number of years obtained by reading the source file, for example:
<Temperatures>
  <YearlyStats Year="2006">
    <MinimumTemp>-3.6</MinimumTemp>
    <MaximumTemp>23.2</MaximumTemp>
    <AverageTemp>11.375</AverageTemp>
  </YearlyStats>
  <YearlyStats Year="2007">
    <MinimumTemp>-3.2</MinimumTemp>
    <MaximumTemp>22.3</MaximumTemp>
    <AverageTemp>11.5</AverageTemp>
  </YearlyStats>
</Temperatures>

Note: For simplicity, the code listings above contain less data than the actual input and output used by the demo mapping.
5.10 Filters and Conditions

When you need to filter data, or get a value conditionally, you can use one of the following component types:

- Filter: Nodes/Rows (.getLongitude)
- SQL WHERE/ORDER (.getLongitude)
- If-Else Condition (.getLongitude)

You can add these components to the mapping either from the Insert menu, or from the Insert Component toolbar. Importantly, each of the components above has specific behavior and requirements. The differences are explained in the following sections.

Filtering nodes or rows

When you need to filter data, including XML nodes or CSV rows, use a Filter Nodes/Rows component. The Filter Nodes/Rows component enables you to retrieve a subset of nodes from a larger set of data, based on a true or false condition. Its structure on the mapping area reflects this:

In the structure above, the condition connected to bool determines whether the connected node/row goes to the on-true or on-false output. Namely, if the condition is true, the node/row will be redirected the on-true output. Conversely, if the condition is false, the node/row will be redirected to the on-false output.

When your mapping needs to consume only items that meet the filter condition, you can leave the on-false output unconnected. If you need to process the items that do not meet the filter condition, connect the on-false output to a target where such items should be redirected. If you want to add an exception when the filter condition is not met, connecting the on-false output is mandatory (see Adding Exceptions).

For a step-by-step mapping example, see Example: Filtering Nodes.

Filtering database data

Filter Nodes/Rows components can filter data from any other component structure supported by MapForce, including databases. However, if you want to filter data from a database, it is recommended to use a SQL WHERE/ORDER component instead. The SQL WHERE/ORDER component is optimized for working with databases and provides better performance than a Filter Nodes/Rows component.

For more information about such components, see SQL WHERE / ORDER Component.
Returning a value conditionally

If you need to get a single value (not a node or row) conditionally, use an **If-Else Condition**. Note that If-Else conditions are not suitable for filtering nodes or rows. Unlike **Filter Nodes/Rows** components, an **If-Else Condition** returns a value of simple type (such as a string or integer). Therefore, **If-Else Conditions** are only suitable for scenarios where you need to process a simple value conditionally. For example, let's assume you have a list of average temperatures per month, in the format:

```xml
<Temperatures>
  <data temp="19.2" month="2010-06" />
  <data temp="22.3" month="2010-07" />
  <data temp="19.5" month="2010-08" />
  <data temp="14.2" month="2010-09" />
  <data temp="7.8" month="2010-10" />
  <data temp="6.9" month="2010-11" />
  <data temp="-1.0" month="2010-12" />
</Temperatures>
```

An **If-Else Condition** would enable you to return, for each item in the list, the value "high" if temperature exceeds 20 degrees Celsius, and value "low" if temperature is lower than 5 degrees Celsius.

On the mapping, the structure of the **If-Else Condition** looks as follows:

```
<table>
<thead>
<tr>
<th>if-else</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;bool&gt;</td>
</tr>
<tr>
<td>&lt;value-true&gt;</td>
</tr>
<tr>
<td>&lt;value-false&gt;</td>
</tr>
</tbody>
</table>
```

If the condition connected to `bool` is true, then the value connected to `value-true` is output as `result`. If the condition is false, the value connected to `value-false` is output as `result`. The data type of `result` is not known in advance; it depends on the data type of the value connected to `value-true` or `value-false`. The important thing is that it should always be a simple type (string, integer, and so on). Connecting input values of complex type (such as nodes or rows) is not supported by **If-Else Conditions**.

If-Else Conditions are extendable. This means that you can add multiple conditions to the component, by clicking the **Add** button. To delete a previously added condition, click the **Delete** button. This feature enables you to check for multiple conditions and return a different value for each condition, if it is true.
Expanded **If-Else Conditions** are evaluated from top to bottom (first conditions is checked first, then the second one, and so on). If you want to return a value when none of the conditions are true, connect it to `otherwise`.

For a step-by-step mapping example, see [Example: Returning a Value Conditionally](#).

### 5.10.1 Example: Filtering Nodes

This example shows you how to filter nodes based on a true/false condition. A **Filter: Nodes/Rows** component is used to achieve this goal. The technique illustrated in this example works in the same way not only for XML, but also for other component types, such as CSV or text. In case of databases, although you can use a filter, it is recommended to use a SQL WHERE/ORDER component instead, for better performance (see [SQL WHERE / ORDER Component](#)).

The mapping described in this example is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\MarketingExpenses.mfd`.

As shown above, the mapping reads data from a source XML which contains an expense report ("ExpReport") and writes data to a target XML ("MarketingExpenses"). There are several other components between the target and source. The most relevant component is the **expense-item**.
filter (🔍), which represents the subject of this topic.

The goal of the mapping is to filter out only those expense items that belong to the Marketing department. To achieve this goal, a filter component has been added to the mapping. (To add a filter, click the Insert menu, and then click Filter: Nodes/Rows.)

To identify whether each expense item belongs to Marketing, this mapping looks at the value of the "expto" attribute in the source. This attribute has the value "Marketing" whenever the expense is a marketing expense. For example, in the code listing below, the first and third expense item belongs to Marketing, the second belongs to Development, and the fourth belongs to Sales:

```
...  
  <expense-item type="Meal" expto="Marketing">
    <Date>2003-01-01</Date>
    <expense>122.11</expense>
  </expense-item>
  <expense-item type="Lodging" expto="Development">
    <Date>2003-01-02</Date>
    <expense>122.12</expense>
  </expense-item>
  <expense-item type="Lodging" expto="Marketing">
    <Date>2003-01-02</Date>
    <expense>299.45</expense>
  </expense-item>
  <expense-item type="Entertainment" expto="Sales">
    <Date>2003-01-02</Date>
    <expense>13.22</expense>
  </expense-item>
...  
```

*XML input before the mapping is executed*

On the mapping area, the node/row input of the filter is connected to the expense-item node in the source component. This ensures that the filter component gets the list of nodes that it must process.

To add the condition based on which filtering should occur, we have added the equal function from the MapForce core library (for more information, see Working with Functions). The equal function compares the value of the "type" attribute to a constant which has the value "Marketing". (To add a constant, click the Insert menu, and then click Constant.)

Since we need to filter only those items that satisfy the condition, we connected only the on-true output of the filter to the target component.

When you preview the mapping result, by clicking the Output tab, MapForce evaluates, for each expense-item node, the condition connected to the bool input of the filter. When the condition is true, the expense-item node is passed on to the target; otherwise, it is ignored. Consequently, only the expense items matching the criteria are displayed in the output:

```
...  
  <expense-item>
```
5.10.2 Example: Returning a Value Conditionally

This example shows you how to return a simple value from a component, based on a true/false condition. An *If-Else Condition* is used to achieve the goal. Note that *If-Else Conditions* should not be confused with filter components. *If-Else Conditions* are only suitable when you need to process simple values conditionally (string, integer, etc.). If you need to filter complex values such as nodes, use a filter instead (see Example: Filtering Nodes).

The mapping described in this example is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\ClassifyTemperatures.mfd`.

This mapping reads data from a source XML which contains temperature data ("Temperatures") and writes data to a target XML which conforms to the same schema. There are several other
components between the target and source, one of them being the \textbf{if-else} condition (highlighted in red), which is also the subject of this topic.

The goal of the mapping is to add short description to each temperature record in the target. Specifically, if temperature is above 20 degrees Celsius, the description should be "high". If the temperature is below 5 degrees Celsius, the description should be "low". For all other cases, no description should be written.

To achieve this goal, conditional processing is required; therefore, an If-Else Condition has been added to the mapping. (To add an If-Else Condition, click the \textbf{Insert} menu, and then click \textbf{If-Else Condition}.) In this mapping, the If-Else Condition has been extended (with the help of the \textbf{button}) to accept two conditions: bool1 and bool2.

The conditions themselves are supplied by the \textbf{greater} and \textbf{less} functions, which have been added from the MapForce core library (for more information, see \textit{Working with Functions}). These functions evaluate the values provided by two input components, called "upper" and "lower". (To add an input component, click the \textbf{Insert} menu, and then click \textbf{Insert Input}. For more information about input components, see \textit{Supplying Parameters to the Mapping}.)

The \textbf{greater} and \textbf{less} functions return either true or false. The function result determines what is written to the target instance. Namely, if the value of the "temp" attribute in the source is greater than 20, the constant value "high" is passed to the \textbf{if-else} condition. If the value of the "temp" attribute in the source is less than 5, the constant value "low" is passed on to the \textbf{if-else} condition. The \textbf{otherwise} input is not connected. Therefore, if none of the above conditions is met, nothing is passed to the \textbf{result} output connector.

Finally, the \textbf{result} output connector supplies this value (once for each temperature record) to the "desc" attribute in the target.

When you are ready to preview the mapping result, click the \textbf{Output} tab. Notice that the resulting XML output now includes the "desc" attribute, whenever the temperature is either greater than 20 or lower than 5.

\begin{quote}
\begin{verbatim}
...<data temp="-3.6" month="2006-01" desc="low"/>
<data temp="-0.7" month="2006-02" desc="low"/>
<data temp="7.5" month="2006-03"/>
<data temp="12.4" month="2006-04"/>
<data temp="16.2" month="2006-05"/>
<data temp="19" month="2006-06"/>
<data temp="22.7" month="2006-07" desc="high"/>
<data temp="23.2" month="2006-08" desc="high"/>
...
\end{verbatim}
\end{quote}

\textit{XML output after the mapping is executed}
5.11 Joining Data

Sometimes, you may need to combine data from two or more structures based on some condition (for example, if field A in the first structure has the same value as field B in the second structure). For such mapping requirements, a Join component can be used.

A Join component is a MapForce component which enables joining two or more structures on the mapping based on custom-defined conditions. It returns the association (joined set) of items that satisfy the condition. Joins are particularly useful to combine data from two structures which share a common field (such as an identity).

For example, on the mapping illustrated below, the middle component is a "Join" component. In this mapping, two XML structures (a list of people and a list of addresses) are being joined. The goal here is to get the full details of each person into a target XML file. The FirstName and LastName fields act as joining keys. Namely, if value of FirstName and LastName (under Person) is the same as that of FirstName and LastName (under Address), the address details belong to one and the same person and they become "joined". Any items from the joined structure can further be mapped to a subsequent target (in this case, an XML file). The join condition itself is defined in the properties of the Join component, by clicking the Define Join Condition button. This example is accompanied by a mapping sample and is explained in more detail in Example: Join XML Structures.

As illustrated above, the source structures and the Join component are connected by means of "Copy-All" connection, which reduces the mapping clutter. In general, such connections are created automatically by MapForce when the context is relevant (for more information, see Copy-All Connections).

The structures that are to be joined may either be from separate components (as in the mapping above), or belong to the same component. The structures to be joined may also be of different kinds (for example, an XML structure and a database table). For more information about database-
related joins, see Joining Database Data.

To add a Join component:

1. Set the mapping transformation language to BUILT-IN (to do this, either click the toolbar button, or use the Output | Built-In Execution Engine menu command).
2. On the Insert menu, click Join. Alternatively, click the Join toolbar button. The Join component appears on the mapping. By default, it accepts data from two structures, so it has two nodes/rows inputs. If necessary, you can add new inputs to the join by clicking the Add Input (a) button, see Joining Three or More Structures.

3. Connect the structures that are to be joined to the nodes/rows items of the join component.
4. Add the condition for the join (or multiple conditions). To do this, right-click the Join component and select Properties. Join conditions can also be added directly from the mapping, by connecting the Boolean result of some function to the condition item of the Join component. In certain cases when database tables are joined, the join condition (or conditions) can be created automatically by MapForce. For further information, see Adding Join Conditions.

Notes:

- Join components are supported when the target language of the mapping is set to BUILT-IN. Code generation in C#, C++, or Java is not supported.
- When a structure is not a valid or supported input source for the join, MapForce displays hints either immediately directly on the mapping, or in the Messages window, when you validate the mapping (see Validating Mappings).
- Join components should not be connected to input parameters or results of inline user-defined functions. If such connections exist, validation errors will occur during mapping validation.
- When you connect eligible database components (such as tables or views) directly to a Join component, an SQL mode (SQL) button automatically appears at the top-right corner of the Join component. When enabled, this button provides special SQL features applicable to the join operation (see About Joins in SQL Mode).
- It is not possible to connect the output of the joined item to another Join component. If necessary, however, you can connect a partial result of one join to another one.

Join components compared to other component types

In some cases, complex variables or filters can be used instead of Join components to achieve the same results (see Using Variables and Filters and Conditions, respectively). However, unlike other component types, Join components make the mapping easier to understand, because you can see at a glance the data that is being joined. Additionally, if SQL mode is enabled on the join
component, the mapping performance improves significantly (this applies to database joins, see Joining Database Tables).

Adding a parent context
In some special cases, in order to achieve a specific mapping result, you can explicitly provide a mapping context (a so-called "parent context") for data connected to the Join component. To add a parent context, right-click the joined item of the Join component, and select Add Parent Context from the context menu. The Join component changes appearance to include an additional parent-context input where you can connect the required source item. For more information, see Overriding the Mapping Context.

5.11.1 Adding Join Conditions

A join works by combining items of two or more structures according to a condition, so a join always requires at least one condition. There are several ways to add join conditions, as shown below.

Note: When database tables are joined in SQL mode, MapForce will create the join condition (or conditions) automatically, based on foreign key relationships detected between tables. For automatic join conditions to happen, the database tables must be in a child-parent relationship on the MapForce component (that is, one table must be "parent" or "child" of another one on the component), see Example: Join Tables in SQL Mode.

Approach 1: Add a join condition from the component properties
1. On the mapping, make sure that at least two structures (or database tables) are connected to the Join component. The Join component illustrated in this example is part of the JoinPeopleInfo.mfd mapping available in the folder <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\. This mapping is discussed in more detail in Example: Join XML Structures.
2. On the Join component, click the Define Join Condition (_condition) button (or right-click the header of the component, and select Properties from the context menu).
3. Select an item from the left structure and another one from the right structure (that is, whenever the comparison of this pair returns true, the left and right structures become joined).
If you need to add multiple conditions, click **Add Condition**, and then select a new pair of items. For example, in the image above, two join conditions are defined:

1. **FirstName** in the Structure 1 must be equal to **FirstName** in Structure 2, and
2. **LastName** in Structure 1 must be equal to **LastName** in Structure 2.

To remove a join condition, click the **Delete** button next to it.

**Notes:**

- When multiple join conditions exist, all of them must be satisfied in order for the two structures to be joined. In other words, multiple conditions are joined by a logical AND operation. This also includes optional conditions that were added from the mapping (see Approach 2 below).
- If more than two structures are connected to the Join component, such additional structures appear in the drop-down list below “Structure 2”. When you select such an additional structure from the drop-down list, the left pane displays all structures that occur **before** it on the Join component. This way you can define join conditions between any of the multiple structures. For an example, see Example: Create CSV Report from Multiple Tables.
To view the data type of items in each structure, select the **Show types** check box. The **Show annotations** option displays additional information about items, provided that such information exists in the underlying schema (or database). If both check boxes are selected, the layout changes to accommodate the display of both annotations and types, for example:

![Diagram showing mapping components and data types]

**Approach 2: Add a join condition from the mapping**

- On the mapping, add components which produce a Boolean value, and then connect the Boolean output to the input of the **condition** item. For example, the **equal** function may compare a value with some mapping item, and supply the Boolean result as input to the **condition** item of the join component.
5.11.2 Joining Three or More Structures

When you add a Join component to the mapping using the menu command Insert | Join, it accepts two structures by default (that is, the component contains only two nodes/rows inputs).

If you need to join more than two structures, click the Add input ( addButton and create as many nodes/rows as necessary. If you need to remove a nodes/rows input, click the Delete input ( ) button. Note that a join requires at least two structures, so the button is only available when more than two inputs exist.

Note: If no condition is defined from the join component properties (Approach 1), the condition item of the join component must be connected (Approach 2).

Approach 3: Mixed approach

In the same mapping, it is possible to define some join conditions in component properties (Approach 1) and combine them with the one from the mapping (Approach 2). However, if you intend to join database tables in SQL mode, the conditions must be defined strictly using Approach 1 (see also About Joins in SQL Mode).
When a join has multiple inputs, the join conditions must accordingly take into consideration each of the inputs that you want to be joined, see Adding Join Conditions. For a step-by-step example of how to join multiple database tables, see Example: Create CSV Report from Multiple Tables.

5.11.3 Example: Join XML Structures

This example shows you how to combine data from two XML structures conditionally, by using a join component. The example is accompanied by a mapping sample which is available at the following path: <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\JoinPeopleInfo.mfd.

The purpose of this mapping is to collect people data (name, surname, address, email, and phone) from two source XML files into a single target XML file.

The first XML file stores the name and surname of each person, as well as their email and phone, as shown in the sample code listing below (note that the XML declaration, namespaces, and some records have been omitted, for simplicity):

```xml
<People>
  <Person>
    <FirstName>Marquita</FirstName>
    <LastName>Bailey</LastName>
    <Email>m.bailey@nanonull.com</Email>
    <Phone>555323698</Phone>
  </Person>
  <Person>
    <FirstName>Totie</FirstName>
    <LastName>Rea</LastName>
    <Email>t.rea@nanonull.com</Email>
    <Phone>555598653</Phone>
  </Person>
</People>
```

People.xml

The second XML file stores the name and surname of each person, as well as their address details:

```xml
<Addresses>
  <Address>
```

Addresses.xml
The goal of the mapping is to combine the <Person> information from the first file with <Address> information from the second file, wherever the first and last names match. Specifically, for each <Person> in the first file, and for each <Address> in the second file, the FirstName and LastName must be compared. If both values are the same, then the corresponding <Person> and <Address> records refer to the same person, and must be joined. The target XML structure should look like this:

```
<PeopleInfo>
  <Row>
    <FirstName>Marquita</FirstName>
    <LastName>Bailey</LastName>
    <City>Bridgedell</City>
    <Street>Olive Street</Street>
    <Number>4</Number>
    <Email>m.bailey@nanonull.com</Email>
    <Phone>555323698</Phone>
  </Row>
  <Row>
    <FirstName>Totie</FirstName>
    <LastName>Rea</LastName>
    <City>Roseford</City>
    <Street>Evergreen Lane</Street>
    <Number>34</Number>
    <Email>t.rea@nanonull.com</Email>
    <Phone>555598653</Phone>
  </Row>
</PeopleInfo>
```

This mapping goal can be easily achieved by adding a Join component to the mapping. Note that it is also possible to achieve the same result using other component types; however, in the steps below, you will be using a Join component, which is the subject of this example.

To create the required mapping, follow the steps below.
Step 1: Add the source XML files to the mapping

1. On the Insert menu, click XML Schema/File, and browse for the following source file: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\People.xml`.
2. Repeat the step above for `Addresses.xml` (the second source file).

Step 2: Add the target schema file to the mapping

- On the Insert menu, click XML Schema/File, and browse for `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\PeopleInfo.xsd` (the target XSD schema file). When prompted to supply a sample XML file, click Skip. When prompted to select a root element, select PeopleInfo as root element.

Step 3: Add the Join component

1. On the Insert menu, click Join. (Alternatively, click the Join toolbar button). At this stage, the mapping should look as follows (you will need to drag and resize the components in order to make them look as illustrated below):

2. Draw a connection from Person to the first nodes/rows item of the Join component. Likewise, connect Address to the second nodes/rows item.

Observe the structure of the Join component. It has two nodes/rows items, which makes it possible to connect to it the two structures that need to be compared (in this case, the Person and the Address structures).
3. As mentioned earlier, the join should take place only if the FirstName and LastName values are equal in both structures. To define this condition, click the Define Join Condition button.

4. Select the pair of items that define the first join condition (FirstName under Structure 1, and FirstName under Structure 2).

5. Click Add Condition, and repeat the step above for LastName.
In some mappings, a condition consisting of one comparison may be sufficient to perform the join. However, in this example, it is important that two comparisons are created:

1) **FirstName** in Structure 1 = **FirstName** in Structure 2  
2) **LastName** in Structure 1 = **LastName** in Structure 2.

When multiple conditions are defined, all of them must be true in order for the join to take place. Therefore, in this example, a join will happen only when both comparisons are true (which is the intended behaviour). Otherwise, if only one of the comparisons above were defined, a join could happen for persons that have the same first name but different last names.

**Step 4: Map the Join component to the target schema**

Now that the two structures are joined, you can define which items of the joined structure should be mapped to the target. To do this, create connections from items of both joined structures to the target component, as shown below. The connection between **joined** and **Row** has the following purpose: whenever the join condition is satisfied, it creates a new **Row** item in the target.
To preview the mapping output, click the **Output** tab. As expected, each person record (`<Row>`) now includes the full address details, joined from two different sources.
5.12 Using Value-Maps

The Value-Map component allows you to transform an input value to a different output value using a lookup table. This is useful for converting different enumeration types. The component has only one input and one output item.

Note: If you want to retrieve/filter data based on specific criteria, please use the Filter component, see Filters and Conditions.

To use a Value-Map component:

1. Select the menu option Insert | Value-Map, or click the Value-Map icon in the icon bar.

2. Double-click the Value-Map component to open the value map table.

3. Click into the column headers and enter Weekday input in the first, and Day of the Week in the second.
4. Enter the input value that you want to transform, in the **Weekday input** column.
5. Enter the output value you want to transform that value to, in the **Day of the week** column.
6. Simply type in the *(new entry)* input field to enter a new value pair.
7. Click the **datatype** combo box, below the column header to select the input and output datatypes, e.g. integer and string.

**Note:** Activate the **Otherwise** check box, and enter the value, to define an alternative output value if the supplied values are not available on input. To pass through source data without changing it please see **Passing data through a Value-Map unchanged**.

8. You can click the edit icons in the header rows to change the column names, which are also displayed in the mapping. This will make it easier to identify the purpose of the component in the mapping.

The **Expense-valmap.mfd** file in the `...\MapForceExamples\Tutorial` folder is a sample mapping that shows how the Value-Map can be used.
What this mapping does:
Extracts the day of the week from the Date item in the data source, converts the numerical value into text, and places it in the Weekday item of the target component i.e. Sunday, Monday etc.

- The `weekday` function extracts the weekday number from the `Date` item in the ExpReport source file. The result of this function are integers ranging from 1 to 7.
- The Value-Map component transforms the integers into weekdays, i.e. Sunday, Monday, etc. as shown in the graphic at the top of this section.
- If the output contains "Tuesday", then the corresponding output "Prepare Financial Reports" is mapped to the Notes item in the target component.
- Clicking the Output tab displays the target XML file with the transformed data.

```xml
3  <Name>Landis</Name>
4  <expense-item>
5  |   <type>Meal</type>
6  |   <Weekday>Tuesday</Weekday>
7  |   <Notes>-- Prepare financial reports -- !</Notes>
8  |   <Date>2003-01-01</Date>
9  |   <expense>122.11</expense>
10 <expense-item>
11 <expense-item>
12 |   <type> LODging</type>
13 |   <Weekday>Monday</Weekday>
14 |   <Notes> -- </Notes>
15 |   <Date>2003-01-14</Date>
16 |   <expense>122.12</expense>
17 </expense-item>
```

**Note:** Placing the mouse cursor over the value map component opens a popup containing the currently defined values.

The output from various types of logical, or string functions, can only be a boolean "true" or "false" value. The value you want to test for, must thus be entered into the input field of the value...
map table e.g. "true".

5.12.1 Passing data through a Value-Map unchanged

This section describes a mapping situation where some specific node data have to be transformed, while the rest of the node data have to be passed on to the target node unchanged.

An example of this would be a company that changes some of the titles in a subsidiary. In this case it might change two title designations and want to keep the rest as they currently are.

The obvious mapping would be the one shown above, which uses the value-map component to transform the specific titles. Clicking the Output tab shows us the result of the mapping:

For those persons who are neither of the two types shown in the value-map component, the Title element is deleted in the output file.
Possible alternative:
Clicking the **Otherwise** check box and entering a substitute term, does make the Title node reappear in the output file, but it now contains the same **New Title** for all other persons of the company.

**Solution**
Create a user-defined function containing the value-map component, and use the **substitute-missing** function to supply the original data for the empty nodes.

1. Click the value-map component and select **Function | Create user-defined function from Selection**.

2. Enter a name for the function e.g. Pass-Through and click OK.

3. Insert a **substitute-missing** function from the **core | node function** section of the Libraries pane, and create the connections as shown in the screen shot below.
4. Click the Output tab to see the result:

Result of the mapping:

- The two Title designations in the value-map component are transformed to New Title.
- All other Title nodes of the source file, retain their original Title data in the target file.

```xml
38   <Person>
39      <First>Fred</First>
40      <Last>Landis</Last>
41      <Title>Program Manager</Title>
42      <PhoneExt>051</PhoneExt>
43      <Email>f.landis@nanonull.com</Email>
44   </Person>
45   <Person>
46      <First>Michelle</First>
47      <Last>Butler</Last>
48      <Title>Code Magician</Title>
49      <PhoneExt>854</PhoneExt>
50      <Email>m.landis@nanonull.com</Email>
51   </Person>
```

Why is this happening:
The value-map component evaluates the input data.

- If the incoming data matches one of the entries in the first column, the data is transformed and passed on to the node parameter of substitute-missing, and then on to Title2.
- If the incoming data does not match any entry in the left column, then nothing is passed on from value-map to the node parameter i.e. this is an empty node.

When this occurs the substitute-missing function retrieves the original node and data from the Title node, and passes it on through the replace-with parameter, and then on to Title2.

### 5.12.2 Value-Map component properties

**Actions:**

- Click the insert icon to **insert** a new row before the currently active one.
Click the delete icon to **delete** the currently active row.

Click the edit icon to **edit** the column header.

You can also reorder lines by dragging them.

**Changing the column header**

Double clicking the column header, or clicking the pencil icon, allows you to edit the column name and change it to something more meaningful. This will make it easier to identify the purpose of the component, as the column names are also displayed in the mapping.

**Using unique Input values:**

The values entered into the input column must be unique. If you enter two identical values, both are automatically highlighted for you to enable you to correct one of them.

Having corrected one of the values, the OK button is again enabled.

**Input and output datatypes**

The input and result datatypes are automatically checked when a selection is made using the combo box. If a mismatch occurs, then the respective fields are highlighted and the OK button is disabled. Change the datatype to one that is supported.

In the screenshot below a boolean and string have been selected.
### Value-Map Properties

Value-Map table to map specific values to others:

<table>
<thead>
<tr>
<th>input</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>string</td>
</tr>
<tr>
<td>1</td>
<td>Sunday</td>
</tr>
<tr>
<td>2</td>
<td>Monday</td>
</tr>
<tr>
<td>3</td>
<td>Tuesday</td>
</tr>
<tr>
<td>4</td>
<td>Wednesday</td>
</tr>
<tr>
<td>5</td>
<td>Thursday</td>
</tr>
<tr>
<td>6</td>
<td>Friday</td>
</tr>
<tr>
<td>7</td>
<td>Saturday</td>
</tr>
<tr>
<td>(new entry)</td>
<td></td>
</tr>
<tr>
<td>Otherwise</td>
<td>incorrect date</td>
</tr>
</tbody>
</table>

Values mismatching their types detected.

[OK]  [Cancel]
5.13 Adding Exceptions

An exception is a special component type that enables you to stop the mapping process and
display an error when a condition returned by a filter occurs. You can add an exception when your
mapping includes a filter that checks for a true/false condition (see Filters and Conditions). For
example, you may want to throw an exception if the value of some mapping item is greater than
some custom threshold.

Exceptions can also be added when you design WSDL Web services with MapForce (see Web
Service Faults).

To add an exception to the mapping:

1. On the Insert menu, click Exception.
2. Click the Insert Exception ( ) toolbar button.
3. Connect the throw input of the exception either to an on-true or on-false output of a
filter.
4. Optionally, connect the error-text input of the exception to another component (typically,
a constant) that supplies the text of the error when this exception is thrown.

Note: Both the on-true and on-false outputs of the filter must be connected. Specifically, one
of these outputs must be connected directly to the exception (without any intermediary
functions or components). The other output must be connected to the target component,
either directly, or through other intermediary components.

When the mapping encounters an exception, you are notified about it as follows:

- In MapForce, the Messages window displays an error, and the exception text (in this
case, "Expense limit exceeded").

If the mapping language is XSLT 2.0 or XQuery, an "Execution failed" error appears in the
Messages window, and the respective XSLT2 or XQuery tab is opened. The error line is
highlighted in the Messages window.

- If you run the mapping with MapForce Server, the error "Exception was thrown!" is
returned, followed by the custom exception text you have defined in MapForce.

- If you run the mapping from the generated C#, C++, or Java code, the error "USER
EXCEPTION" is returned, followed by the custom exception text you have defined in
MapForce.
5.13.1 Example: Exception on "Greater Than" Condition

This example illustrates a mapping that throws an exception when a "Greater Than" condition occurs. The sample mapping accompanying this example can be found at: `<Documents>\Altova\MapForce2019\MapForceExamples\ExpenseLimit.mfd`.

This mapping throws an exception whenever the `expense` item in the source XML instance has a value greater than 200. The value “200” is provided by a constant. The `less` function is then used to compare the two values. If the value of `expense` is less than 200, then its parent, the `expense-item`, is passed on to the filter, and no exception is thrown. Otherwise, an exception is thrown, with the custom text “Expense limit exceed”.

As shown above, the exception is identified by the icon and it consists of two items: `throw` and `error-text`. The `throw` item must be connected to the `on-false` or `on-true` output of a filter. The `error-text` is connected to a constant which provides the custom text of the exception.

Importantly, both outputs of the filter are connected; otherwise, the exception would not be thrown. In this particular example, the `on-false` output is connected to the exception, while the `on-true` output is connected to the target component.
5.13.2 Example: Exception When Node Does Not Exist

This example illustrates how to throw an exception when a node in the source XML schema does not exist. For the sake of simplicity, this example uses the same XML schema both as source and target component.

To add the source schema to the mapping:

1. On the Insert menu, click XML Schema/File, and browse for `<Documents>\Altova\MapForce2019\MapForceExamples\BookList.xsd`.
2. When prompted to provide an instance file, click Skip.
3. When prompted to select a schema root element, select BookList as root element.

To add the target schema, follow the same steps. Then, using the corresponding commands from Insert menu (or the corresponding toolbar buttons), add the following:

- A Filter: Nodes/Rows component (see also Filters and Conditions)
- A constant with the text "No year defined!"
- An exception

Finally, drag the exists function from the Libraries window into the mapping area, and make the connections as illustrated below.

According to the XML schema, all attributes of the Book element are optional, except the book title. Therefore, the "Year" attribute may or may not exist in a valid XML instance. The goal of the mapping is to process successfully an XML instance where the "Year" attribute exists for each book. Otherwise, the mapping must throw an exception.

To test the successful execution of the mapping:

1. Double-click the header of the source component and, next to Input XML file, browse for the following file: `<Documents>\Altova\MapForce2019\MapForceExamples\BookList.xml`.
2. Click the Output button to run the mapping.
To test the exception:

1. Create, in the same directory, a copy of the BookList.xml file called BookListInvalid.xml.
2. Modify it so as to remove the "Year" attribute from a Book element.
3. Double-click the header of the source component, and, next to Input XML file, browse for the BookListInvalid.xml file.
4. Click the Output button to run the mapping.

Let's now have a closer look at how the mapping works.

Connection A ensures that a book in the target instance is created for each book in the source instance. Connections B, C, D, E ensure that the "Title", "Year", "Price", and "Author" are copied from the source to the target, for each book.

Connection F triggers the exists function to check for the existence of the "Year" attribute. Connection G passes the function result (true or false) to the filter. If the result is true, the "Year" attribute exists, and the book is passed on to the filter, and subsequently to the target through connection H.

Notice that the filter was not connected directly to the Year output of the source component. Had we done so, the filter would filter the Year by its own existence, which is not meaningful, and the exception would never be thrown.

Connection I is there because the exception must be connected either to an on-false or on-true output of a filter, according to the rules. Finally, connection K passes the custom error text from the constant to the exception component.
5.14 Parsing and Serializing Strings

String parsing and serialization is an advanced mapping technique that enables you to configure the component to either parse data from a string, or serialize data to a string. This technique can be regarded as an alternative to reading data from (or writing data to) files. MapForce components which parse strings or serialize data to strings can be useful in a variety of situations, for example:

- You need to insert structures such as XML, text, or JSON files into database fields or Excel spreadsheet cells.
- You need to convert XML fragments stored in database fields into standalone XML files.
- You have legacy data stored as text (for example, fixed-length content in a single database field), and you would like to convert this data into a fully sortable, field-based structure (using FlexText, for example).

String parsing and serialization is available for the following MapForce component types:

- Text (CSV, fixed-length field text, EDI, and MapForce FlexText templates)
- JSON schema files
- XML schema files

String parsing and serialization is supported in MapForce target languages as follows.

<table>
<thead>
<tr>
<th>Language</th>
<th>Reading</th>
<th>Writing</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUILT-IN (preview the mapping transformation)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BUILT-IN (run the MapForce Server execution file)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

This section includes the following topics:

- About the Parse/Serialize Component
- Example: Parse String (Fixed-Length Text to Excel)
- Example: Serialize to String (XML to Database)

5.14.1 About the Parse/Serialize Component

A Parse/Serialize component in MapForce is a hybrid component which is neither a source nor a target component. Given the role they play in the mapping design, such components must be placed in between other source and target components.

You can use a "Parse/Serialize String" component for string parsing when, for some reason, you need to convert a string that has structure (for example, some XML stored as string in a database) into another format. Parsing data from the source string to the "Parse/Serialize" component means that the source string is turned into a MapForce structure, and, thus, you get access to any element or attribute of the source XML stored as string.
The diagram above illustrates the typical structure of a MapForce component which parses a string. Note that the "Parse/Serialize String" component is placed in between the source and target of the mapping. What this component does is accept some string structure as input, by means of a single MapForce connector which is connected to its top String node. The output structure can be any of the data targets supported by MapForce. For an example, see Example: Parse String (Fixed-Length Text to Excel).

When you serialize data from a component to string, the reverse happens. Specifically, the entire structure of the MapForce component becomes a string structure which you can further manipulate as necessary. For example, this enables you to write an XML file (or XML fragment) to a database field or to a single cell of an Excel spreadsheet.

The diagram above illustrates a generic MapForce "Serialize to String" component. What this component does is accept as input any data source supported by MapForce (by means of standard MapForce connectors). The output structure is a string which you can pass further by means of a single MapForce connector drawn from the top String node of the component to a target component item (for example, a spreadsheet cell). For an example, see Example: Serialize...
You can designate a component for string parsing or serialization at any time from the mapping window. To do so, click the File/String (File/String) button adjacent to the root node, and then select the desired option.

### Changing the component mode

**Note:** A "Parse/Serialize String" component cannot read data from a string and write to a string simultaneously. Therefore, the root node can have either an incoming connector or an outgoing connector (not both). An error will be generated if you attempt to use the same component for both operations.

When you designate a component for string parsing or serialization, the appearance of component changes as follows:

- The component gets the **parse** or **serialize** prefix in the title.
- The title bar has yellow background color, similar to function components.
- The top node begins with the **String** prefix and is identified by the ☑ icon.
- If the component parses a string, the output connector from the root node is not meaningful and thus it is not available.
- If the component serializes to a string, the input connector to the root node is not meaningful and thus it is not available.

When a component is in "Parse/Serialize String" mode, you can change its settings in a similar way as if it were in a file-based mode (see Changing the Component Settings). Note that not all component settings are available when a component is in either "Parse" or "Serialize" mode.

#### 5.14.2 Example: Parse String (Fixed-Length Text to Excel)

This example walks you through the steps required to create a mapping design which parses string data. The example is accompanied by a sample file. If you want to look at the sample file before starting this example, you can open it from the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ParseString.mfd`.

Let's assume a scenario where you have some legacy text data stored as a single database field.
The text data is a list of employees, stored in the `RESOURCE` column, and formatted as fixed-length fields, as follows:

<table>
<thead>
<tr>
<th>ID</th>
<th>RESOURCE</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P00001 Callaby</td>
<td>582 Office Manager</td>
</tr>
<tr>
<td>2</td>
<td>P00002 Further</td>
<td>671 Accounts Receivable</td>
</tr>
<tr>
<td>3</td>
<td>P00003 Matise</td>
<td>963 Accounting Manager</td>
</tr>
<tr>
<td>4</td>
<td>P00004 Firstbread</td>
<td>62 Marketing Manager Europe</td>
</tr>
<tr>
<td>5</td>
<td>P00005 Sanna</td>
<td>753 Art Director</td>
</tr>
<tr>
<td>6</td>
<td>P00006 Landis</td>
<td>654 Software Engineer</td>
</tr>
<tr>
<td>7</td>
<td>P00007 Butler</td>
<td>655 Software Engineer</td>
</tr>
<tr>
<td>8</td>
<td>P00008 Way</td>
<td>961 Technical Writer</td>
</tr>
<tr>
<td>9</td>
<td>P00009 Gardner</td>
<td>753 Software Engineer</td>
</tr>
<tr>
<td>10</td>
<td>P00011 Smith</td>
<td>334 Software Engineer</td>
</tr>
</tbody>
</table>

Because the text data is stored as a single database field, you cannot easily access and manipulate each individual employee record. This makes it difficult to add or remove new employees, or to sort data. For the purpose of this example, your goal is to extract the text data from the `RESOURCE` database field and split it into a structure so that you can easily process the records.

This task can be accomplished by using a "Parse/Serialize String" component. First, the "Parse/Serialize String" component will take the text data as input. Then it will parse it and convert it into a structure. Finally, it will write the structure to a target format. In this example, the target format is an Excel spreadsheet; however, in general, it can be any other output format supported by MapForce.

To summarize, the mapping described in this example will convert the contents of the `RESOURCE` database field to a table. After the mapping is executed, each table row will correspond to an employee and each column will correspond to one of the fixed-length fields, in this order: ID, Last Name, First Name, Extension, Job Title, Email.

Expected output after parsing the string

To accomplish the goal, follow the steps below:

1. Add to the mapping area the source database. The database is available as a standalone SQLite database file at the following path: `<Documents>\Altova\MapForce2019`
2. When prompted to insert the database objects, select the `resources` table.

3. Add to the mapping area a Text component (use the **Insert | Text File** menu command). Since the source data is field-delimited text, choose **Use simple processing...** when prompted by MapForce.

4. Configure the structure of the Text component to map to the structure of the source text data stored in the database. As you may have already noticed, in this example, the source text consists of six fixed-length fields of fixed size, as follows:

<table>
<thead>
<tr>
<th>Size in characters</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>ID</td>
</tr>
<tr>
<td>15</td>
<td>Surname</td>
</tr>
<tr>
<td>15</td>
<td>First Name</td>
</tr>
<tr>
<td>3</td>
<td>Extension</td>
</tr>
<tr>
<td>25</td>
<td>Job Title</td>
</tr>
<tr>
<td>25</td>
<td>Email</td>
</tr>
</tbody>
</table>

To achieve this, declare the Text component as **Fixed**, and add to it six fields that correspond to positions above. (To open the Component Settings dialog box, right-click the component, and then select **Properties** from the context menu.)
Observe the Field4 field, which uses integer as data type. Although declaring the Field4 as numeric type is optional for the scope of the current example, this ensures that the phone extension (Field4) extracted from the source text is validated as a numeric value.

5. Click the File/String node, and then select Parse Strings to FLF from the context menu. This instructs MapForce that this component will parse a string to fixed-length field (FLF) format.

6. Add to the mapping area the target Excel 2007+ component. When prompted to select a sample file, click Skip. (You can add the Excel component using the Insert | Excel 2007 + menu command, see also Adding Excel 2007+ Files as Mapping Components).

7. Click the button next to Row 1, n=dyn node, and configure the Excel component to write a row for each text field, starting with the first row, as shown below. (For more information about Excel 2007+ component types and their configuration, see About the Excel 2007+ Component).
8. Draw the connections between component items, as shown below.
On the left side of the mapping, the contents of the resource database column is being converted (parsed) from a string value to a MapForce structure. On the right side of the mapping, the items of the Parse (Text file) component are connected to individual Excel columns, thus splitting the source string into individual sortable cells.

You have now finished creating a MapForce design file which parses string data and creates a structure from it. If you click the Output tab, the legacy text data is now converted to individual rows and columns of the Excel spreadsheet, which was the intended goal of this mapping.

5.14.3 Example: Serialize to String (XML to Database)

This example walks you through the steps required to create a mapping design which serializes data to a string. The example is accompanied by a sample file. If you want to look at the sample file before starting this example, you can open it from the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\SerializeToString.mfd`.

Let's assume you have an XML file (and its related schema) which consists of multiple `<Person>` elements. Each `<Person>` element describes a person's first name, last name, job title, phone extension, and email address, as follows:

```xml
<Person>
  <First>Joe</First>
  <Last>Firstbread</Last>
  <Title>Marketing Manager Europe</Title>
  <PhoneExt>621</PhoneExt>
  <Email>j.firstbread@nanonull.com</Email>
</Person>
```

Your goal is to extract each `<Person>` element from the XML file and insert it literally (including XML tags) as a new database record in the PEOPLE table of a SQLite database. The PEOPLE table contains only two columns: `ID` and `PERSON`. Its full definition is as follows:

```
CREATE TABLE PEOPLE (ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, PERSON TEXT);
```

After the mapping is executed, the expected result is that the PEOPLE table will have the same number of rows as the number of `<Person>` elements in the XML file.

To achieve the goal, do the following:
1. Add to the mapping area the source XML component (use the Insert | XML Schema/File menu command). The sample file is available at: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\MFCompany.xml`.

2. Duplicate (copy-paste) the XML component.

3. On the duplicated XML component, click File/Strings, and then select Serialize XML to Strings.

4. Right-click the duplicated component and select Change Root Element from the context menu. Then change the root element to <Person>.

In general, you can change the root element to any element that has a global (not local) declaration in the XML schema. Any elements that are not defined globally in your schema are not listed in the "Select Root Element" dialog box.

5. Double-click the component and clear the Write XML Declaration check box. This
Prevents the XML declaration from being written for each `<Person>` element.

6. Add to the mapping area the target SQLite database component, from the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\dbserialize.db`.

   (To add the database component, use the **Insert | Database** menu command, see also [Connecting to a Database](#)). When prompted to insert a database object, select the `PEOPLE` table.
7. Link the components as shown below. On the left side of the mapping, the `<Person>` element maps to the serialization component. On the right side of the mapping, the serialized string value is inserted into the `PERSON` column of the `PEOPLE` database table. Finally, the connector drawn from `<Person>` to `PEOPLE` table instructs MapForce to create a new record for each `<Person>` element encountered.

8. Click the A:In button on the database component, and do the following:
   a. Select the **Delete all records** option. At mapping runtime, this will delete any existing records from the database before new ones are inserted.
   b. Select the **DB-generated** option next to the `ID` column. This ensures that the ID of the record will be generated by the database. Note that the **DB-generated** option appears only if the column supports this option. For columns that are not an identity or auto-incremented field, the **max+1** option is available instead—this option will check what is the maximum value already existing in that column, and insert the next
available integer, incremented by 1.

You have now created a mapping design which serializes data to string. If you click the **Output** tab, the preview SQL query indicates that separate records will be inserted into the database for each `<Person>` element in the XML file, which was the goal of this mapping.
5.15 Mapping Node Names

Most of the time when you create a mapping with MapForce, the goal is to read values from a source and write values to a target. However, there might be cases when you want to access not only the node values from the source, but also the node names. For example, you might want to create a mapping which reads the element or attribute names (not values) from a source XML and converts them to element or attribute values (not names) in a target XML.

Consider the following example: you have an XML file that contains a list of products. Each product has the following format:

```
<product>
  <id>1</id>
  <color>red</color>
  <size>10</size>
</product>
```

Your goal is to convert information about each product into name-value pairs, for example:

```
<product>
  <attribute name="id" value="1" />
  <attribute name="color" value="red" />
  <attribute name="size" value="10" />
</product>
```

For such scenarios, you would need access to the node name from the mapping. With dynamic access to node names, which the subject of this topic, you can perform data conversions such as the one above.

**Note:** You can also perform the transformation above by using the `node-name` and `static-node-name` core library functions. However, in this case, you need to know exactly what element names you expect from the source, and you need to connect every single such element manually to the target. Also, these functions might not be sufficient, for example, when you need to filter or group nodes by name, or when you need to manipulate the data type of the node from the mapping.

Accessing node names dynamically is possible not only when you need to read node names, but also when you need to write them. In a standard mapping, the name of attributes or elements in a target is always known before the mapping runs; it comes from the underlying schema of the component. With dynamic node names, however, you can create new attributes or elements whose name is not known before the mapping runs. Specifically, the name of the attribute or element is supplied by the mapping itself, from any source supported by MapForce.

For dynamic access to a node's children elements or attributes to be possible, the node must actually have children elements or attributes, and it must not be the XML root node.

Dynamic node names are supported when you map to or from the following component types:
When the mapping target is a CSV or FLF (fixed-length field) file, the fields must be defined in the component settings (and it is not possible to change the name, order, or number of the target fields). Unlike XML, the format of text files is fixed, so only the actual field value can be manipulated, not the field name, number or order.

Dynamic node names are supported in any of the following mapping languages: Built-In*, XSLT2, XQuery*, C#*, C++*, Java*.

* Requires MapForce Professional or Enterprise Edition.

For information about how dynamic node names work, Getting Access to Node Names. For a step-by-step mapping example, see Example: Map Element Names to Attribute Values.

5.15.1 Getting Access to Node Names

When a node in an XML component (or a field in a CSV/FLF component) has children nodes, you can get both the name and value of each child node directly on the mapping. This technique is called "dynamic node names". "Dynamic" refers to the fact that processing takes place "on the fly", during mapping runtime, and not based on the static schema information which is known before the mapping runs. This topic provides details on how to enable dynamic access to node names and what you can do with it.

When you read data from a source, "dynamic node names" means that you can do the following:

- Get a list of all children nodes (or attributes) of a node, as a sequence. In MapForce, "sequence" is a list of zero or more items which you can connect to a target and create as many items in the target as there are items in the source. So, for example, if a node has five attributes in the source, you could create five new elements in the target, each corresponding to an attribute.
- Read not only the children node values (as a standard mapping does), but also their names.

When you write data to a target, "dynamic node names" means that you can do the following:

- Create new nodes using names supplied by the mapping (so-called "dynamic" names), as opposed to names supplied by the component settings (so-called "static" names).

To illustrate dynamic node names, this topic makes use of the following XML schema: `<Documents>`\Altova\MapForce2019\MapForceExamples\Tutorial\Products.xsd. This schema is accompanied by a sample instance document, Products.xml. To add both the schema and the instance file to the mapping area, select the Insert | XML Schema/File menu.
command and browse for `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial \Products.xml`. When prompted to select a root element, choose products.

To enable dynamic node names for the product node, right-click it and select one of the following context menu commands:

- **Show Attributes with Dynamic Name**, if you want to get access to the node’s attributes
- **Show Child Elements with Dynamic Name**, if you want to get access to the node’s children elements

![Fig. 1 Enabling dynamic node names (for child elements)](image)

**Note:** The commands above are available only for nodes that have children nodes. Also, the commands are not available for root nodes.

When you switch a node into dynamic mode, a dialog box such as the one below appears. For the purpose of this topic, set the options as shown below; these options are further discussed in Accessing Nodes of Specific Type.
Fig. 2  "Dynamically Named Children Settings" dialog box

Fig. 3 illustrates how the component looks when dynamic node names are enabled for the `product` node. Notice how the appearance of the component has now significantly changed.

Fig. 3  Enabled dynamic node names (for elements)
To switch the component back to standard mode, right-click the `product` node, and disable the option **Show Child Elements with Dynamic Name** from the context menu.

The image below shows how the same component looks when dynamic access to attributes of a node is enabled. The component was obtained by right-clicking the `product` element, and selecting **Show Attributes with Dynamic Name** from the context menu.

![Image showing enabled dynamic node names](image)

*Fig. 4  Enabled dynamic node names (for attributes)*

To switch the component back to standard mode, right-click the `product` node, and disable the option **Show Attributes with Dynamic Name** from the context menu.

As illustrated in Fig. 3 and Fig. 4, the component changes appearance when any node (in this case, `product`) is switched into "dynamic node name" mode. The new appearance opens possibilities for the following actions:

- Read or write a list of all children elements or attributes of a node. These are provided by the `element()` or `attribute()` item, respectively.
- Read or write the name of each child element or attribute. The name is provided by the `node-name()` and `local-name()` items.
- In case of elements, read or write the value of each child element, as specific data type. This value is provided by the type cast node (in this case, the `text()` item). Note that only elements can have type cast nodes. Attributes are treated always as "string" type.
- Group or filter child elements by name. For an example, see **Example: Group and Filter Nodes by Name**.

The node types that you can work with in "dynamic node name" mode are described below.

**element()**

This node has different behaviour in a source component compared to a target component. In a source component, it supplies the child elements of the node, as a sequence. In Fig. 3, `element()` provides a list (sequence) of all children elements of `product`. For example, the sequence created from the following XML would contain three items (since there are three child elements of `product`):

```xml
<product>
    <id>1</id>
</product>
```
Note that the actual name and type of each item in the sequence is provided by the `nodeName()` node and the type cast node, respectively (discussed below). To understand this, imagine that you need to transform data from a source XML into a target XML as follows:

The role of `element()` here is to supply the sequence of child elements of `product`, while `nodeName()` and `text()` supply the actual name and value of each item in the sequence. This mapping is accompanied by a tutorial sample and is discussed in more detail in Example: Map Element Names to Attribute Values.

In a target component, `element()` does not create anything by itself, which is an exception to the basic rule of mapping "for each item in the source, create one target item". The actual elements...
are created by the type cast nodes (using the value of `node-name()` and by name test nodes (using their own name).

**attribute()**
As shown in Fig. 4, this item enables access to all attributes of the node, at mapping runtime. In a source component, it supplies the attributes of the connected source node, as a sequence. For example, in the following XML, the sequence would contain two items (since `product` has two attributes):

```xml
<product id="1" color="red" />
```

Note that the `attribute()` node supplies only the value of each attribute in the sequence, always as string type. The name of each attribute is supplied by the `node-name()` node.

In a target component, this node processes a connected sequence and creates an attribute value for each item in the sequence. The attribute name is supplied by the `node-name()`. For example, imagine that you need to transform data from a source XML into a target XML as follows:

![Fig. 8 Mapping attribute values to attribute names (requirement)](image)

The mapping which would achieve this goal looks as follows:
Note: This transformation is also possible without enabling dynamic access to a node's attributes. Here it just illustrates how attribute() works in a target component.

If you want to reconstruct this mapping, it uses the same XML components as the ConvertProducts.mfd mapping available in the <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\folder. The only difference is that the target has now become the source, and the source has become the target. As input data for the source component, you will need an XML instance that actually contains attribute values, for example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<products>
  <product>
    <attribute name="id" value="1"/>
    <attribute name="color" value="red"/>
    <attribute name="size" value="big"/>
  </product>
</products>
```

Note that, in the code listing above, the namespace and schema declaration have been omitted, for simplicity.

node-name()

In a source component, node-name() supplies the name of each child element of element(), or the name of each attribute of attribute(), respectively. By default, the supplied name is of type xs:QName. To get the name as string, use the local-name() node (see Fig. 3), or use the function QName-as-string.

In a target component, node-name() writes the name of each element or attribute contained in element() or attribute().

local-name()

This node works in the same way as node-name(), with the difference that the type is xs:string instead of xs:QName.
**Type cast node**

In a source component, the type cast node supplies the value of each child element contained in `element()`. The name and structure of this node depends on the type selected from the “Dynamically Named Children Settings” dialog box (Fig. 2).

To change the type of the node, click the **Change Selection** button and select a type from the list of available types, including a schema wildcard (`xs:any`). For more information, see **Accessing nodes of specific type**.

In a target component, the type cast node writes the value of each child element contained in `element()`, as specific data type. Again, the desired data type can be selected by clicking the **Change Selection** button.

**Name test nodes**

In a source component, name test nodes provide a way to group or filter child elements from a source instance by name. You may need to filter child elements by name in order to ensure that the mapping accesses the instance data using the correct type (see **Accessing Nodes of Specific Type**). For an example, see **Example: Group and Filter Nodes by Name**.

In general, the name test nodes work almost like normal element nodes for reading and writing values and subtree structures. However, because the mapping semantics is different when dynamic access is enabled, there are some limitations. For example, you cannot concatenate the value of two name test nodes.

On the target side, name test nodes create as many elements in the output as there are items in the connected source sequence. Their name overrides the value mapped to `node-name()`.

If necessary, you can hide the name test nodes from the component. To do this, click the **Change Selection** button next to the `element()` node. Then, in the “Dynamically Named Children Settings” dialog box (Fig. 2), clear the **Show name test nodes** check box.

**5.15.2 Accessing Nodes of Specific Type**

As mentioned in the previous section, **Getting Access to Node Names**, you can get access to all child elements of a node by right-clicking the node and selecting the **Show Child Elements with Dynamic Name** context menu command. At mapping runtime, this causes the name of each child element to be accessible through the `node-name()` node, while the value—through a special type cast node. In the image below, the type cast node is the `text()` node.
Importantly, the data type of each child element is not known before the mapping runtime. Besides, it may be different for each child element. For example, a product node in the XML instance file may have a child element id of type xs:integer and a child element size of type xs:string. To let you access the node content of a specific type, the dialog box shown below opens every time when you enable dynamic access to a node’s child elements. You can also open this dialog box at any time later, by clicking the Change Selection (esture icon) button next to the element() node.
To access the content of each child element at mapping runtime, you have several options:

1. Access the content as string. To do this, select the `text()` check box on the dialog box above. In this case, a `text()` node is created on the component when you close the dialog box. This option is suitable if the content is of simple type (`xs:int`, `xs:string`, etc.) and is illustrated in the Example: Map Element Names to Attribute Values. Note that a `text()` node is displayed only if a child node of the current node can contain text.

2. Access the content as a particular complex type allowed by the schema. When custom complex types defined globally are allowed by the schema for the selected node, they are also available in the dialog box above, and you can select the check box next to them. In the image above, there are no complex types defined globally by the schema, so none are available for selection.

3. Access the content as any type. This may be useful in advanced mapping scenarios (see “Accessing deeper structures” below). To do this, select the check box next to `xs:anyType`.

Be aware that, at mapping runtime, MapForce (through the type cast node) has no information as to what the actual type of the instance node is. Therefore, your mapping must access the node content using the correct type. For example, if you expect that the node of a source XML instance may have children nodes of various complex types, do the following:

a) Set the type cast node to be of the complex type that you need to match (see item 2 in
b) Add a filter to read from the instance only the complex type that you need to match. This technique is illustrated in Example: Group and Filter Nodes by Name.

Accessing deeper structures
It is possible to access nodes at deeper levels in the schema than the immediate children of a node. It is useful for advanced mapping scenarios. In simple mappings such as Example: Map Element Names to Attribute Values, you don’t need this technique because the mapping accesses only the immediate children of an XML node. However, if you need to access deeper structures dynamically, such as “grandchildren”, “grand-grandchildren”, and so on, this is possible as shown below.

1. Create a new mapping.
2. On the Insert menu, click Insert XML Schema/File and browse for the XML instance file (in this example, the Articles.xml file from the <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ folder).
3. Right-click the Articles node and select the Show Child Elements with Dynamic Name context command.
4. Select xs:anyType from the “Dynamically Named Children Settings” dialog box.
5. Right-click the xs:anyType node and select again the Show Child Elements with Dynamic Name context command.
6. Select text() from the “Dynamically Named Children Settings” dialog box.

In the component above, notice there are two element() nodes. The second element() node provides dynamic access to grandchildren of the <Articles> node in the Articles.xml instance.

```
<?xml version=“1.0” encoding=“UTF-8”?>
<Articles xmlns:xsi=“http://www.w3.org/2001/XMLSchema-instance”>
For example, to get "grandchildren" element names \((\text{Number}, \text{Name}, \text{SinglePrice})\), you would draw a connection from the \(\text{local-name()}\) node under the second \(\text{element()}\) node to a target item. Likewise, to get "grandchildren" element values \((1, \text{T-Shirt}, 25)\), you would draw a connection from the \(\text{text()}\) node.

Although not applicable to this example, in real-life situations, you can further enable dynamic node names for any subsequent \(\text{xs:anyType}\) node, so as to reach even deeper levels.

Note the following:

- The \(\text{Type}\) button allows you to select any derived type from the current schema and display it in a separate node. This may only be useful if you need to map to or from derived schema types (see Derived XML Schema Types).
- The Change Selection (\(\text{element()}\)) button next to an \(\text{element()}\) node opens the "Dynamically Named Children Settings" dialog box discussed in this topic.
- The Change Selection (\(\text{attribute()}\)) button next to \(\text{xs:anyAttribute}\) allows you to select any attribute defined globally in the schema. Likewise, the Change Selection (\(\text{element()}\)) button next to \(\text{xs:any}\) element allows you to select any element defined globally in the schema. This works in the same way as mapping to or from schema wildcards (see also Wildcards - \(\text{xs:any} / \text{xs:anyAttribute}\)). If using this option, make sure that the selected attribute or element can actually exist at that particular level according to the schema.

### 5.15.3 Example: Map Element Names to Attribute Values

This example shows you how to map element names from an XML document to attribute values in a target XML document. The example is accompanied by a sample mapping, which is available at
the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial \ConvertProducts.mfd`.

To understand what the example does, let's assume you have an XML file that contains a list of products. Each product has the following format:

```
<product>
  <id>1</id>
  <color>red</color>
  <size>10</size>
</product>
```

Your goal is to convert information about each product into name-value pairs, for example:

```
<product>
  <attribute name="id" value="1" />
  <attribute name="color" value="red" />
  <attribute name="size" value="10" />
</product>
```

To perform a data mapping such as the one above with minimum effort, this example uses a MapForce feature known as "dynamic access to node names". "Dynamic" means that, when the mapping runs, it can read the node names (not just values) and use these names as values. You can create the required mapping in a few simple steps, as shown below.

**Step 1: Add the source XML component to the mapping**
- On the Insert menu, click XML Schema/File, and browse for the following file: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\Products.xml`. This XML file points to the Products.xsd schema located in the same folder.

**Step 2: Add the target XML component to the mapping**
- On the Insert menu, click XML Schema/File, and browse for the following schema file: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ProductValuePairs.xsd`. When prompted to supply an instance file, click Skip. When prompted to select a root element, select products as root element.

At this stage, the mapping should look as follows:
Step 3: Enable dynamic access to child nodes

1. Right-click the product node on the source component, and select Show Child Elements with Dynamic Name from the context menu.
2. In the dialog box which opens, select text() as type. Leave other options as is.

Notice that a text() node has been added on the source component. This node will supply the content of each child item to the mapping (in this case, the value of "id", "color", and "size").

Step 4: Draw the mapping connections

Finally, draw the mapping connections A, B, C, D as illustrated below. Optionally, double-click each connection, starting from the top one, and enter the text "A", "B", "C", and "D", respectively,
In the mapping illustrated above, connection A creates, for each product in the source, a product in the target. So far, this is a standard MapForce connection that does not address the node names in any way. The connection B, however, creates, for each encountered child element of product, a new element in the target called attribute.

Connection B is a crucial connection in the mapping. To reiterate the goal of this connection, it carries a sequence of child elements of product from the source to the target. It does not carry the actual names or values. Therefore, it must be understood as follows: if the source element has N child elements, create N instances of that item in the target. In this particular case, product in the source has three children elements (id, color and size). This means that each product in the target will have three child elements with the name attribute.

Although not illustrated in this example, the same rule is used to map child elements of attribute(): if the source attribute() item has N child attributes, create N instances of that item in the target.

Next, connection C copies the actual name of each child element of product to the target (literally, "id", "color", and "size").

Finally, connection D copies the value of each child element of product, as string type, to the target.

To preview the mapping output, click the Output tab and observe the generated XML. As expected, the output contains several products whose data is stored as name-value pairs, which was the intended goal of this mapping.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<products xsi:noNamespaceSchemaLocation="ProductValuePairs.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
</products>
```
Example: Group and Filter Nodes by Name

This example shows you how to design a mapping that reads key-value pairs from an XML property list (or XML plist) and writes them to a CSV file. (XML property lists represent a way of storing macOS and iOS object information in XML format, see https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/PropertyLists/UnderstandXMLPlist/UnderstandXMLPlist.html.) The example is accompanied by a mapping sample which is available at the following path: <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ReadPropertyList.mfd.

The code listing below represents the source XML file.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist SYSTEM "https://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>First Name</key>
    <string>William</string>
    <key>Last Name</key>
    <string>Shakespeare</string>
    <key>Birthdate</key>
    <integer>1564</integer>
    <key>Profession</key>
    <string>Playwright</string>
    <key>Lines</key>
    <array>
      <string>It is a tale told by an idiot,</string>
      <string>Full of sound and fury, signifying nothing.</string>
    </array>
  </dict>
</plist>
```
The goal of the mapping is to create a new line in the CSV file from certain key-value pairs found under \texttt{<dict>} node in the property list file. Specifically, the mapping must filter only \texttt{<key> - <string>} pairs. Other key-value pairs (for example, \texttt{<key> - <integer>}) must be ignored. In the CSV file, the line must store the name of the property, separated from the value of the property by a comma. In other words, the output must look as follows:

<table>
<thead>
<tr>
<th>First Name, William</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last Name, Shakespeare</td>
</tr>
<tr>
<td>Profession, Playwright</td>
</tr>
</tbody>
</table>

To achieve this goal, the mapping uses dynamic access to all children nodes of the \texttt{dict} node. Secondly, the mapping uses the \texttt{group-starting-with} function to group the key-value pairs retrieved from the XML file. Finally, the mapping uses a filter to filter only those nodes where the node name is "string".

The following steps show how the required mapping can be created.

**Step 1: Add the source XML component to the mapping**
1. Set the mapping transformation language to BUILT-IN (see Selecting a Transformation Language).
2. On the Insert menu, click XML Schema/File, and browse for the following file: <Documents>/Altova/MapForce2019/MapForceExamples/Tutorial/plist.xml. This XML file points to the plist.dtd schema located in the same folder.

**Step 2: Add the target CSV component to the mapping**
1. On the Insert menu, click Text File. When prompted, select the Use simple processing for standard CSV... option.
2. Add a CSV field to the component, by clicking Append field.
3. Double-click the name of each field, and enter "Key" as name of the first field, and "Value" as name of the second field. The "Key" field will store the name of the property, while the "Value" field will store the property value. For more information about CSV components, see CSV and Text Files.

**Step 3: Add the filter and functions**
1. Drag the equal, exists and group-starting-with functions from the Libraries window into the mapping. For general information about functions, see Working with Functions.
2. To add the filter, click the Insert menu, and then click Filter: Nodes/Rows. For general information about filters, see Filters and Conditions.
3. On the Insert menu, click Constant, and then enter the text "string".
4. In the source component, right-click the `dict` node select **Show Child Elements with Dynamic Name** from the context menu. On the "Dynamically Named Children Settings" dialog box, make sure that the check box **Show name test nodes to filter or create elements by fixed node name** is selected.

5. Draw the connections as shown below.
The mapping explained

The `element()` item on the source component provides all children of the `dict` node, as a sequence, to the `group-starting-with` function. The `group-starting-with` function creates a new group whenever a node with the name `key` is encountered. The `exists` function checks for this condition and returns the result as Boolean true/false to the grouping function.

For each group, the filter checks if the name of the current node is equal to "string", with the help of the `equal` function. The name itself is read from the `local-name()`, which supplies the node's name as a string.

The connections to the target component have the following role:

- Only when the filter condition is true, a new row is created in the target CSV.
- Key (property name) is taken from the value of the `key` element in the source.
- Value (property value) is taken from the `string name test` node.
5.16 Mapping Rules and Strategies

MapForce generally maps data in an intuitive way, but you may come across situations where the resulting output seems to have too many, or too few items. This topic is intended to help you avoid such mapping problems.

General rule
Generally, every connection between a source and target item means: for each source item, create one target item. If the source node contains simple content (for example, string or integer) and the target node accepts simple content, then MapForce copies the content to the target node and, if necessary, converts the data type.

This generally holds true for all connections, with the following exceptions:

- A target XML root element is always created once and only once. If you connect a sequence to it, only the contents of the element will be repeated, but not the root element itself, and the result might not be schema-valid. If attributes of the root element are also connected, the XML serialization will fail at runtime, so you should avoid connecting a sequence to the root element. If what you want to achieve is creating multiple output files, connect the sequence to the "File" node instead, via some function that generates file names.

- Some nodes accept a single value, not a sequence (for example, XML attributes, database fields, and output components in user-defined functions).

The "context" and "current" items
MapForce displays the structure of a schema, database, or EDI file as a hierarchy of mappable items in the component. Each of these nodes may have many instances (or none) in the instance file or database.

Example: If you look at the source component in PersonListByBranchOffice.mfd, there is only a single node first (under Contact). In the BranchOffices.xml instance file, there are multiple first nodes and Contact nodes having different content, under different Office parent nodes.

It depends on the current context (of the target node) which source nodes are actually selected and have their data copied, via the connector, to the target component/item.
This context is defined by the current target node and the connections to its ancestors:

- Initially the context contains only the source components, but no specific nodes. When evaluating the mapping, MapForce processes the target root node first (PersonList), then works down the hierarchy.
- The connector to the target node is traced back to all source items directly or indirectly connected to it, even via functions that might exist between the two components. The source items and functions results are added to the context for this node.
- For each new target node a new context is established, that initially contains all items of the parent node's context. Target sibling nodes are thus independent of each other, but have access to all source data of their parent nodes.

Applied to the example mapping above (PersonListByBranchOffice.mfd):

- The connection from Office through the filter (Office) to PersonList defines a single office as the context for the whole target document (because PersonList is the root element of the target component). The office name is supplied by the input component, which has a default containing "Nanonull, Inc."
- All connections/data to the descendants of the root element PersonList, are automatically affected by the filter condition, because the selected single office is in the context.
- The connection from Contact to Person creates one target Person per Contact item of the source XML (general rule). For each Person one specific Contact is added to the context, from which the children of Person will be created.
- The connector from first to First selects the first name of the current Contact and writes it to the target item First.

Leaving out the connector from Contact to Person would create only one Person with multiple
First, Last, and Detail nodes, which is not what we want here. In such situations, MapForce issues a warning and a suggestion to fix the problem: "You can try to connect Contact with Person to resolve":

<table>
<thead>
<tr>
<th>Warning Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>You can try to connect Contact with Person to resolve.</td>
</tr>
</tbody>
</table>

Sequences

MapForce displays the structure of a schema, database, or EDI file as a hierarchy of mappable items in the component.

Depending on the (target) context, each mappable item of a source component can represent:

- a **single instance** node of the assigned input file (or database)
- a **sequence** of zero to **multiple instance** nodes of the input file (or database)

If a sequence is connected to a **target** node, a loop is created to create as many target nodes as there are source nodes.

If a **filter** is placed between the sequence and target node, the bool condition is checked for each input node i.e. each item in the sequence. More exactly, a check is made to see if there is at least one bool in each sequence that evaluates to true. The priority context setting can influence the order of evaluation, see below.

As noted above, filter conditions automatically apply to all descendant nodes.

**Note:** If the source schema specifies that a specific node occurs exactly once, MapForce may remove the loop and take the first item only, which it knows must exist. This optimization can be disabled in the source Component Settings dialog box (check box "Enable input processing optimizations based on min/maxOccurs").

**Function inputs** (of normal, non-sequence functions) work similar to target nodes: If a sequence is connected to such an input, a loop is created around the function call, so it will produce as many results as there are items in the sequence.

If a sequence is connected to more than one such function input, MapForce creates nested loops which will process the Cartesian product of all inputs. Usually this is not desired, so only one single sequence with multiple items should be connected to a function (and all other parameters bound to singular current items from parents or other components).

**Note:** If an empty sequence is connected to such a function (e.g. concat), you will get an empty sequence as result, which will produce no output nodes at all. If there is no result in your target output because there is no input data, you can use the “substitute-missing” function to insert a substitute value.

Functions with **sequence inputs** are the only functions that can produce a result if the input sequence is empty:
• `exists`, `not-exists` and `substitute-missing` (also, `is-not-null`, `is-null` and `substitute-null`, which are aliases for the first three)
• aggregate functions (`sum`, `count`, etc.)
• regular user-defined functions that accept sequences (i.e. non-inlined functions)

The sequence input to such functions is always evaluated independently of the current target node in the context of its ancestors. This also means that any filter or SQL-Where components connected to such functions, do not affect any other connections.

Priority context
Usually, function parameters are evaluated from top to bottom, but it is possible to define one parameter to be evaluated before all others, using the `priority context` setting.

In functions connected to the bool input of `filter` conditions, the priority context affects not only the comparison function itself but also the evaluation of the filter, so it is possible to join together two source sequences (see CompletePO.mfd, CustomerNo and Number). See Priority Context node/item.

Overriding the context
Some aggregate functions have an optional “parent-context” input. If this input is not connected, it has no effect and the function is evaluated in the normal context for sequence inputs (that is, in the context of the target node’s parent).
If the parent-context input is connected to a source node, the function is evaluated for each parent-context node and will produce a separate result for each occurrence. See also Overriding the Mapping Context.

Bringing multiple nodes of the same source component into the context
This is required in some special cases and can be done with Intermediate variables.

5.16.1 Changing the Processing Order of Mapping Components

MapForce supports mappings that have several target components. Each of the target components has a preview button allowing you to preview the mapping result for that specific component.

If the mapping is executed from the command line or from generated code, then, regardless of the currently active preview, the full mapping is executed and the output for each target component is generated.

The order in which the target components are processed can be directly influenced by changing the position of target components in the mapping window. The position of a component is defined as its top left corner.

Target components are processed according to their Y-X position on screen, from top to bottom and left to right.

- If two components have the same vertical position, then the leftmost takes precedence.
- If two component have the same horizontal position, then the highest takes precedence.
- In the unlikely event that components have the exact same position, then an unique internal component ID is automatically used, which guarantees a well-defined order but
which cannot be changed.

The screenshot below shows the tutorial sample Tut-ExpReport-multi.mfd available in the <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ folder. Both target components (ExpReport-Target) have the same vertical position, and the preview button is active on the right hand target component.

Having selected XSLT2 and generated the code:

- The leftmost target component is processed first and generates the ExpReport.xml file.
- The component to the right of it is processed next and generates the SecondXML.xml file.

You can check that this is the case by opening the DoTransform.bat file (in the output folder you specified) and see the sequence the output files are generated. ExpReport-Target.xml is the first output to be generated by the batch file, and SecondXML.xml the second.
Changing the mapping processing sequence:

1. Click the left target component and move it below the one at right.

2. Regenerate your code and take a look at the DoTransform.bat file.

Chained mappings

The same processing sequence as described above is followed for chained mappings. The chained mapping group is taken as one unit however. Repositioning the intermediate or final target component of a single chained mapping has no effect on the processing sequence.
Only if multiple "chains" or multiple target components exist in a mapping does the position of the final target components of each group determine which is processed first.

- If two final target components have the same vertical position, then the leftmost takes precedence.
- If two final target component have the same horizontal position, then the highest takes precedence.
- In the unlikely event that components have the exact same position, then an unique internal component ID is automatically used, which guarantees a well-defined order but which cannot be changed.

5.16.2 Priority Context node/item

When applying a function to different items in a schema or database, MapForce needs to know what the context node will be. All other items are then processed relative to this one. This is achieved by designating the item (or node) as the priority context.

Priority-context is used to prioritize execution when mapping unrelated items.

Mappings are always executed top-down; if you loop/search through two tables then each loop is processed consecutively. When mapping unrelated elements, without setting the priority context, MapForce does not know which loop needs to be executed first, it therefore automatically selects the first table, or data source.

Solution:
Decide which table, or source data is to be looped/searched first, and then set the priority context on the connector to that table.

A simplified version of the complete DB_CompletePO.mfd file available in the MapForceExamples folder, is shown below.

Note that there are multiple source components in this example. ShortPO is a Schema with an associated XML instance file, while CustomersAndArticles is a database. The data from both are then mapped to the CompletePO schema / XML file. The priority context icon is enclosed in a circle as a visual indication.

Designating the a parameter of the equal function as the priority context would cause:

- The CustomerNr in ShortPO is compared with the item Number in the database.
- CustomerNr has been designated as the priority context, and is placed in the a parameter of the equal function.
- The CustomersAndArticles database is then searched (once) for the same number. The b parameter contains the Number item from the database.
- If the number is found, then the result is passed to the bool parameter of the filter component (Customers).
- The node/row parameter passes on the Customer data to "on-true" when the bool parameter is true, i.e. when the same number has been found.
- The rest of the customer data is then passed on as: Number, FirstName, LastName items, are all connected to the corresponding items in the target schema.
This means that the database is only searched once per CustomerNr supplied by ShortPO.

Designating the \( b \) parameter of the equal function as the priority context would cause:

- MapForce to search and load the first Number into the \( b \) parameter from the database
- Check against the CustomerNr in the \( a \) parameter of ShortPO
- If not equal, search through all CustomerNr of ShortPO
- Search the database and load the next Number into \( b \), check against \( a \), and
- Iterate through every Number in the database while trying to find that number in ShortPO.

This means that a database query is generated for each Number and the result is then compared
to every CustomerNr of ShortPO.

**Priority context and user-defined functions**

If a user-defined function has been defined of type "inline", the default setting, then a priority context cannot be defined on one of the parameters of the user-defined function. The user-defined function can, of course, contain other regular (non-inlined) user-defined functions which have priority contexts set on their parameters.

### 5.16.3 Overriding the Mapping Context

In some mappings, in order to achieve the desired mapping output, it may be necessary to override the mapping context. For this reason, some components provide an optional parent-context item in their structure which enables you to influence the mapping context if so required. Examples of such components are: aggregate functions, variables, and Join components.

![An aggregate function with optional parent-context](image)

To understand why the mapping context is important, let's add to the mapping an XML file that contains nested nodes with multiple levels. On the **Insert** menu, click **XML Schema/File**, and browse for the file: `<Documents>\Altova\MapForce2019\MapForceExamples\Altova_Hierarchical.xml`. 
Importantly, in the XML file above, the **Office** parent node contains multiple **Department** nodes, and each **Department** contains multiple **Person** nodes. If you open the actual XML file in an XML editor, you can see that the distribution of people by office and department is as follows:

<table>
<thead>
<tr>
<th>Office</th>
<th>Department</th>
<th>Number of people</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nanonull, Inc.</td>
<td>Administration</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Marketing</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Engineering</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>IT &amp; Technical Support</td>
<td>4</td>
</tr>
<tr>
<td>Nanonull Partners, Inc.</td>
<td>Administration</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Marketing</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>IT &amp; Technical Support</td>
<td>3</td>
</tr>
</tbody>
</table>
Now let's assume that your mapping should count all people in all departments. To achieve this requirement, you can add the `count` function from `core | aggregate functions` and map data as follows:

If you preview the mapping at this stage, the output is 21, which corresponds to the total number of people in all departments. Notice that the `count` function includes an optional `parent-context` item, which so far has not been connected. As a result, the parent context of the `count` function is the default root node of the source component (which, in this case, is the `Altova` item). This means that all the persons, from all departments, are considered for the scope of the `count` function. This is the way the mapping context works by default, as outlined in Mapping Rules and Strategies, and this is sufficient in most mapping scenarios.

However, it is possible to override the default mapping context if necessary. To do this, add a connection from the `Office` node to the `parent-context` item as shown below.
By changing the mapping as shown above, you are instructing the mapping to iterate over people records in the context of each office. Therefore, if you preview the mapping now, the output will be 15*. This is exactly the number of people in the first office, "Nanonull, Inc.". The explanation is that this time the people nodes were counted twice (once for each office). The count of people in each office was 15 and 6, respectively. However, only the first result was returned (because the function cannot return a sequence of values, only a simple value).

* Assuming that the target language of the mapping is other than XSLT 1.0.

You can further modify the mapping so as to change the mapping context to Department, as shown below. This time the people records would be counted in the context of each department (that is, 7 times, which corresponds to the total number of departments). Again, only the first of the results is returned, so the mapping output is 3, which corresponds to the number of people in the first department of the first office.
While this mapping is not doing much yet, its point is to illustrate how the parent-context item influences the output of the mapping. Having this in mind, you can override the parent-context in other mappings, such as those that contain variables or Join components. See also Example: Grouping and Subgrouping Records.
Chapter 6

Debugging Mappings
6 Debugging Mappings

MapForce includes a mapping debugger available for the MapForce BUILT-IN transformation language. The mapping debugger helps you achieve the following goals:

- View and analyze the values produced by the mapping at each individual connector level.
- Highlight on the mapping the context (set of nodes) responsible for producing a particular value.
- Execute a mapping step-by-step, in order to see how MapForce processes or computes each value in real time, and preview the mapping output as it is being generated.
- Set milestones (breakpoints) at which the mapping execution should stop and display the value(s) currently being processed.
- View the history of values processed by a connector since mapping execution began up until the current execution position.

The mapping debugger is available when the transformation language of the mapping is BUILT-IN. If you start debugging a mapping designed for a different language, you will be prompted to change the mapping language to BUILT-IN. You can also convert a mapping to BUILT-IN by selecting the menu command Output | Built-in Execution Engine. In either case, the conversion to BUILT-IN will be successful if the mapping does not include components that are not available in the BUILT-IN language (for example, XSLT functions).

The MapForce debugger is unlike a traditional debugger in that it does not traverse your program code line by line (since you do not write any code with MapForce). Instead, the debugger exposes the results of MapForce-generated code produced from the mappings you design. More specifically, the debugger logs values that are passed from and to mapping components through their input and output connectors. The logged values are then available for your analysis directly on the mapping or through dedicated windows.

The following sections highlight various ways in which you can use the mapping debugger.

Debug with breakpoints

When you need to stop the debugging execution at a particular place in the mapping, you can set breakpoints, similar to how you would do that in a traditional development environment. The difference is that breakpoints are added not to a line of code, but to an input or output connector of a mapping component. You can also add conditions to breakpoints (this can be useful if you want to stop the execution only if the set condition is satisfied).

You can define breakpoints on the desired connectors and execute the mapping up to the first encountered breakpoint, then go to the next one, and so on. This way you can analyze the
mapping context and values associated with chosen connectors. You can also speed up or slow down the execution by means of the **Step Into**, **Step Out**, **Step Over**, and **Minimal Step** commands provided by the debugger. These commands enable you to skip portions of the mapping, or, on the contrary, execute portions of the mapping in a more granular way if necessary.

**Debug step-by-step**
You can debug a mapping step-by-step, and analyze the mapping context and values associated with each step. This scenario is similar to the previous one, in that you can speed up or slow down execution using the **Step Into**, **Step Out**, **Step Over**, and **Minimal Step** commands.

**Analyze the log of values**
You can configure MapForce to remember the log of all values (trace history) that were processed by all connectors while you debug a mapping. Keeping the full trace history may not be suitable for mappings that are data-intensive, so this option can be disabled if necessary. When the option is enabled, you can analyze the full log of values processed by each connector up until the current execution position. You can also instruct MapForce to recreate the mapping context associated with any particular value, which would help you understand why that value was produced.

**Set the context to a value related to the current execution position**
When the debugger is at a particular execution position on the mapping, it is possible to analyze the context of a past value relative to the current execution position (this can be compared to stepping slightly back in time):
A context is meant to explain why a value is computed; in other words, it describes how a particular value on the mapping came to be generated. The context is normally the current execution position, although it can also be a context in the recent past that MapForce enables you to set. When the context is set to a particular value, MapForce highlights directly on the mapping the nodes that are relevant to it, provides tips next to mapping connectors, and exposes additional information in debugger-related windows (the Values, Context, and Breakpoints windows).

After you have inspected a mapping context that is not the same as the current execution position, you can reset the context back to the current execution position:

Limitations

- When MapForce executes a mapping, it may internally optimize code (for example, by caching data, or by calculating intermediate results at arbitrary points). This may cause certain connectors (and thus breakpoints) to be unreachable for debugging, in which case MapForce displays a notification. Note that the MapForce code optimizations (and, consequently, the behavior exposed by the debugger) may be different from one MapForce release to the other, even though the mapping output is the same for a given mapping.
- The debugger can debug the output generation for one target component at a time. If there are multiple target components on the mapping, you will need to select which one should be executed by the debugger.
- Currently, debugging is not supported for the database table actions (such as "Insert All", "Update If", etc.) of database components.
- Breakpoints cannot be added on any of the following entities: constants, the core position function, descendent items of "Copy-all" connections, parameters of "inline" user-defined functions.
6.1 Debugging Preparation

Debugging preparation is primarily required for big data mappings that are likely to need a lot of system memory to execute. This is the case of mappings that either process very big input or output files, or repeatedly iterate through large collections of data.

To make debugging faster and reduce memory requirements, it is recommended to do the following before you start debugging:

- If the mapping is complex, remove or disconnect parts of the mapping that need not be debugged.
- If the mapping uses big input files, replace them with files of smaller size.
- Ensure that the Keep full trace history option is disabled (see Debugger Settings).

Also, to ensure you are debugging the right output, check the following if applicable:

- If the mapping has multiple target components, select the target component to be debugged by clicking the Preview button.
- If the mapping is a chained mapping (see Chained Mappings), release the Pass-Through button on the intermediary component. Debugging Pass-Through components is currently not supported.

Optionally, if you want the debugger to stop at some important connectors whose value you want to analyze, add breakpoints to these connectors (see Adding and Removing Breakpoints).
## 6.2 Debugger Commands

You can access the debugger commands as follows:

- In the **Debug** menu
- As keyboard shortcuts
- In the Debug toolbar.

<table>
<thead>
<tr>
<th>Menu Command</th>
<th>Keyboard Shortcut</th>
<th>Toolbar button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>Start debugging</td>
<td>F5</td>
<td></td>
</tr>
<tr>
<td>Debug</td>
<td>Stop debugging</td>
<td>Shift + F5</td>
<td></td>
</tr>
<tr>
<td>Debug</td>
<td>Step Into</td>
<td>F11</td>
<td></td>
</tr>
<tr>
<td>Debug</td>
<td>Step Over</td>
<td>F10</td>
<td></td>
</tr>
<tr>
<td>Debug</td>
<td>Step Out</td>
<td>Shift + F11</td>
<td></td>
</tr>
<tr>
<td>Debug</td>
<td>Minimal Step</td>
<td>Ctrl + F11</td>
<td></td>
</tr>
</tbody>
</table>
### Menu Command | Keyboard Shortcut | Toolbar button | Description
---|---|---|---
| | | | necessarily compute values in the order the mapping would suggest, so production and consumption events do not always follow each other.

*Debug toolbar*
6.3 About the Debug Mode

When you start debugging (by pressing F5, or F11, or Ctrl + F11), MapForce executes the mapping in debug mode.

While MapForce is in debug mode, the mapping is read-only. Although you can move components on the mapping area, most commands are not available. This includes commands such as mapping validation and deployment, code generation, documenting mappings, adding new components to the mapping area or reloading existing ones, and others.

The debug mode enables you to analyze the context responsible for producing a particular value. This information is available directly on the mapping, as well as in the Values, Context, and Breakpoints windows. By default, these windows are displayed when you start debugging and are hidden when you stop debugging.

MapForce is in debug mode (and the mapping is read-only) until you stop debugging, by pressing Shift + F5 (or by clicking the Stop debugging toolbar button).

The following image illustrates a sample mapping (SimpleTotal.mfd, from the <Documents>\Altova\MapForce2019\MapForceExamples directory) that is debugged in steps (by pressing F11 to advance a step).
The mapping pane
While debugging, the mapping pane displays additional information:

- Data overlays (see below) show the current value and related values near their connectors.
- The current context (shown as a structure in the Context window) is highlighted as follows:
  - Connectors in the context are striped magenta ( \[\textcolor{magenta}{\text{\textbullet\text{-}}\text{\textbullet}}\)).
  - Connectors in ambiguous context are dotted magenta ( \[\textcolor{magenta}{\text{\textbullet\text{-}}\text{\textbullet}}\)).
  - Connections in the context are striped magenta.
  - Connections in ambiguous context are striped magenta but lighter.
- The current execution location is displayed with a green connector icon ( \[\textcolor{green}{\text{\textbullet\text{-}}\text{\textbullet}}\).

Data overlays
The values processed by each connector are displayed as data overlays (small rectangles) near their corresponding connector. A currently selected data overlay is displayed with thick red border. Values changed from the last step are displayed in dark red. For nodes with simple content, the data overlay combines two values - the node name and the value. If the node name has been iterated multiple times before the current execution position, the index of the current iteration is indicated by the number in square brackets.

Data overlays have the following behavior:

- Pointing the mouse to a data overlay brings it temporarily to the foreground, clicking it does it permanently. Clicking also selects the corresponding connector.
- Data overlays can be moved by dragging.
- Data overlays move when a component is moved. Therefore, if the data overlays appear stacked because the components are too close to each other, drag the components around the mapping area to make more space, and the data overlays will move together with the component.
- Clicking a data overlay shows its value in the Values window.
- Clicking a connector also selects its data overlay.

Breakpoints
Breakpoints are designated milestones at which the mapping should break during execution in debug mode. This term may be already familiar to you by analogy with other integrated development environments. Unlike other development environments where you add breakpoints to a line of code, a breakpoint in MapForce can be added to an input or output connector (small triangle to the left or right of the connection). On the mapping pane, breakpoints are represented as red circles. Any defined breakpoints are also displayed in the Breakpoints window. See also Adding and Removing Breakpoints.

Current debugger position
The green triangle ( \[\textcolor{green}{\text{\textbullet\text{-}}\text{\textbullet}}\) indicates the position of the debugger. This position is either an input or an output connector of any given component.

The value currently being processed is also displayed in the Values window, on the Context tab.
The set of connections and/or connectors colored in striped magenta indicate the current mapping context. The same information is also displayed as a hierarchical structure in the Context window (see Using the Context Window).

When you set manually the context of a value, the current debugger position is in a position in the past relative to the most current execution position. To help you distinguish between the most current execution position and the one in the past, the "current position" connector may appear with the following colors in the debugger interface.

- **Green** is "the present"; it indicates the current execution position (see Viewing the Current Value of a Connector).
- **Yellow** is "the past"; it indicates that you are looking at some connector in the past, relative to the current execution position. This may happen after you set a context manually (see Setting the Context to a Value).

### Values window

The Values window provides information about the values processed by the mapping. It enables you to see what the mapping processes at the current execution position, or in a particular context that you can set yourself. See also Using the Values Window.

### Context window

The Context window provides a hierarchical view of the set of nodes and functions that are relevant for the current debugger position. See also Using the Context Window.

### Breakpoints window

The Breakpoints window displays the list of debugging breakpoints created since MapForce was started. If you have defined breakpoints on multiple mappings, all of them appear in the Breakpoints window. See also Using the Breakpoints Window.
6.4 Adding and Removing Breakpoints

Breakpoints are designated milestones at which the mapping should break during execution in debug mode. Any breakpoints you create are stored globally for all mappings and are displayed in the Breakpoints window. Breakpoints are valid until you either explicitly delete them, or close MapForce.

**Note:** Breakpoints cannot be added on any of the following entities: constants, the `core | position` function, descendent items of "Copy-all" connections, parameters of "inline" user-defined functions.

Breakpoints can be simple or conditional. Simple breakpoints stop the mapping execution unconditionally. Conditional breakpoints stop the mapping execution only when the condition assigned to them is satisfied. Conditions take the form of MapForce built-in library functions to which you supply custom values. In other words, if the condition returns true, the breakpoint will stop the mapping execution.

**To create a simple breakpoint, do one of the following:**

- Right-click an input or output connector (the small triangles to the left or right of a component), and select **Debugger Breakpoint**.
- Click an input or output connector, and then press **F9**.

**To create a conditional breakpoint:**

1. Right-click a connector, and select **Breakpoint properties**.
2. Click to select both the Breakpoint and Condition check boxes.
3. Select the required function from the list, and enter the function value (if applicable). For example, in the example above, the breakpoint will stop the mapping execution if the value passing through it is greater than 2.

If the data type of the connector where you add the conditional breakpoint does not match the type(s) expected by the function, MapForce will attempt to convert the data type automatically. If automatic conversion is not possible, mapping execution will fail. To avoid this, make sure to use compatible data types. For example, the function `core.starts-with` expects a string value, so the breakpoint's connector must have the same type.

### Removing breakpoints

To remove a breakpoint, right-click the connector on which the breakpoint exists, and select Debugger Breakpoint. Alternatively, click the input or output connector on which the breakpoint exists, and then press F9.

You can also remove breakpoints from the Breakpoints window (see Using the Breakpoints Window).

### Unreachable breakpoints

There may be cases when MapForce displays a "Breakpoints cannot be reached" message:

![MapForce dialog showing unreachable breakpoints]

This indicates that breakpoints cannot be reached by the debugger, because of one of the following reasons:

- A breakpoint has been defined on a connector that does not take part in the mapping.
- The breakpoint cannot be reached by MapForce because of execution optimizations (see Limitations).

Click Continue to advance to the next defined breakpoint (or go to the end of debugging execution). Click Step to start debugging in steps.

You can disable notifications about unreachable breakpoint encountered by the debugger, either by clicking Don't show this message again, or as follows:

1. On the Tools menu, click Options.
2. Click Messages.
3. Click to clear the Inform about unreachable breakpoints check box.
Using the Values Window

The Values window displays information about the values processed by the mapping when in debug mode. The information displayed in the Values window depends on the current debugger position, and on the user interface elements that you clicked. The Values window contains the following tabs:

The "Context" tab
The Context tab displays the value currently being processed (the same value whose context is shown in the Context window). This is either the value at the current execution position of the debugger, or the value of a connector processed in the past. MapForce helps you distinguish between the two using colors:

<table>
<thead>
<tr>
<th>Color</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Green</td>
<td>&quot;the present&quot;; it indicates the current execution position (see Viewing the Current Value of a Connector).</td>
</tr>
<tr>
<td>Yellow</td>
<td>&quot;the past&quot;; it indicates that you are looking at some connector in the past, relative to the current execution position. This may happen after you set a context manually (see Setting the Context to a Value).</td>
</tr>
</tbody>
</table>

The "Related" tab
The Related tab displays values that are related to (or represent the "near past" of) the currently processed value. Normally, you do need to explicitly click this tab; MapForce switches to it automatically when you click the data overlay of a connector that is related to the current execution position of the debugger. See Stepping back into Recent Past.

The "Sequence" tab
When present, the Sequence tab enables you to get access to the values of a connector that processes a sequence. This tab is visible only when a connector has processed a sequence of items (for example, an aggregate function such as sum or count does that). When you click the data overlay of a connector that processed a sequence of items, the Values window displays an entry in the format "n items", where n is the number of items processed by the connector. To get access to each value, double-click this entry (or right-click it, and select Expand Sequence from the context menu).

The values are then displayed in the Sequence tab.
The "History" tab

The History tab displays values have been processed by a particular node since debugging started and up to the current execution position. See Viewing the History of Values Processed by a Connector.
## 6.6 Using the Context Window

While MapForce is in debug mode, the Context window displays a structure of connectors that are relevant to the current position of the debugger. In other words, it provides the mapping context responsible for producing the current mapping value.

MapForce builds the current context as follows:

1. Start with the root node of the target structure.
2. Descend to the current target node.
3. From the current target node, move left inside the mapping through any components that lead to the current position. These components may be filter or sort components, built-in or user-defined functions, variables, and so on.

The Context window serves both as informational and as a navigational aid. To select a particular node in the mapping directly from the current context, right-click the node in the Context window, and click **Select in mapping**. This might be especially useful when the mapping is large, so as to avoid extensive scrolling.

The Context window may display the following special icons and notation:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Icon]</td>
<td>Represents the mapping to which the context belongs. This can be either the main mapping or the mapping of a user-defined function.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Represents a connector. The target nodes processed so far have their position displayed in square brackets.</td>
</tr>
</tbody>
</table>
## Icons and Description

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image.png" alt="Icon" /></td>
<td>Represents the current connector (the most recent execution position). This is the source of the current value in the Values window. In some rare situations, it is possible that a computed value is used for multiple connectors. In this case, multiple green icons may appear.</td>
</tr>
<tr>
<td><img src="image.png" alt="Icon" /></td>
<td>Represents the current connector when the debugger is at some position in the past relative to the most recent execution post. This may happen after you set the context to a value (see <a href="#">Setting the Context to a Value</a>).</td>
</tr>
</tbody>
</table>

In addition to the icons above, the Context window includes the standard icons of any component types that are present in the mapping.

### Context window and user-defined functions

If the current context includes any user-defined functions, they are displayed in the Context window as well. Note that if the current context is for computing an input value of a user-defined function, the context is determined as follows:

1. From the target to the output connector of the user-defined function to the input connector of the user-defined function
2. From there further to the left.

**Note:** A user-defined function may occur multiple times in the context. This happens either because several function calls are chained or because the user-defined function is defined as recursive.
6.7 Using the Breakpoints Window

The Breakpoints window enables you to view and manage breakpoints globally. By default, the Breakpoints window is displayed when MapForce is in debug mode. To make the Breakpoints window visible at all times, select the menu command **View | Debug Windows | Breakpoints**.

The Breakpoints window displays all breakpoints created since you started MapForce, grouped by the mapping file to which they belong. While MapForce is open, any breakpoints associated with any mapping are "remembered" by MapForce and displayed in the Breakpoints window, even if you closed the mapping file in the meanwhile. The mapping that is currently being debugged is represented with standard text color in the Breakpoints window, while other mappings (the ones that are closed or not active) are grayed out.

You can quickly open any mapping by double-clicking it (or any of its breakpoints) in the Breakpoints window.

**Note:** Once you close or restart MapForce, all breakpoints are removed.

Information about breakpoints is displayed as a grid with the following columns:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the node where the breakpoint belongs.</td>
</tr>
<tr>
<td>Parent</td>
<td>The name of the mapping component where the breakpoint belongs.</td>
</tr>
<tr>
<td>Trace value</td>
<td>The value that passes through the connector on which the breakpoint is. The trace value is displayed during debugging execution.</td>
</tr>
<tr>
<td>Condition</td>
<td>If the breakpoint is conditional, this column displays the condition of the breakpoint.</td>
</tr>
</tbody>
</table>

Breakpoints may be associated with any of the following icons.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Active breakpoint" /></td>
<td>Active breakpoint. Denotes a breakpoint from the mapping that is currently being debugged.</td>
</tr>
<tr>
<td>Icon</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>![Circle]</td>
<td>Inactive breakpoint. Denotes a breakpoint from a mapping that is open, but is not currently being debugged.</td>
</tr>
<tr>
<td>![Exclamation Mark]</td>
<td>Inaccessible breakpoint. Denotes a breakpoint that cannot be reached by the debugger.</td>
</tr>
<tr>
<td>![Plus Sign]</td>
<td>Conditional breakpoint. Denotes a breakpoint with a condition attached to it.</td>
</tr>
</tbody>
</table>

**To view or change the properties of a breakpoint:**

- Right-click it, and select **Breakpoint Properties** from the context menu.

**To delete a breakpoint:**

- Right-click the breakpoint you want to delete, and then select **Delete Breakpoint** from the context menu.
- Click a breakpoint, and then press **Delete**.

The context command **Delete All Breakpoints** removes all breakpoints displayed in the Breakpoints window, regardless of the mapping where they belong.

See also: **Adding and Removing Breakpoints**
6.8  Previewing Partially Generated Output

When you are debugging in steps or using breakpoints, you can view the mapping output generated up to the current debugger position. Previewing partially generated output is supported by XML, flat text, and EDI target components.

By default, when you press F5 (without having defined any breakpoints), MapForce executes the entire mapping in debug mode, and then switches to the Output tab, displaying the final generated output. However, if you have defined breakpoints, or if you are debugging in steps (F11, or Ctrl + F11), the debugger execution stops while the mapping output is still being generated. Even if the mapping output is partially written at this stage, you can still click to the Output tab, and preview it.

Limitations

- The currently computed target node is not always displayed in the Output tab. For example, XML attributes are collected internally and written at once.
- If the output produces multiple files, only the currently written file can be displayed; switching to another output file is disabled.
6.9 Viewing the Current Value of a Connector

When the current execution position of the debugger (▶) is on a particular connector (either because you are debugging in steps, or because there is a breakpoint defined on the connector), the current value processed by the connector is displayed in the Context tab of the Values window. This is the value that is about to be written to the output, that is, “the present”. It is also the value whose context is displayed in the Context window (see Using the Context Window).

To understand this case, open the PreserveFormatting.mfd sample from the <Documents>\Altova\MapForce2019\MapForceExamples directory. Click the input connector of the Number node on the target component, and press F9 to add a breakpoint on it.

Then press F5 to start debugging and observe the results.

As shown in the image, the current debugger position (▶) (and the breakpoint (●)) is on the Number node of the target component. The Values window indicates that this node processes the value “1” (this value is also highlighted with a thick red border on the mapping).
6.10 Stepping back into Recent Past

When you click a data overlay (small rectangular box) next to a mapping connector, the Values window displays the name and, optionally, the value associated with the selected connector. The focus now is no longer on the current debugger position, but on the selected data overlay. You can consider this view as stepping slightly back in the debugging history. This is the "near" past, since the mapping displays data overlays only for the last few connectors related to the current debugger position. When you click such a "related" data overlay, the Values window switches automatically to the Related tab.

For an illustration of this scenario, open the mapping PreserveFormatting.mfd from the <Documents>\Altova\MapForce2019\MapForceExamples directory.

After opening the mapping, click the connector next to the Number node on the target component, and press F9 to add a breakpoint on it. Press F5 to start debugging, and then click the data overlay (small rectangular box) next to the Number node of the source component.

Because a connector is typically iterated multiple times for the lifetime of a mapping, the current index of the iteration is displayed enclosed with square brackets: <Number>[1]. Also, because the connector carries a value, its value is also represented after the equal sign: <Number>[1]=1. The same value is displayed on a new row in the Values window, as shown below.

If you need additional information about a particular value, remember that you can recreate the context that produced it (see Setting the Context to a Value).
6.11 Viewing the History of Values Processed by a Connector

If the option Keep full trace history is enabled (see Debugger Settings), you can view the history of all values that were processed by that connector (up to the current execution position).

The history is displayed when you click a connector, and then click the History tab of the Values window. Note that this operation is meaningful only for connectors that have processed values since the beginning of mapping execution until the current debugger position.

To illustrate this case, let's debug a mapping from begging till end without using any breakpoints, and then watch the history of all values that were processed by a particular connector. First, open the mapping PreserveFormatting.mfd from the <Documents>\Altova\MapForce2019\MapForceExamples\ directory. If it is already open, make sure to do the following:

- Clear any breakpoints, if such exist (see Adding and Removing Breakpoints)
- Stop debugging if it is currently in progress, by pressing Shift + F5.

When ready, press F5 start a new debugging operation. When you press F5, MapForce executes the mapping in debug mode, and switches to the Output tab. Click the Mapping tab to go back to the main mapping window, and then click the result node of the format-number function (highlighted in red in the image below). Finally, click the History tab of the Values window, and notice the displayed values.

As shown in the image above, this particular node (result) has processed four values in total. If you need additional information about a particular value, remember that you can recreate the context that produced it (see Setting the Context to a Value).
6.12 Setting the Context to a Value

Setting the context to a value is an action that can be compared to stepping into the past, in order to view more details about the mapping context that produced that value. You can set the context to any value displayed in the Values window (in the Related tab, Sequence tab, or History tab).

If you have enabled the Keep full trace history option (see Debugger Settings), the History tab displays all values processed by the currently selected connector; therefore, in this case, you can additionally set the context to any value in the past for that connector.

To set the context to a value, do one of the following:

- Right-click the value, and select Set Context from the context menu.
- Double-click the value.

When you set the context to a value, MapForce highlights the mapping area so as to recreate the situation that produced that value, and populates the Values window and the Context window according to the selected context. For a legend to visual clues used on the mapping area while in a context, see About the Debug Mode. For information about the context itself, see Using the Context Window.

The connector of a manually-set context is yellow ( ), which indicates that you are no longer at the most recent execution position. To switch back to the most recent execution position (when applicable), click the Reset to Current button on the Context tab of the Values window.
### 6.13 Debugger Settings

To access the settings applicable to the MapForce debugger, select the menu command **Tools | Options**, and then click **Debugger**. The available settings are as follows:

#### Maximum storage length of values
Defines the string length of values displayed in the Values window (at least 15 characters). Note that setting the storage length to a high value may deplete available system memory.

#### Keep full trace history
Instructs MapForce to keep the history of all values processed by all connectors of all components in the mapping for the duration of debugging. If this option is enabled, all values processed by MapForce since the beginning of debug execution will be stored in memory and available for your analysis in the Values window, until you stop debugging. It is not recommended to enable this option if you are debugging data-intensive mappings, since it may slow down debugging execution and deplete available system memory. If this option is disabled, MapForce keeps only the most recent trace history for nodes related to the current execution position.
Chapter 7

Data Sources and Targets
7 Data Sources and Targets

This section provides information specific to various source and target component types that MapForce can map from or to:

- XML and XML Schema
- Databases
- CSV and Text Files
- MapForce FlexText
- EDI (this includes EDI-based or EDI-related formats, such as ASC X12, HIPAA X12, HL7 Version 2, IATA PADIS, NCPDP SCRIPT, SAP IDoc, TRADACOMS, UN/EDIFACT)
- JSON
- Microsoft OOXML Excel 2007+
- XBRL
- HL7 Version 3
7.1 XML and XML schema

Altova website: XML mapping

In the introductory part of this documentation, you have seen examples of simple mappings that use XML and XML schema files as source or target components. This section provides further information about using XML components in your mappings. It includes the following topics:

- Generating an XML Schema
- XML Component Settings
- Using DTDs as "Schema" Components
- Derived XML Schema Types
- QNames
- Nil Values / Nillable
- Comments and Processing Instructions
- CData Sections
- Wildcards - xs:any / xs:anyAttribute
- Merging Data from Multiple Schemas
- Declaring Custom Namespaces

7.1.1 Generating an XML Schema

MapForce can automatically generate an XML schema based on an existing XML file if the XML Schema is not available. Whenever you add to the mapping area an XML file without a schema (using the menu command Insert | XML Schema/File), the following dialog box appears.

Click Yes to generate the schema, you will then be prompted to select the directory where the generated schema should be saved.

When MapForce generates a schema from an XML file, data types for elements/attributes must be inferred from the XML instance document and may not be exactly what you expect. It is recommended that you check whether the generated schema is an accurate representation of the instance data.

If elements or attributes in more than one namespace are present, MapForce generates a separate XML Schema for each distinct namespace; therefore, multiple files may be created on the disk.
7.1.2 XML Component Settings

After you add an XML component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component settings dialog box in one of the following ways:

- Select the component on the mapping, and, on the Component menu, click Properties.
- Double-click the component header.
- Right-click the component header, and then click Properties.
The available settings are as follows.

<table>
<thead>
<tr>
<th><strong>Component name</strong></th>
<th>The component name is automatically generated when you</th>
</tr>
</thead>
</table>

**XML Component Settings dialog box**
create the component. However, you can change the name at any time.

The component name can contain spaces and full stop characters. It may not contain slashes, backslashes, colons, double quotes, leading or trailing spaces. If you want to change the name of the component, be aware of the following:

- If you intend to deploy the mapping to FlowForce Server, the component name must be unique.
- It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line.

### Schema file

Specifies the name or path of the XML schema file used by MapForce to validate and map data.

To change the schema file, click **Browse** and select the new file. To edit the file in XMLSpy, click **Edit**.

### Input XML file

Specifies the XML instance file from which MapForce will read data. This field is meaningful for a source component and is filled when you first create the component and assign to it an XML instance file.

In a source component, the instance file name is also used to detect the XML root element and the referenced schema, and to validate against the selected schema.

To change the location of the file, click **Browse** and select the new file. To edit the file in XMLSpy, click **Edit**.

### Output XML file

Specifies the XML instance file to which MapForce will write data. This field is meaningful for a target component.

To change the location of the file, click **Browse** and select the new file. To edit the file in XMLSpy, click **Edit**.

### Prefix for target namespace

Allows you to enter a prefix for the target namespace. Ensure that the target namespace is defined in the target schema, before assigning the prefix.

### Add schema/DTD reference

Adds the path of the referenced XML Schema file to the root element of the XML output. The path of the schema entered in this field is written into the generated target instance files in the `xsi:schemaLocation` attribute, or into the `DOCTYPE` declaration if a DTD is used.

Note that, if you generate code in XQuery or C++, adding the DTD reference is not supported.

Entering a path in this field allows you to define where the
schema file referenced by the XML instance file is to be located. This ensures that the output instance can be validated at the mapping destination when the mapping is executed. You can enter an http:// address as well as an absolute or relative path in this field.

Deactivating this option allows you to decouple the XML instance from the referenced XML Schema or DTD (for example, if you want to send the resulting XML output to someone who does not have access to the underlying XML Schema).

**Write XML declaration**

This option enables you to suppress the XML declaration from the generated output. By default, the option is enabled, meaning that the XML declaration is written to the output.

This feature is supported as follows in MapForce target languages and execution engines.

<table>
<thead>
<tr>
<th>Target language / Execution engine</th>
<th>When output is a file</th>
<th>When output is a string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Built-in</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MapForce Server</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>XSLT, XQuery</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Code generator (C++, C#, Java)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Cast values to target types**

Allows you to define if the target XML schema types should be used when mapping, or if all data mapped to the target component should be treated as string values. By default, this setting is enabled.

Deactivating this option allows you to retain the precise formatting of values. For example, this is useful to satisfy a pattern facet in a schema that requires a specific number of decimal digits in a numeric value.

You can use mapping functions to format the number as a string in the required format, and then map this string to the target.

Note that disabling this option will also disable the detection of invalid values, e.g. writing letters into numeric fields.

**Pretty print output**

Reformats the output XML document to give it a structured look. Each child node is offset from its parent by a single tab character.
### Create digital signature

Allows you to add a digital signature to the XML output instance file. Adding a digital signature is possible when you select "Built-in" as transformation language (see also Digital Signatures).

### Output Encoding

Allows you to specify the following settings of the output instance file:

- Encoding name
- Byte order
- Whether the byte order mark (BOM) character should be included.

By default, any new components have the encoding defined in the Default encoding for new components option. You can access this option from Tools | Options, General tab.

If the mapping generates XSLT 1.0/2.0, activating the Byte Order Mark check box does not have any effect, as these languages do not support Byte Order Marks.

### StyleVision Power Stylesheet file

This option allows you to select or create an Altova StyleVision stylesheet file. Such files enable you to output data from the XML instance file to a variety of formats suitable for reporting, such as HTML, RTF, and others.

See also Using Relative Paths on a Component.

### Enable input processing optimizations based on min/ maxOccurs

This option allows special handling for sequences that are known to contain exactly one item, such as required attributes or child elements with minOccurs and maxOccurs="1". In this case, the first item of the sequence is extracted, then the item is directly processed as an atomic value (and not as a sequence).

If the input data is not valid against the schema, an empty sequence might be encountered in a mapping, which stops the mapping with an error message. To allow the processing of such invalid input, disable this check box.

### Save all file paths relative to MFD file

When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. See also Using Relative Paths on a Component.

---

### 7.1.3 Using DTDs as "Schema" Components

Starting with MapForce 2006 SP2, namespace-aware DTDs are supported for source and target components. The namespace-URIs are extracted from the DTD "xmlns"-attribute declarations, to make mappings possible.

However, some DTDs contain xmlns*-attribute declarations without namespace-URIs (for
example, DTDs used by StyleVision). Such DTDs have to be extended to make them useable in MapForce. Specifically, you can make such DTDs useable by defining the xmlns-attribute with the namespace-URI, as shown below:

```xml
<!ATTLIST fo:root
    xmlns:fo CDATA #FIXED 'http://www.w3.org/1999/XSL/Format'
    ...>
```

### 7.1.4 Derived XML Schema Types

MapForce supports the mapping to/from derived types of a complex type. Derived types are complex types of an XML Schema that use the `xsi:type` attribute to identify the specific derived types.

The screenshot below shows the definition of a derived type called `US-Address`, in XMLSpy. The base type (or originating complex type) is `AddressType`. Two extra elements were added to create the derived type `US-Address`: `Zip` and `State`.

The following example shows you how to map data to or from derived XML schema types.

1. On the **Insert** menu, click **XML Schema/File**, and open the following XML Schema: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\MFCompany.xsd`.
2. When prompted to supply an instance file, click **Skip**, and then select **Company** as the root element.
3. Click the **TYPE** button next to the **Address** element. This button indicates that derived types exist for this element in the schema.

![Derived Types (xsi:type)](image)

4. Select the check box next to the derived type you want to use (**US-Address**, in this case), and confirm with OK. A new element **Address xsi:type="US-Address"** has been added to the component.

![Company Element](image)

You can now map data to or from the **US-Address** derived type.

Note that you can also include multiple derived types by selecting them in the Derived Types dialog box. In this case, each would have its own **xsi:type** element in the component.
7.1.5 **QNames**

MapForce resolves QName (qualified name) prefixes ([https://www.w3.org/TR/xml-names/#ns-qualnames](https://www.w3.org/TR/xml-names/#ns-qualnames)) when reading data from XML files at mapping execution run-time.

QNames are used to reference and abbreviate namespace URIs in XML and XBRL instance documents. There are two types of QNames: Prefixed and Unprefixed QNames.

- **PrefixedName** : Prefix ':': LocalPart
- **UnPrefixedName** : LocalPart

where LocalPart is an Element or Attribute name.

For example, in the listing below, `<x:p/>` is a QName, where:

- the prefix "x" is an abbreviation of the namespace "http://myCompany.com".
- `p` is the local part.

```xml
<?xml version='1.0'?>
<doc xmlns:x='http://myCompany.com'>
  <x:p/>
</doc>
```

MapForce also includes several QName-related functions in the [core | QName functions](https://www.altova.com/products/mapforce/library-features/alternative-features/core-qname-functions).

7.1.6 **Nil Values / Nillable**

The XML Schema specification allows for an element to be valid without content if the `nillable="true"` attribute has been defined for that specific element in the schema. In the instance XML document, you can then indicate that the value of an element is nil by adding the `xsi:nil="true"` attribute to it. This section describes how MapForce handles nil elements in source and target components.

`'xsi:nil' versus 'nillable'`

The `xsi:nil="true"` attribute is defined in the XML instance document.
The xsi:nil="true" attribute indicates that, although the element exists, it has no content. Note that the xsi:nil="true" attribute applies to element values, and not to attribute values. An element with xsi:nil="true" may still have other attributes, even if it does not have content.

The xsi:nil attribute is not displayed explicitly in the MapForce graphical mapping, because it is handled automatically in most cases. Specifically, a "nilled" node (one that has the xsi:nil="true" attribute) exists, but its content does not exist.

The nillable="true" attribute is defined in the XML schema. In MapForce, it can be present in both the source and target components.

**Nillable elements as mapping source**
MapForce checks the xsi:nil attribute automatically, whenever a mapping reads data from nilled XML or XBRL elements. If the value of xsi:nil is true, the content will be treated as nonexistent.

When you create a Target-driven mapping from a nillable source element to a nillable target element with simple content (a single value with optional attributes, but without child elements), where xsi:nil is set on a source element, MapForce adds the xsi:nil attribute to the target element (for example, <OrderID xsi:nil="true"/>).
When you create a **Copy-All** mapping from a nillable source element to a nillable target element, where xsi:nil is set on a source element, MapForce adds the xsi:nil attribute to the target element (for example, `<OrderID xsi:nil="true"/>`).

To check explicitly whether a source element has the xsi:nil attribute set to `true`, use the **is-xsi-nil** function. It returns TRUE for nilled elements and FALSE for other nodes.

To substitute a nilled (non-existing) source element value with something specific, use the **substitute-missing** function.

### Notes:
- Connecting the `exists` function to a nilled source element returns TRUE, since the element node actually exists, even if it has no content.
- Using functions that expect simple values (such as `multiply` and `concat`) on elements where xsi:nil has been set does not yield a result, as no element content is present and no value can be extracted. These functions behave as if the source node did not exist.

**Nillable elements as mapping target**

When you create a **Target-driven** mapping from a nillable source element to a nillable target element with **simple content** (a single value with optional additional attributes, but without child elements), where xsi:nil is set on a source element, MapForce inserts the xsi:nil attribute into the target element (for example, `<OrderID xsi:nil="true"/>`). If the xsi:nil="true" attribute has not been set in the XML source element, then the element content is mapped to the target element in the usual fashion.

When mapping to a nillable target element with **complex type** (with child elements), the xsi:nil attribute will not be written automatically, because MapForce cannot know at the time of writing the element's attributes if any child elements will follow. For such cases, define a **Copy-All** connection to copy the xsi:nil attribute from the source element.

When mapping an **empty sequence** to a target element, the element will not be created at all, independent of its nillable designation.

To force the creation of an empty target element with xsi:nil set to `true`, connect the `set-xsi-nil` function directly to the target element. This works for target elements with simple and complex types.

If the node has simple type, use the **substitute-missing-with-xsi-nil** function to insert xsi:nil in the target if no value from your mapping source is available. This can happen if the source node does not exist at all, or if a calculation (for example, multiply) involved a nilled source node and therefore yielded no result.

### Note:
- Functions which generate xsi:nil cannot be passed through functions or components which only operate on values (such as the **if-else** function).
Mapping NULL database fields to xsi:nil

If you map a NULL database field to a nillable element of an XML schema, MapForce generates only those target elements which actually contain database data. Elements of NULL database fields are not created in the target component. Connecting the `exists` node function to such a source element results in `false` for the NULL fields.

To force the creation of all elements in the target component, use the `substitute-missing-with-xsi-nil` function from the node functions of the core library.

The screenshot above illustrates how the `substitute-missing-with-xsi-nil` function is used to create target elements for all database fields:

- All missing/NULL database fields contain `<OrderID xsi:nil="true"/>` in the target element.
- Existing data from database fields is mapped directly to the target element e.g. `<OrderID>1</OrderID>`.

To see the NULL fields of a database component, click the `Database Query` button and run a query on the database table(s). Null fields are shown as `[NULL]` in the Results window.

Mapping xsi:nil to NULL database fields

If you map a nilled XML element to a database column, MapForce writes a NULL value to the database. You can also use the `set-null` function if you want to set a database field to NULL unconditionally.
7.1.7 Comments and Processing Instructions

Comments and Processing Instructions can be inserted into target XML components. Processing instructions are used to pass information to applications that further process XML documents. Note that Comments and Processing instructions cannot be defined for nodes that are part of a copy-all mapped group.

To insert a Processing Instruction:

1. Right-click an element in the target component and select Comment/Processing Instruction, then one of the Processing Instruction options from the menu (Before, After).
2. Enter the Processing Instruction (target) name in the dialog and press OK to confirm, e.g. xml-stylesheet. This adds a node of this name to the component tree.
3. You can use, for example, a constant component to supply the value of the Processing Instruction attribute, e.g. href="book.css" type="text/css".

Note: Multiple Processing Instructions can be added before or after any element in the target component.

To insert a comment:

1. Right-click an element in the target component and select Comment/Processing Instruction, then one of the Comment options from the menu (Before, After).
2. Enter the Comment node (<!--comment--> ) to the component tree.
3. Use a constant component to supply the comment text, or connect a source node to the comment node.

Note: Only one comment can be added before and after a single target node. To create multiple
To delete a Comment/Processing Instruction:

- Right-click the respective node, select Comment/Processing Instruction, then select Delete Comment/Processing Instruction from the flyout menu.

### 7.1.8 CDATA Sections

CDATA sections are used to escape blocks of text containing characters which would normally be interpreted as markup. CDATA sections start with "<![CDATA[" and end with the "]>".

Target nodes can now write the input data that they receive as CDATA sections. The target node components can be any of the following:

- XML data
- XML data embedded in database fields
- XML child elements of typed dimensions in an XBRL target

To create a CDATA section:

1. Right-click the target node that you want to define as the CDATA section and select "Write Content as CDATA section".

A prompt appears warning you that the input data should not contain the CDATA section close delimiter "]>", click OK to close the prompt. The [C.. icon shown below the element tag shows that this node is now defined as a CDATA section.

**Note:** CDATA sections can also be defined on duplicate nodes, and xsi:type nodes.
Example

The HTMLinCDATA.mfd mapping file available in the ...\MapForceExamples folder shows an example of where CDATA sections can be very useful.

In this example:

- Bold start (<b>) and end (</b>) tags are added to the content of the Trademark source element.
- Italic start (<i>) and end (</i>) tags are added to the content of the Keyword source element.
- The resulting data is passed on to duplicate text() nodes in the order that they appear in the source document, due to the fact the Subsection element connector, has been defined as a Source Driven (Mixed content) node.
- The output of the MixedContent node is then passed on to the Description node in the ShortInfo target component, which has been defined as a CDATA section.

Clicking the Output button shows the CDATA section containing the marked-up text.

7.1.9 Wildcards - xs:any / xs:anyAttribute

The wildcards xs:any (and xs:anyAttribute) allow you to use any elements/attributes from schemas. The screenshot shows the “any” element in the Schema view of XMLSpy.
In MapForce, a **Change Selection** (牢记使命) button appears to the right of the \texttt{xs:}any element (or \texttt{xs:}anyAttribute).

When clicked, the **Change Selection** button opens the "Wildcard selections" dialog box. The entries in this list show the global elements and attributes declared in the current schema.
Clicking one or more of the check boxes and confirming with OK, inserts that element/attribute (and any other child nodes) into the component. The wildcard elements or attributes are inserted immediately after the node whose **Change Selection** button was clicked.

You can now map to/from these nodes as with any other element.

On a component, the wildcard elements or attributes can be recognized by the (xs:any) text appended to their name.

To remove a wildcard element, click the Change Selection button, and then deselect it from the "Wildcard selections" dialog box.

**Wildcards and dynamic node names**

Mapping data to or from wildcards is generally suitable where all possible elements or attributes that appear in the XML instance are declared by the component's XML schema (or can be imported from external schemas). However, there may be situations where elements or attributes appearing in an instance are too many to be declared in the schema. Consider the following instance where the number of child elements of <message> is arbitrary:
For such situations, use dynamic access to node names (see Mapping Node Names) instead of wildcards.

Adding elements from a different schema as wildcards

Elements from a schema other than the one assigned to the component can also be used as wildcards. To make such elements visible on the component, click the Import a different schema button on the "Wildcard selections" dialog box. This opens a new dialog box where you have two options:

1. Import schema
2. Generate wrapper schema

For example, the image below illustrates what happens if you attempt to import an external schema called HasExpenses.xsd into a current schema assigned to a component.

The Import schema option imports the external schema into the current schema assigned to the component. Be aware that this option overrides the existing schema of the component on the disk. If the current schema is a remote schema that was opened from a URL (see Adding Components from a URL) and not from the disk, it cannot be modified. In this case, use the Generate wrapper schema option.
The *Generate wrapper schema* option creates a new schema file called a "wrapper" schema. The advantage of using this option is that the existing schema of the component is not modified. Instead, a new schema will be created (that is, the wrapper schema) which will include both the existing schema and the schema to be imported. When you select this option, you are prompted to choose where the wrapper schema should be saved. By default, the wrapper schema has a name in the form `somefile-wrapper.xsd`. After you save the wrapper schema, it is by default automatically assigned to the component, and a dialog box prompts you:

![MapForce dialog box](image)

Click **Yes** to revert to the previous schema; otherwise click **No** to keep the newly created wrapper schema assigned to the component.

### 7.1.10 Merging Data from Multiple Schemas

MapForce allows you to merge multiple files into a single target file.

This example merges multiple source components with different schemas to a target schema. To merge an arbitrary number of files using the same schema, see [Processing Multiple Input or Output Files Dynamically](#).

The `CompletePO.mfd` file available in the `...\MapForceExamples` folder shows how three XML files are merged into one purchasing order XML file.
Note that multiple source component data are combined into one target XML file - CompletePO

- **ShortPO** is a schema with an associated XML instance file and contains only customer number and article data, i.e. Line item, number and amount. (There is only one customer in this file with the Customer number of 3)
- **Customers** is a schema with an associated XML instance file and contains customer number and customer information details, i.e. Name and Address info.
- **Articles** is a schema with an associated XML instance and contains article data, i.e. article name number and price.
- **CompletePO** is a schema file without an instance file as all the data is supplied by the three XML instance files. The hierarchical structure of this file makes it possible to merge and output all XML data.

This schema file has to be created in an XML editor such as XMLSpy, it is not generated by MapForce (although it would be possible to create if you had a **CompletePO.xml** instance file).

The structure of CompletePO is a combination of the source XML file structures.

The **filter** component (Customer) is used to find/filter the data where the customer numbers are identical in both the ShortPO and Customers XML files, and pass on the associated data to the target CompletePO component.

- The **CustomerNr** in ShortPO is compared with the **Number** in Customers using the "equal" function.
• As ShortPO only contains one customer (number 3), only customer and article data for customer number 3, can be passed on to the filter component.
• The node/row parameter, of the filter component, passes on the Customer data to "on-true" when the bool parameter is true, i.e. when the same number has been found, in this case customer number 3.
• The rest of the customer and article data are passed on to the target schema through the two other filter components.

7.1.11 Declaring Custom Namespaces

By default, when a mapping produces XML output, the namespace (or set of namespaces) of each element and attribute is automatically derived by MapForce from the schema associated with the target component. This is the default behavior in MapForce and is suitable for most mapping scenarios that involve generation of XML output.

However, there might be cases when you want to have more control over namespaces of elements in the resulting XML output. For example, you may want to manually declare the namespace of an element directly from the mapping.

To understand how this works, open the BooksToLibrary.mfd mapping available in the <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial. Right-click the library node, and select Add Namespace from the context menu.

Notice that two new nodes are now available under the library node: a namespace and a prefix.
You can now map to them string values from the mapping. In the image below, two constants were defined (from Insert | Constant menu command) that provide the namespace "altova.library" and the prefix "lib":

The result is that, in the generated output, an xmlns:<prefix>="<namespace>" attribute is added to the element, where <prefix> and <namespace> are values that come from the mapping (in this case, from constants). The generated output will now look as follows (notice the highlighted part):

```
<?xml version="1.0" encoding="UTF-8"?>
<library xmlns:lib="altova.library" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="library.xsd">...
```

**Note:** Declaring custom namespaces (and the Add Namespace command) is meaningful only for target XML components, and applies to elements only. The Add Namespace
command is not available for attributes and wildcard nodes. It is also not available for nodes which receive data by means of a **Copy-All** connection.

You can also declare multiple namespaces for the same element, if necessary. To do this, right-click the node again, and select **Add Namespace** from the context menu. A new pair of namespace and prefix nodes become available, to which you can connect the new prefix and namespace values.

To remove a previously added namespace declaration, right-click the `ns:namespace` node, and select **Remove Namespace** from the context menu.

Both the `namespace` and `prefix` input connectors must be mapped, even if you provide empty values to them.

If you want to declare a default namespace (that is, one in the format `xmlns="mydefaultnamespace"`), map an empty string value to `prefix`. To see this case in action, edit the example mapping above so as to make the second constant an empty string.

The resulting output would then looks as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<library xmlns="altova.library" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="library.xsd">
...
</library>
```

If you need to create prefixes for attribute names, for example `<number prod:id="prod557">557</number>`, you can achieve this by either enabling dynamic access to node's attributes (see **Mapping Node Names**), or by editing the schema so that it has a `prod:id` attribute for `<number>`. 
7.1.12 Digital Signatures

Digital signatures are a W3C specification to digitally sign an XML document with an encrypted code that can be used to verify that the XML document has not been altered. The XML Signature feature in MapForce supports only certificates of type RSA-SHA1 and DSA-SHA1.

For more details about XML signatures, see the W3C specification for XML signatures at https://www.w3.org/TR/xmldsig-core/

MapForce supports creating XML digital signatures for XML and XBRL output files. Digital signatures can only be generated when the output target is BUILT-IN and only in the preview. A signature is created for the generated result file, when the output button is pressed, and the result file is saved.

Note: MapForce Server does not support digital signatures.

Digital signatures can be embedded as the last element of the output document or stored in a separate signature file.

- If "Enveloped" is selected, then the signature is the last child element below the root element of the XML file.
- If "Detached" is selected, then the signature file is generated as a separate document.

To activate generation of digital signatures:

1. Open the Component Settings dialog box of the output component, by double-clicking its header, or by selecting Component | Properties.
2. Select the Create digital signature check box.
3. The XML Signature Settings dialog box opens, where you can define the required settings (see XML Signature Settings).

To change settings for digital signatures:

1. Open the Component Settings dialog box of the output component, by double-clicking its header, or by selecting Component | Properties.
2. Click the "Signature Settings" button to open the XML Signature Settings dialog box.
3. Enter settings and click OK.

Using the MarketingExpenses_DetachedSignature.mfd file in the ...\MapForceExamples folder, as an example:

1. Double click the MarketingExpenses target component, then click the "Signature Settings" button. The selected options are shown.
2. Click OK to close the dialog box.
3. Click the Output button to see the mapping result.

Two files are generated in the preview window. The first file, `MarketingExpenses.xml`, is the mapping result of that target component.

The second file, `MarketingExpenses.xml.xsig`, is the temporary digital signature file generated by the target component.

To generate the signature file, click the **Save all generated outputs** toolbar button. This generates the .xml and .xsig files in the output directory.

The `MarketingExpenses_EnvelopedSignature.mfd` file in the `...\MapForceExamples` folder shows the result when the signature placement is "Enveloped".
XML document validity

If an XML signature is embedded in the XML document, a Signature element in the namespace http://www.w3.org/2000/09/xmldsig# is added to the XML document. In order for the document to remain valid according to a schema, the schema must contain the appropriate element declarations. MapForce embeds signatures using the Enveloped option:

- **Enveloped**: The Signature element is created as the last child element of the root (or document) element.

  If you do not wish to modify the schema of the XML document, the XML signature can be created in an external file using the "Detached" option.

Given below are excerpts from XML Schemas that show how the Signature element of an enveloped signature can be allowed. You can use these examples as guides to modify your own schemas.

In the first of the two listings below, the XML Signature Schema is imported into the user's schema. The XML Signature Schema is located at the web address: [https://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd](https://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd)

```xml
<?xml version="1.0" encoding="UTF-8"?>
```
A second option (listing below) is to add a generic wildcard element which matches any element from other namespaces. Setting the processContents attribute to lax causes the validator to skip over this element—because no matching element declaration is found. Consequently, the user does not need to reference the XML Signatures Schema. The drawback of this option, however, is that any element (not just the Signature element) can be added at the specified location in the XML document without invalidating the XML document.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:element name="Root">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="selection"/>
        <xs:element ref="newsitems" minOccurs="0"/>
        <xs:element ref="team" minOccurs="0"/>
        <xs:any namespace="##other" minOccurs="0" processContents="lax"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

7.1.12.1 XML Signature Settings

Signature settings are stored for each component individually in the component settings dialog box, and are all stored in the MFD file when it is saved.
Authentication method: Certificate or Password

The signature can be based on a certificate or a password. Select the radio button of the method you wish to use.

- **Certificate:**
  
  If you wish to use a certificate, the certificate must have a private key and be located in an accessible certificate store. The signature is generated using the private key of the certificate. To verify the signature, access to the certificate (or a public-key version of it) is required. The public key of the certificate is used to verify the signature. To select the
private-public-key certificate you wish to use, click the Select button and browse for the certificate.

- **Password:**
Enter a password with a length of 5 to 16 characters. This password is used to create the signature and will subsequently be required to verify the signature. The OK button of the dialog box only becomes active if this requirement is fulfilled.

- **Save password in MFD file:**
When active, the password entered in the Password field is saved in obfuscated form in the MFD file, i.e. the password is encrypted and not human readable. Note that anyone who has access to the MFD file can create signatures using this password.

### Transformations

The XML data is transformed and the result of the transformation is used for the creation of the signature. You can specify the canonicalization algorithm to be applied to the file's XML data (the SignedInfo content) prior to performing signature calculations. Significant points of difference between the algorithms are noted below:

- **Canonical XML with or without comments:**
If comments are included for signature calculation, then any change to comments in the XML data will result in verification failure. Otherwise, comments may be modified or be added to the XML document after the document has been signed, and the signature will still be verified as authentic.

  Note:
  "...with comments" is only available for "Detached" placement.

- **Base64:**
The root (or document) element of the XML document is considered to be Base64 encoded, and is read in its binary form. If the root element is not Base64, an error is returned or the element is read as empty, depending on what type of element is encountered.

- **None:**
No transformation is carried out and the XML data from the binary file saved on disk, is passed directly for signature creation. Any subsequent change in the data will result in a failed verification of the signature.

  However, if the **Strip Whitespace between XML elements** check box option is selected, then all whitespace is stripped and changes in whitespace will be ignored.

A major difference between the **None** option and a **Canonicalization** option is that canonicalization produces an XML data stream, in which some differences, such as attribute order, are normalized. As a result, a canonicalization transformation will normalize any changes such as that of attribute order (so verification will succeed), while no-transformation will reflect such a change (verification will fail).

Note, however, that a default canonicalization is performed if the signature is embedded (enveloped). So the XML data will be used as is (i.e. with no transformation) when: the signature is detached, **None** is selected, and the **Strip Whitespaces** check box is unchecked.
Signature Placement

The signature can be placed within the XML file or be created as a separate file. The following options are available:

- **Enveloped:**
  The signature element is created as the last child element of the root (document) element. Note: the associated XML Schema must contain the signature definition elements for the output XML to be valid. Please see the top of this section for more information.

- **Detached:**
  The XML signature is created as a separate file. In this case, you can specify the file extension of the signature file and whether the file name is created with: (i) the extension appended to the name of the XML file (for example, test.xml.xsig), or (ii) the extension replaces the XML extension of the XML file (for example, test.xsig). You can also specify whether, in the signature file, the reference to the XML file is a relative or an absolute path.

**Note:** XML signatures for XML Schema (.xsd) files and for XBRL files can only be created as external signature files. For WSDL files, signatures can be created as external files and can be "enveloped" in the WSDL file.

**Note:** If the XML signature is created as a separate file, then the XML file and signature file are associated with each other via a reference in the signature file. Consequently, signature verification in cases where the signature is in an external file must be done with the signature file active—not with the XML file active.

Append KeyInfo

The Append KeyInfo option is available when the signature is certificate-based. It is unavailable if the signature is password-based.

If Append KeyInfo is active/checked, public-key information is placed inside the signature, otherwise key information is not included in the signature. The advantage of including key information is that the certificate itself (specifically the public-key information in it) will not be required for the verification process (since the key information is present in the signature).

Invalid signature settings

MapForce cannot digitally sign an output if the signature settings are invalid. Signature settings are invalid if:

- The selected certificate is not accessible, or is not suitable for signing xml documents, or
- No password is set, e.g. the option "Save password in mfd file" is not checked.

When clicking the Output button MapForce prompts -

for the password with:

Please specify a password to sign the output of component "MarketingExpensesSigned"

for the certificate with:

Please choose the store containing the certificate you want to use to sign the output of
component "MarketingExpensesSigned".

If no password or certificate is chosen, then the processing is either stopped, or continued without a signature. You can determine this behavior in the Component Settings dialog box via the "Stop processing" or "Continue without signature" radio buttons.

If the mapping is executed from the command line, no prompt dialog box appears. The mapping execution either stops with an error, or continues without signature.
## 7.2 Databases and MapForce

**Altova website:** [Database mapping](https://www.altova.com/database-mapping)

MapForce 2019 provides powerful support for mapping databases to XML, flat files, and other database formats. With MapForce Enterprise edition, you can additionally map databases to EDI formats, Excel 2007+, JSON, XBRL, and Web services.

The following databases are supported. While Altova endeavors to support other databases, successful connection and data processing have only been tested with the databases listed below. If your Altova application is a 64-bit version, ensure that you have access to the 64-bit database drivers needed for the specific database you are connecting to.

<table>
<thead>
<tr>
<th>Database</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firebird 2.5.4, 3.0</td>
<td></td>
</tr>
<tr>
<td>IBM DB2 8.x, 9.1, 9.5, 9.7, 10.1, 10.5</td>
<td></td>
</tr>
<tr>
<td>IBM Db2 for i 6.1, 7.1, 7.2, 7.3</td>
<td>Logical files are supported and shown as views.</td>
</tr>
<tr>
<td>IBM Informix 11.70, 12.10</td>
<td>Informix supports connections via ADO, JDBC and ODBC. The implementation does not support large object data types in any of the code generation languages. MapForce will generate an error message (during code generation) if any of these data types are used.</td>
</tr>
<tr>
<td>MariaDB 10.2, 10.3</td>
<td></td>
</tr>
<tr>
<td>Microsoft Azure SQL Database</td>
<td>SQL Server 2016 codebase</td>
</tr>
<tr>
<td>MySQL 5.0, 5.1, 5.5, 5.6, 5.7, 8.0</td>
<td></td>
</tr>
<tr>
<td>Oracle 9i, 10g, 11g, 12c</td>
<td></td>
</tr>
<tr>
<td>PostgreSQL 8.0, 8.1, 8.2, 8.3, 9.0.10, 9.1.6, 9.2.1, 9.4, 9.5, 9.6, 10, 11</td>
<td>PostgreSQL connections are supported both as native connections and driver-based connections through interfaces (drivers) such as ODBC or JDBC. Native connections do not require any drivers.</td>
</tr>
<tr>
<td>Progress OpenEdge 11.6</td>
<td></td>
</tr>
<tr>
<td>SQLite 3.x</td>
<td>SQLite connections are supported as native, direct connections to the SQLite database file. No separate drivers are required.</td>
</tr>
</tbody>
</table>
### Database mappings in various execution environments

When you generate program code from a mapping, or when you compile a mapping to MapForce Server execution files, or when you deploy a mapping to FlowForce Server, the database connection details saved with the generated files are adapted to drivers applicable or supported for the chosen target environment, as shown in the table below. For example, if the mapping transformation language is set to Java, ADO connections are converted to JDBC when Java code is generated from the mapping.

When the mapping is executed in an environment other than MapForce, you will need to make sure that the database connection details are meaningful on the machine which executes the mapping (for example, the database driver is installed, the database path is correct, the database server is accessible, etc.).

Some database connection types are not supported in some target environments, as shown in the table below.

<table>
<thead>
<tr>
<th>Connection type/Execution Environment</th>
<th>C#</th>
<th>C++</th>
<th>Java</th>
<th>MapForce Server on Windows</th>
<th>MapForce Server on Linux/Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADO</td>
<td>ADO bridge</td>
<td>As is</td>
<td>Converted to JDBC</td>
<td>As is</td>
<td>Converted to JDBC</td>
</tr>
<tr>
<td>ADO.NET</td>
<td>As is</td>
<td>User defined</td>
<td>Converted to JDBC</td>
<td>As is</td>
<td>Converted to JDBC</td>
</tr>
<tr>
<td>JDBC</td>
<td>User defined</td>
<td>User defined</td>
<td>As is</td>
<td>As is</td>
<td>As is</td>
</tr>
<tr>
<td>ODBC</td>
<td>ODBC bridge</td>
<td>ODBC bridge</td>
<td>Converted to JDBC</td>
<td>As is</td>
<td>Converted to JDBC</td>
</tr>
<tr>
<td>Native PostgreSQL</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>As is</td>
<td>As is</td>
</tr>
<tr>
<td>Native SQLite</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>As is</td>
<td>As is</td>
</tr>
</tbody>
</table>

Table legend:

- "As is" means that the database connection type (for example, JDBC, ODBC) remains as defined in MapForce.
7.2.1 Connecting to a Database

In the most simple case, a database can be a local file such as a Microsoft Access or SQLite database file. In a more advanced scenario, a database may reside on a remote or network database server which does not necessarily use the same operating system as the application that connects to it and consumes data. For example, while MapForce runs on a Windows operating system, the database from which you want to access data (for example, MySQL) might run on a Linux machine.

To interact with various database types, both remote and local, MapForce relies on the data connection interfaces and database drivers that are already available on your operating system or released periodically by the major database vendors. In the constantly evolving landscape of database technologies, this approach caters for better cross-platform flexibility and interoperability.

The following diagram illustrates, in a simplified way, data connectivity options available between MapForce (illustrated as a generic client application) and a data store (which may be a database server or database file).
* Direct native connections are supported for SQLite and PostgreSQL databases. To connect to such databases, no additional drivers are required to be installed on your system.

As shown in the diagram above, MapForce can access any of the major database types through the following data access technologies:

- ADO (Microsoft® ActiveX® Data Objects), which, in its turn, uses an underlying OLE DB (Object Linking and Embedding, Database) provider
- ADO.NET (A set of libraries available in the Microsoft .NET Framework that enable interaction with data)
- JDBC (Java Database Connectivity)
- ODBC (Open Database Connectivity)

Some ADO.NET providers are not supported or have limited support. See [ADO.NET Support Notes](#).

The data connection interface you should choose largely depends on your existing software infrastructure. You will typically choose the data access technology and the database driver which integrates tighter with the database system to which you want to connect. For example, to connect to a Microsoft Access 2013 database, you would build an ADO connection string that uses a native provider such as the [Microsoft Office Access Database Engine OLE DB Provider](#). To connect to Oracle, on the other hand, you may want to download and install the latest JDBC, ODBC, or ADO.NET interfaces from the Oracle website.

While drivers for Windows products (such as Microsoft Access or SQL Server) may already be available on your Windows operating system, they may not be available for other database types. Major database vendors routinely release publicly available database client software and drivers which provide cross-platform access to the respective database through any combination of ADO,
ADO.NET, ODBC, or JDBC. In addition to this, several third party drivers may be available for any of the above technologies. In most cases, there is more than one way to connect to the required database from your operating system, and, consequently, from MapForce. The available features, performance parameters, and the known issues will typically vary based on the data access technology or drivers used.

7.2.1.1 *Starting the Database Connection Wizard*

Whenever you take an action that requires a database connection, a wizard appears that guides you through the steps required to set up the connection.

Before you go through the wizard steps, be aware that for some database types it is necessary to install and configure separately several database prerequisites, such as a database driver or database client software. These are normally provided by the respective database vendors, and include documentation tailored to your specific Windows version. For a list of database drivers grouped by database type, see [Database Drivers Overview](#).

**To add the database as a source or target component on a mapping:**

- On the **Insert** menu, click **Database**.

**To add the database as a reusable global resource:**

1. On the **Tools** menu, click **Global Resources**.
2. Click **Add**, and then click Database.
3. Click **Choose Database**.
After you select a database type and click **Next**, the on-screen instructions will depend on the database kind, technology (ADO, ADO.NET, ODBC, JDBC) and driver used.

For examples applicable to each database type, see [Database Connection Examples](#). For instructions applicable to each database access technology, refer to the following topics:

- Setting up an ADO Connection
- Setting up an ADO.NET Connection
- Setting up an ODBC Connection
- Setting up a JDBC Connection
7.2.1.2 Database Drivers Overview

The following table lists common database drivers you can use to connect to a particular database through a particular data access technology. Note that this list does not aim to be either exhaustive or prescriptive; you can use other native or third party alternatives in addition to the drivers shown below.

Even though a number of database drivers might be already available on your Windows operating system, you may still need to download an alternative driver. For some databases, the latest driver supplied by the database vendor is likely to perform better than the driver that shipped with the operating system.

Database vendors may provide drivers either as separate downloadable packages, or bundled with database client software. In the latter case, the database client software normally includes any required database drivers, or provides you with an option during installation to select the drivers and components you wish to install. Database client software typically consists of administration and configuration utilities used to simplify database administration and connectivity, as well as documentation on how to install and configure the database client and any of its components.

Configuring the database client correctly is crucial for establishing a successful connection to the database. Before installing and using the database client software, it is strongly recommended to read carefully the installation and configuration instructions of the database client; these may vary for each database version and for each Windows version.

To understand the capabilities and limitations of each data access technology with respect to each database type, refer to the documentation of that particular database product and also test the connection against your specific environment. To avoid common connectivity issues, note the following:

- Some ADO.NET providers are not supported or have limited support. See ADO.NET Support Notes.
- When installing a database driver, it is recommended that it has the same platform as the Altova application (32-bit or 64-bit). For example, if you are using a 32-bit Altova application on a 64-bit operating system, install the 32-bit driver, and set up your database connection using the 32-bit driver, see also Viewing the Available ODBC Drivers.
- When setting up an ODBC data source, it is recommended to create the data source name (DSN) as System DSN instead of User DSN. For more information, see Setting up an ODBC Connection.
- When setting up a JDBC data source, ensure that JRE (Java Runtime Environment) or Java Development Kit (JDK) is installed and that the CLASSPATH environment variable of the operating system is configured. For more information, see Setting up a JDBC Connection.
- For the installation instructions and support details of any drivers or database client software that you install from a database vendor, check the documentation provided with the installation package.

<table>
<thead>
<tr>
<th>Database</th>
<th>Interface</th>
<th>Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Interface</td>
<td>Drivers</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>JDBC</td>
<td>Firebird JDBC driver (<a href="https://www.firebirdsql.org/en/jdbc-driver/">https://www.firebirdsql.org/en/jdbc-driver/</a>)</td>
</tr>
<tr>
<td></td>
<td>ODBC</td>
<td>Firebird ODBC driver (<a href="https://www.firebirdsql.org/en/odbc-driver/">https://www.firebirdsql.org/en/odbc-driver/</a>)</td>
</tr>
<tr>
<td>IBM DB2</td>
<td>ADO</td>
<td>IBM OLE DB Provider for DB2</td>
</tr>
<tr>
<td></td>
<td>ADO.NET</td>
<td>IBM Data Server Provider for .NET</td>
</tr>
<tr>
<td></td>
<td>JDBC</td>
<td>IBM Data Server Driver for JDBC and SQLJ</td>
</tr>
<tr>
<td></td>
<td>ODBC</td>
<td>IBM DB2 ODBC Driver</td>
</tr>
<tr>
<td>IBM DB2 for i</td>
<td>ADO</td>
<td>• IBM DB2 for i5/OS IBMDA400 OLE DB Provider</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• IBM DB2 for i5/OS IBMDARLA OLE DB Provider</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• IBM DB2 for i5/OS IBMDASQL OLE DB Provider</td>
</tr>
<tr>
<td></td>
<td>ADO.NET</td>
<td>.NET Framework Data Provider for IBM i</td>
</tr>
<tr>
<td></td>
<td>JDBC</td>
<td>IBM Toolbox for Java JDBC Driver</td>
</tr>
<tr>
<td></td>
<td>ODBC</td>
<td>iSeries Access ODBC Driver</td>
</tr>
<tr>
<td>IBM Informix</td>
<td>ADO</td>
<td>IBM Informix OLE DB Provider</td>
</tr>
<tr>
<td></td>
<td>JDBC</td>
<td>IBM Informix JDBC Driver</td>
</tr>
<tr>
<td></td>
<td>ODBC</td>
<td>IBM Informix ODBC Driver</td>
</tr>
<tr>
<td>Microsoft</td>
<td>ADO</td>
<td>• Microsoft OLE DB Provider</td>
</tr>
<tr>
<td>Access</td>
<td></td>
<td>• Microsoft Access Database Engine OLE DB Provider</td>
</tr>
<tr>
<td></td>
<td>ADO.NET</td>
<td>.NET Framework Data Provider for OLE DB</td>
</tr>
<tr>
<td></td>
<td>ODBC</td>
<td>• Microsoft Access Driver</td>
</tr>
<tr>
<td>MariaDB</td>
<td>ADO.NET</td>
<td>In the absence of a dedicated .NET connector for MariaDB, use Connector/NET for MySQL (<a href="https://dev.mysql.com/downloads/connector/net/">https://dev.mysql.com/downloads/connector/net/</a>).</td>
</tr>
<tr>
<td></td>
<td>JDBC</td>
<td>MariaDB Connector/J (<a href="https://downloads.mariadb.org/">https://downloads.mariadb.org/</a>)</td>
</tr>
<tr>
<td></td>
<td>ODBC</td>
<td>MariaDB Connector/ODBC (<a href="https://downloads.mariadb.org/">https://downloads.mariadb.org/</a>)</td>
</tr>
<tr>
<td>Microsoft</td>
<td>ADO</td>
<td>• Microsoft OLE DB Provider for SQL Server</td>
</tr>
<tr>
<td>SQL Server</td>
<td></td>
<td>• SQL Server Native Client</td>
</tr>
<tr>
<td></td>
<td>ADO.NET</td>
<td>• .NET Framework Data Provider for SQL Server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• .NET Framework Data Provider for OLE DB</td>
</tr>
<tr>
<td></td>
<td>JDBC</td>
<td>• Microsoft JDBC Driver for SQL Server (<a href="https://docs.microsoft.com/en-us/sql/connect/jdbc/microsoft-jdbc-driver-for-sql-server">https://docs.microsoft.com/en-us/sql/connect/jdbc/microsoft-jdbc-driver-for-sql-server</a>)</td>
</tr>
<tr>
<td></td>
<td>ODBC</td>
<td>• SQL Server Native Client</td>
</tr>
<tr>
<td>MySQL</td>
<td>ADO.NET</td>
<td>Connector/NET (<a href="https://dev.mysql.com/downloads/connector/net/">https://dev.mysql.com/downloads/connector/net/</a>)</td>
</tr>
<tr>
<td>Database</td>
<td>Interface</td>
<td>Drivers</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Oracle       | ADO       | • Oracle Provider for OLE DB  
• Microsoft OLE DB Provider for Oracle                                       |
|              | ADO.NET   | Oracle Data Provider for .NET ([http://www.oracle.com/technetwork/topics/dotnet/index-085163.html](http://www.oracle.com/technetwork/topics/dotnet/index-085163.html)) |
|              | JDBC      | • JDBC Thin Driver  
• JDBC Oracle Call Interface (OCI) Driver  
These drivers are typically installed during the installation of your Oracle database client. Connect through the OCI Driver (not the Thin Driver) if you are using the Oracle XML DB component. |
|              | ODBC      | • Microsoft ODBC for Oracle  
• Oracle ODBC Driver (typically installed during the installation of your Oracle database client) |
| PostgreSQL   | JDBC      | PostgreSQL JDBC Driver ([https://jdbc.postgresql.org/download.html](https://jdbc.postgresql.org/download.html)) |
|              | ODBC      | psqlODBC ([https://odbc.postgresql.org/](https://odbc.postgresql.org/)) |
|              | Native Connection | Available. There is no need to install any drivers if using native connection. |
| Progress OpenEdge | JDBC     | JDBC Connector ([https://www.progress.com/jdbc/openedge](https://www.progress.com/jdbc/openedge)) |
|              | ODBC      | ODBC Connector ([https://www.progress.com/odbc/openedge](https://www.progress.com/odbc/openedge)) |
| SQLite       | Native Connection | Available. There is no need to install any drivers if using native connection. |
| Sybase       | ADO       | Sybase ASE OLE DB Provider |
|              | JDBC      | jConnect™ for JDBC |
|              | ODBC      | Sybase ASE ODBC Driver |
|              | JDBC      | Teradata JDBC Driver ([https://downloads.teradata.com/download/connectivity/jdbc-driver](https://downloads.teradata.com/download/connectivity/jdbc-driver)) |
7.2.1.3 Setting up an ADO Connection

Microsoft ActiveX Data Objects (ADO) is a data access technology that enables you to connect to a variety of data sources through OLE DB. OLE DB is an alternative interface to ODBC or JDBC; it provides uniform access to data in a COM (Component Object Model) environment. ADO is the typical choice for connecting to Microsoft native databases such as Microsoft Access or SQL Server, although you can also use it for other data sources.

To set up an ADO connection:

1. Start the database connection wizard.
2. Click ADO Connections.

![ADO Connections](image)

Generate a valid connection string by clicking on the build button. Once the connection string is generated click on the 'Connect' button to proceed.

3. Click Build.
4. Select the data provider through which you want to connect. The table below lists a few common scenarios.

<table>
<thead>
<tr>
<th>To connect to this database...</th>
<th>Use this provider...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Access</td>
<td>• Microsoft Office Access Database Engine OLE DB Provider</td>
</tr>
<tr>
<td></td>
<td>When connecting to Microsoft Access 2003, you can also use the Microsoft Jet OLE DB Provider.</td>
</tr>
</tbody>
</table>
| SQL Server                     | • SQL Server Native Client  
|                                | • Microsoft OLE DB Provider for SQL Server |
| Other database                 | Select the provider applicable to your database. |
|                                | If an OLE DB provider to your database is not available, install the required driver from the database vendor (see Database Drivers Overview). Alternatively, set up an ODBC or JDBC connection. |
|                                | If the operating system has an ODBC driver to the required database, you can also use the Microsoft OLE DB Provider for ODBC Drivers. |

5. Click **Next** and complete the wizard.
The subsequent wizard steps are specific to the provider you chose. For SQL Server, you will need to provide or select the host name of the database server, as well as the database username and password. For Microsoft Access, you will be asked to browse for or provide the path to the database file.

The complete list of initialization properties (connection parameters) is available in the All tab of the connection dialog box—these properties vary depending on the chosen provider. The following sections provide guidance on configuring the basic initialization properties for Microsoft Access and SQL Server databases:

- Setting up the SQL Server Data Link Properties
- Setting up the Microsoft Access Data Link Properties

7.2.1.3.1 Connecting to an Existing Microsoft Access Database

This approach is suitable when you want to connect to a Microsoft Access database which is not password-protected. If the database is password-protected, set up the database password as shown in Connecting to Microsoft Access (ADO).

To connect to an existing Microsoft Access database:

1. Run the database connection wizard (see Starting the Database Connection Wizard).
2. Select Microsoft Access (ADO), and then click Next.
3. Browse for the database file, or enter the path to it (either relative or absolute).
4. Click Connect.

7.2.1.3.2 Setting up the SQL Server Data Link Properties

When you connect to a Microsoft SQL Server database through ADO (see Setting up an ADO Connection), ensure that the following data link properties are configured correctly in the All tab of the Data Link Properties dialog box.
### Setting up the Microsoft Access Data Link Properties

When you connect to a Microsoft Access database through ADO (see Setting up an ADO Connection), ensure that the following properties are configured correctly in the All tab of the Data Link Properties dialog box.

<table>
<thead>
<tr>
<th>Property</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrated Security</td>
<td>If you selected the SQL Server Native Client data provider on the Provider tab, set this property to a space character.</td>
</tr>
<tr>
<td>Persist Security Info</td>
<td>Set this property to True.</td>
</tr>
</tbody>
</table>

**Data Link Properties dialog box**
### Property: Data Source

This property stores the path to the Microsoft Access database file. To avoid database connectivity issues, it is recommended to use the UNC (Universal Naming Convention) path format, for example:

```
\anyserver\share$\filepath
```

### Property: Jet OLEDB:System Database

This property stores the path to the workgroup information file. You may need to explicitly set the value of this property before you can connect to a Microsoft Access database.

If you cannot connect due to a "workgroup information file" error, locate the workgroup information file (`System.MDW`) applicable to your user profile, and set the property value to the path of the `System.MDW` file.
### Property | Notes
---|---
 Jet OLEDB:Database Password | If the database is password-protected, set the value of this property to the database password.

#### 7.2.1.4 Setting up an ADO.NET Connection

ADO.NET is a set of Microsoft .NET Framework libraries designed to interact with data, including data from databases. To connect to a database from MapForce through ADO.NET, Microsoft .NET Framework 4 or later is required. As shown below, you connect to a database through ADO.NET by selecting a .NET provider and supplying a connection string.

A .NET data provider is a collection of classes that enables connecting to a particular type of data source (for example, a SQL Server, or an Oracle database), executing commands against it, and fetching data from it. In other words, with ADO.NET, an application such as MapForce interacts with a database through a data provider. Each data provider is optimized to work with the specific type of data source that it is designed for. There are two types of .NET providers:

1. Supplied by default with Microsoft .NET Framework.
2. Supplied by major database vendors, as an extension to the .NET Framework. Such ADO.NET providers must be installed separately and can typically be downloaded from the website of the respective database vendor.

**Note:** Certain ADO.NET providers are not supported or have limited support. See [ADO.NET Support Notes](#).

To set up an ADO.NET connection:

1. [Start the database connection wizard](#).
2. Click [ADO.NET Connections](#).
3. Select a .NET data provider from the list.

The list of providers available by default with the .NET Framework appears in the "Provider" list. Vendor-specific .NET data providers are available in the list only if they are already installed on your system. To become available, vendor-specific .NET providers must be installed into the GAC (Global Assembly Cache), by running the .msi or .exe file supplied by the database vendor.

4. Enter a database connection string. A connection string defines the database connection information, as semicolon-delimited key/value pairs of connection parameters. For example, a connection string such as `Data Source=DBSQLSERV;Initial Catalog=ProductsDB;User ID=dbuser;Password=dbpass` connects to the SQL Server database `ProductsDB` on server `DBSQLSERV`, with the user name `dbuser` and password `dbpass`. You can create a connection string by typing the key/value pairs directly into the "Connection String" dialog box. Another option is to create it with Visual Studio (see Creating a Connection String in Visual Studio).

The syntax of the connection string depends on the provider selected from the "Provider" list. For examples, see Sample ADO.NET Connection Strings.
5. Click **Connect**.

### 7.2.1.4.1 Creating a Connection String in Visual Studio

In order to connect to a data source using ADO.NET, a valid database connection string is required. The following instructions show you how to create a connection string from Visual Studio.

**To create a connection string in Visual Studio:**

1. On the **Tools** menu, click **Connect to Database**.
2. Select a data source from the list (in this example, Microsoft SQL Server). The Data Provider is filled automatically based on your choice.
3. Click **Continue**.
4. Enter the server host name and the user name and password to the database. In this example, we are connecting to the database ProductsDB on server DBSQLSERV, using SQL Server authentication.

5. Click OK.

If the database connection is successful, it appears in the Server Explorer window. You can display the Server Explorer window using the menu command View | Server Explorer. To obtain the database connection string, right-click the connection in the Server Explorer window, and select Properties. The connection string is now displayed in the Properties window of Visual Studio. Note that, before pasting the string into the “Connection String” box of MapForce, you will need to replace the asterisk ( * ) characters with the actual password.
7.2.1.4.2  Sample ADO.NET Connection Strings

To set up an ADO.NET connection, you need to select an ADO.NET provider from the database connection dialog box and enter a connection string (see also Setting up an ADO.NET Connection). Sample ADO.NET connection strings for various databases are listed below under the .NET provider where they apply.

.NET Data Provider for Teradata
This provider can be downloaded from Teradata website (https://downloads.teradata.com/download/connectivity/net-data-provider-for-teradata). A sample connection string looks as follows:

```
Data Source=ServerAddress;User Id=user;Password=password;
```

.NET Framework Data Provider for IBM i
This provider is installed as part of IBM i Access Client Solutions - Windows Application Package. A sample connection string looks as follows:

```
DataSource=ServerAddress;UserID=user;Password=password;DataCompression=True;
```

For more information, see the " .NET Provider Technical Reference" help file included in the installation package above.

.NET Framework Data Provider for MySQL
This provider can be downloaded from MySQL website (https://dev.mysql.com/downloads/connector/net/). A sample connection string looks as follows:

```
Server=127.0.0.1;Uid=root;Pwd=12345;Database=test;
```


.NET Framework Data Provider for SQL Server
A sample connection string looks as follows:

```
Data Source=DBSQLSERV;Initial Catalog=ProductsDB;User ID=dbuser;Password=dbpass
```

See also: https://msdn.microsoft.com/en-us/library/ms254500(v=vs.110).aspx

IBM DB2 Data Provider 10.1.2 for .NET Framework 4.0
**Note:** This provider is typically installed with the IBM DB2 Data Server Client package. If the provider is missing from the list of ADO.NET providers after installing IBM DB2 Data Server Client package, refer to the following technical note: https://www-01.ibm.com/support/docview.wss?uid=swg21429586.


**Oracle Data Provider for .NET (ODP.NET)**

The installation package which includes the ODP.NET provider can be downloaded from Oracle website (see http://www.oracle.com/technetwork/topics/dotnet/downloads/index.html). A sample connection string looks as follows:

```
Data Source=DSORCL;User Id=user;Password=password;
```

Where **DSORCL** is the name of the data source which points to an Oracle service name defined in the **tnsnames.ora** file, as described in Connecting to Oracle (ODBC).

To connect without configuring a service name in the **tnsnames.ora** file, use a string such as:

```
Data Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=host) (PORT=port))) (CONNECT_DATA=(SERVER=DEDICATED) (SERVICE_NAME=MyOracleSID)));User Id=user;Password=password;
```

See also: https://docs.oracle.com/cd/B28359_01/win.111/b28375/featConnecting.htm

### 7.2.1.4.3 ADO.NET Support Notes

The following table lists known ADO.NET database drivers that are currently not supported or have limited support in MapForce.

<table>
<thead>
<tr>
<th>Database</th>
<th>Driver</th>
<th>Support notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>All databases</td>
<td><a href="#">.Net Framework Data Provider for ODBC</a></td>
<td>Limited support. Known issues exist with Microsoft Access connections. It is recommended to use ODBC direct connections instead.</td>
</tr>
<tr>
<td></td>
<td><a href="#">.Net Framework Data Provider for OleDb</a></td>
<td>Limited support. Known issues exist with Microsoft Access connections. It is recommended to use ADO direct connections instead.</td>
</tr>
<tr>
<td>Firebird</td>
<td><a href="#">Firebird ADO.NET Data Provider</a></td>
<td>Limited support. It is recommended to use ODBC or JDBC instead.</td>
</tr>
</tbody>
</table>
### Setting up an ODBC Connection

ODBC (Open Database Connectivity) is a widely used data access technology that enables you to connect to a database from MapForce. It can be used either as primary means to connect to a database, or as an alternative to OLE DB- or JDBC-driven connections.

To connect to a database through ODBC, first you need to create an ODBC data source name (DSN) on the operating system. This step is not required if the DSN has already been created, perhaps by another user of the operating system. The DSN represents a uniform way to describe the database connection to any ODBC-aware client application on the operating system, including MapForce. DSNs can be of the following types:

- **System DSN**
- **User DSN**
- **File DSN**

A System data source is accessible by all users with privileges on the operating system. A User data source is available to the user who created it. Finally, if you create a File DSN, the data source will be created as a file with the .dsn extension which you can share with other users, provided that they have installed the drivers used by the data source.

Any DSNs already available on your machine are listed by the database connection dialog box when you click ODBC connections on the ODBC connections dialog box.
If a DSN to the required database is not available, the MapForce database connection wizard will assist you to create it; however, you can also create it directly on your Windows operating system. In either case, before you proceed, ensure that the ODBC driver applicable for your database is in the list of ODBC drivers available to the operating system (see Viewing the Available ODBC Drivers).

To connect by using a new DSN:

1. Start the database connection wizard.
2. On the database connection dialog box, click ODBC Connections.
3. Select a data source type (User DSN, System DSN, File DSN).

To create a System DSN, you need administrative rights on the operating system, and MapForce must be run as administrator.

4. Click Add.
5. Select a driver, and then click User DSN or System DSN (depending on the type of the DSN you want to create). If the driver applicable to your database is not listed, download it from the database vendor and install it (see Database Drivers Overview).
6. On the dialog box that pops up, fill in any driver specific connection information to complete the setup.

For the connection to be successful, you will need to provide the host name (or IP address) of the
database server, as well as the database username and password. There may be other optional connection parameters—these parameters vary between database providers. For detailed information about the parameters specific to each connection method, consult the documentation of the driver provider. Once created, the DSN becomes available in the list of data source names. This enables you to reuse the database connection details any time you want to connect to the database. Note that User DSNs are added to the list of User DSNs whereas System DSNs are added to the list of System DSNs.

**To connect by using an existing DSN:**

1. [Start the database connection wizard](#).
2. Click [ODBC Connections](#).
3. Choose the type of the existing data source (User DSN, System DSN, File DSN).
4. Click the existing DSN record, and then click [Connect](#).

**To build a connection string based on an existing .dsn file:**

1. [Start the database connection wizard](#).
2. Click [ODBC Connections](#).
3. Select [Build a connection string](#), and then click [Build](#).
4. If you want to build the connection string using a File DSN, click the [File Data Source](#) tab. Otherwise, click the [Machine Data Source](#) tab. (System DSNs and User DSNs are known as "Machine" data sources.)
5. Select the required .dsn file, and then click [OK](#).

**To connect by using a prepared connection string:**

1. [Start the database connection wizard](#).
2. Click [ODBC Connections](#).
3. Select [Build a connection string](#).
4. Paste the connection string into the provided box, and then click [Connect](#).

### 7.2.1.5.1 Viewing the Available ODBC Drivers

You can view the ODBC drivers available on your operating system in the ODBC Data Source Administrator. You can access the ODBC Data Source Administrator (Odbcad32.exe) from the Windows Control Panel, under [Administrative Tools](#). On 64-bit operating systems, there are two versions of this executable:

- The 32-bit version of the Odbcad32.exe file is located in the C:\Windows\SysWoW64 directory (assuming that C: is your system drive).
- The 64-bit version of the Odbcad32.exe file is located in the C:\Windows\System32 directory.

Any installed 32-bit database drivers are visible in the 32-bit version of ODBC Data Source Administrator, while 64-bit drivers—in the 64-bit version. Therefore, ensure that you check the database drivers from the relevant version of ODBC Data Source Administrator.
If the driver to your target database does not exist in the list, or if you want to add an alternative driver, you will need to download it from the database vendor (see Database Drivers Overview). Once the ODBC driver is available on your system, you are ready to create ODBC connections with it (see Setting up an ODBC Connection).

7.2.1.6 Setting up a JDBC Connection

JDBC (Java Database Connectivity) is a database access interface which is part of the Java software platform from Oracle. JDBC connections are generally more resource-intensive than ODBC connections but may provide features not available through ODBC.

Prerequisites

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application Options, see Java Settings; b) The JVM path found in the Windows registry; c) The JAVA_HOME environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The JDBC drivers from the database vendor must be installed. If you are connecting to an Oracle database, note that some Oracle drivers are specific to certain JRE versions and may require additional components and configuration. The documentation of your Oracle
product (for example, the "Oracle Database JDBC Developer's Guide and Reference") includes detailed instructions about the configuration procedure for each JDBC driver.

- The CLASSPATH environment variable must include the path to the JDBC driver (one or several .jar files) on your Windows operating system. When you install some database clients, the installer may configure this variable automatically. The documentation of the JDBC driver will typically include step-by-step instructions on setting the CLASSPATH variable (see also Configuring the CLASSPATH).

**Setting up a JDBC connection**

1. **Start the database connection wizard.**
2. **Click JDBC Connections.**
3. Optionally, enter a semicolon-separated list of .jar file paths in the "Classpaths" text box. The .jar libraries entered here will be loaded into the environment in addition to those already defined in the CLASSPATH environment variable. When you finish editing the "Classpaths" text box, any JDBC drivers found in the source .jar libraries are automatically added to the "Driver" list (see the next step).

<table>
<thead>
<tr>
<th>Classpaths:</th>
<th>C:\jdbc\instantclient_12_1\odbc7.jar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver:</td>
<td>oracle.jdbc.driver.OracleDriver</td>
</tr>
<tr>
<td>Username:</td>
<td>jchndoe</td>
</tr>
<tr>
<td>Password:</td>
<td>*******</td>
</tr>
<tr>
<td>Database URL:</td>
<td>jdbc:oracle:thin:@//ora12c:1521:ord12c</td>
</tr>
</tbody>
</table>

4. Next to "Driver", select a JDBC driver from the list, or enter a Java class name. Note that this list contains any JDBC drivers configured through the CLASSPATH environment variable (see Configuring the CLASSPATH), as well as those found in the "Classpaths" text box.

The JDBC driver paths defined in the CLASSPATH variable, as well as any .jar file paths entered directly in the database connection dialog box are all supplied to the Java Virtual Machine (JVM). The JVM then decides which drivers to use in order to
establish a connection. It is recommended to keep track of Java classes loaded into
the JVM so as not to create potential JDBC driver conflicts and avoid unexpected
results when connecting to the database.

5. Enter the username and password to the database in the corresponding boxes.
6. In the Database URL text box, enter the JDBC connection URL (string) in the format
specific to your database type. The following table describes the syntax of JDBC
connection URLs (strings) for common database types.

<table>
<thead>
<tr>
<th>Database</th>
<th>JDBC Connection URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firebird</td>
<td>jdbc:firebirdsql://&lt;host&gt;[:&lt;port&gt;]/&lt;database path or alias&gt;</td>
</tr>
<tr>
<td>IBM DB2</td>
<td>jdbc:db2://hostName:port/databaseName</td>
</tr>
<tr>
<td>IBM DB2 for i</td>
<td>jdbc:as400://[host]</td>
</tr>
</tbody>
</table>
| IBM Informix      | jdbc:informix-sqli://hostName:port/
|                   | databaseName:INFORMIXSERVER=servername                  |
| MariaDB           | jdbc:mariadb://hostName:port/databaseName               |
| Microsoft SQL Server | jdbc:sqlserver://hostName:port;databaseName=name         |
| MySQL             | jdbc:mysql://hostName:port/databaseName                  |
| Oracle            | jdbc:oracle:thin:@//hostName:port:service               |
| Oracle XML DB     | jdbc:oracle:oci://hostName:port:service                 |
| PostgreSQL        | jdbc:postgresql://hostName:port/databaseName             |
| Progress OpenEdge | jdbc:datadirect:openedge://host:port;databaseName=db_name |
| Sybase            | jdbc:sybase:Tds:hostName:port/databaseName               |
| Teradata          | jdbc:teradata://databaseServerName                      |

**Note:** Syntax variations to the formats listed above are also possible (for example, the database
URL may exclude the port or may include the username and password to the database). Check the documentation of the database vendor for further details.

7. Click **Connect**.

### 7.2.1.6.1 Configuring the CLASSPATH

The **CLASSPATH** environment variable is used by the Java Runtime Environment (JRE) or the Java Development Kit (JDK) to locate Java classes and other resource files on your operating system. When you connect to a database through JDBC, this variable must be configured to include the path to the JDBC driver on your operating system, and, in some cases, the path to additional
library files specific to the database type you are using.

The following table lists sample file paths that must be typically included in the CLASSPATH variable. Importantly, you may need to adjust this information based on the location of the JDBC driver on your system, the JDBC driver name, as well as the JRE/JDK version present on your operating system. To avoid connectivity problems, check the installation instructions and any pre-installation or post-installation configuration steps applicable to the JDBC driver installed on your operating system.

<table>
<thead>
<tr>
<th>Database</th>
<th>Sample CLASSPATH entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firebird</td>
<td>C:\Program Files\Firebird\Jaybird-2.2.8-JDK_1.8\jaybird-full-2.2.8.jar</td>
</tr>
<tr>
<td>IBM DB2</td>
<td>C:\Program Files (x86)\IBM\SQLLIB\java\db2jcc.jar;C:\Program Files (x86)\IBM\SQLLIB\java\db2jcc_license_cu.jar;</td>
</tr>
<tr>
<td>IBM DB2 for i</td>
<td>C:\jt400\jt400.jar;</td>
</tr>
<tr>
<td>IBM Informix</td>
<td>C:\Informix_JDBC_Driver\lib\ifxjdbc.jar;</td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>C:\Program Files\Microsoft JDBC Driver 4.0 for SQL Server\sqljdbc_4.0\enu\sqljdbc.jar</td>
</tr>
<tr>
<td>MariaDB</td>
<td>&lt;installation directory&gt;\mariadb-java-client-2.2.0.jar</td>
</tr>
<tr>
<td>MySQL</td>
<td>&lt;installation directory&gt;\mysql-connector-java-version-bin.jar;</td>
</tr>
<tr>
<td>Oracle</td>
<td>ORACLE_HOME\jdbc\lib\ojdbc6.jar;</td>
</tr>
<tr>
<td>Oracle (with XML DB)</td>
<td>ORACLE_HOME\jdbc\lib\ojdbc6.jar; ORACLE_HOME\LIB\xmlparserv2.jar; ORACLE_HOME\RDBMS\jlib\xdb.jar;</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>&lt;installation directory&gt;\postgresql.jar</td>
</tr>
<tr>
<td>Progress OpenEdge</td>
<td>%DLC%\java\openedge.jar; %DLC%\java\pool.jar;</td>
</tr>
<tr>
<td></td>
<td>Note: Assuming the Progress OpenEdge SDK is installed on the machine, %DLC% is the directory where OpenEdge is installed.</td>
</tr>
<tr>
<td>Sybase</td>
<td>C:\sybase\jConnect-7_0\classes\jconn4.jar</td>
</tr>
<tr>
<td>Teradata</td>
<td>&lt;installation directory&gt;\tdgssconfig.jar;&lt;installation directory&gt;\terajdbc4.jar</td>
</tr>
</tbody>
</table>

- Changing the CLASSPATH variable may affect the behavior of Java applications on your machine. To understand possible implications before you proceed, refer to the Java documentation.
- Environment variables can be user or system. To change system environment variables, you need administrative rights on the operating system.
- After you change the environment variable, restart any running programs for settings to take effect. Alternatively, log off or restart your operating system.
To configure the CLASSPATH on Windows 7:

1. Open the Start menu and right-click Computer.
2. Click Properties.
3. Click Advanced system settings.
4. In the Advanced tab, click Environment Variables.
5. Locate the CLASSPATH variable under user or system environment variables, and then click Edit. If the CLASSPATH variable does not exist, click New to create it.
6. Edit the variable value to include the path on your operating system where the JDBC driver is located. To separate the JDBC driver path from other paths that may already be in the CLASSPATH variable, use the semi-colon separator ( ; ).

To configure the CLASSPATH on Windows 10:

1. Press the Windows key and start typing "environment variables".
2. Click the suggestion Edit the system environment variables.
3. Click Environment Variables.
4. Locate the CLASSPATH variable under user or system environment variables, and then click Edit. If the CLASSPATH variable does not exist, click New to create it.
5. Edit the variable value to include the path on your operating system where the JDBC driver is located. To separate the JDBC driver path from other paths that may already be in the CLASSPATH variable, use the semi-colon separator ( ; ).

7.2.1.7 Setting up a PostgreSQL Connection

Connections to PostgreSQL databases can be set up either as native connections, or connections via ODBC, JDBC, and other drivers. The advantage of setting up a native connection is that no drivers are required to be installed on your system. Also, if you intend to deploy files for execution on a Linux or macOS server, no drivers are required to be installed on the target server as well.

If you prefer to establish a connection by means of a non-native driver, see the following topics:

- Setting up a JDBC Connection
- Connecting to PostgreSQL (ODBC)

Otherwise, if you want to set up a native connection to PostgreSQL, follow the steps below. To proceed, you need the following prerequisites: host name, port, database name, username, and password.

To set up a native PostgreSQL connection:

1. Start the database connection wizard.
2. Click PostgreSQL Connections.
3. Enter the host (localhost, if PostgreSQL runs on the same machine), port (typically 5432, this is optional), the database name, username, and password in the corresponding boxes.
4. Click **Connect**.

If the PostgreSQL database server is on a different machine, note the following:

- The PostgreSQL database server must be configured to accept connections from clients. Specifically, the `pg_hba.conf` file must be configured to allow non-local connections. Secondly, the `postgresql.conf` file must be configured to listen on specified IP address(es) and port. For more information, check the PostgreSQL documentation [here](https://www.postgresql.org/docs/9.5/static/client-authentication-problems.html).

- The server machine must be configured to accept connections on the designated port (typically, 5432) through the firewall. For example, on a database server running on Windows, a rule may need to be created to allow connections on port 5432 through the firewall, from **Control Panel > Windows Firewall > Advanced Settings > Inbound Rules**.
7.2.1.8 Setting up a SQLite Connection

SQLite (https://www.sqlite.org/index.html) is a file-based, self-contained database type, which makes it ideal in scenarios where portability and ease of configuration is important. Since SQLite databases are natively supported by MapForce, you do not need to install any drivers to connect to them.

SQLite database support notes

- SQLite databases are supported in the MapForce BUILT-IN transformation language (either when you preview the mapping or when you run a MapForce Server execution file).
- SQLite databases are not supported in user-defined functions (UDF).
- On Linux, statement execution timeout for SQLite databases is not supported.
- Full text search tables are not supported
- SQLite allows values of different data types in each row of a given table. In MapForce, all processed values must be compatible with the declared column type; therefore, run-time errors may occur if your SQLite database has row values which are not the same as the declared column type.
- If your mapping should write data to a SQLite database, and if you don't have the target database file already, you will need to create it separately. In this case, you can either create it with a tool such as DatabaseSpy (https://www.altova.com/databasespy) or download the SQLite command-line shell from the official website, and create the database file from the command line (see also Example: Mapping data from XML to SQLite). For complete reference to SQLite command syntax, refer to the official SQLite documentation.

7.2.1.8.1 Connecting to an Existing SQLite Database

To connect to an existing SQLite database:

1. Run the database connection wizard (see Starting the Database Connection Wizard).
2. Select SQLite, and then click Next.
3. Browse for the SQLite database file, or enter the path (either relative or absolute) to the database. The Connect button becomes enabled once you enter the path to a SQLite database file.
4. Click Connect.

7.2.1.9 Using a Connection from Global Resources

Altova Global Resources represent a way to refer to files, folders, or databases so as to make these resources reusable, configurable and available across multiple Altova applications.

If you have already configured a database connection to be available as a global resource, you can reuse the connection at any time (even across different Altova applications).

To use a database connection from Global Resources:

1. Start the database connection wizard.
2. Click Global Resources. Any database connections previously configured as global resources are listed.

3. Select the database connection record, and click Connect.

Tip: To get additional information about each global resource, move the mouse cursor over the global resource.

7.2.1.10 Database Connection Examples

This section includes sample procedures for connecting to a database from MapForce. Note that your Windows machine, the network environment, and the database client or server software is likely to have a configuration that is not exactly the same as the one presented in the following examples.

Note: For most database types, it is possible to connect using more than one data access technology (ADO, ADO.NET, ODBC, JDBC) or driver. The performance of the database connection, as well as its features and limitations will depend on the selected driver, database client software (if applicable), and any additional connectivity parameters that you may have configured outside MapForce.
Connecting to Firebird (ODBC)

This topic provides sample instructions for connecting to a Firebird 2.5.4 database running on a Linux server.

Prerequisites:

- The Firebird database server is configured to accept TCP/IP connections from clients.
- The Firebird ODBC driver must be installed on your operating system. This example uses the Firebird ODBC driver version 2.0.3.154 downloaded from the Firebird website (https://www.firebirdsql.org/).
- The Firebird client must be installed on your operating system. Note that there is no standalone installer available for the Firebird 2.5.4 client; the client is part of the Firebird server installation package. You can download the Firebird server installation package from the Firebird website (https://www.firebirdsql.org/), look for “Windows executable installer for full Superclassic/Classic or Superserver”. To install only the client files, choose “Minimum client install - no server, no tools” when going through the wizard steps.

Important:

- The platform of both the Firebird ODBC driver and client (32-bit or 64-bit) must correspond to that of MapForce.
- The version of the Firebird client must correspond to the version of Firebird server to which you are connecting.

- You have the following database connection details: server host name or IP address, database path (or alias) on the server, user name, and password.

To connect to Firebird via ODBC:

1. Start the database connection wizard.
2. Click ODBC Connections.
3. Select User DSN (or System DSN, if you have administrative privileges), and then click Add.

![Create an ODBC DSN dialog box](Image)
4. Select the Firebird driver, and then click **User DSN** (or **System DSN**, depending on what you selected in the previous step). If the Firebird driver is not available in the list, make sure that it is installed on your operating system (see also [Viewing the Available ODBC Drivers](#)).

![Firebird ODBC Setup](image)

5. Enter the database connection details as follows:

<table>
<thead>
<tr>
<th><strong>Data Source Name (DSN)</strong></th>
<th>Enter a descriptive name for the data source you are creating.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Database</strong></td>
<td>Enter the server host name or IP address, followed by a colon, followed by the database alias (or path). In this example, the host name is <code>firebirdserv</code>, and the database alias is <code>products</code>, as follows:</td>
</tr>
</tbody>
</table>

```
firebirdserv:products
```

Using a database alias assumes that, on the server side, the database administrator has configured the alias `products` to point to the actual Firebird (.fdb) database file on the server (see the Firebird documentation for more details).

You can also use the server IP address instead of the host name, and a path instead of an alias; therefore, any of the
following sample connection strings are valid:

firebirdserver:/var/Firebird/databases/butterflies.fdb
127.0.0.1:D:\Misc\Lenders.fdb

If the database is on the local Windows machine, click **Browse** and select the Firebird (.fdb) database file directly.

<table>
<thead>
<tr>
<th>Client</th>
<th>Enter the path to the fbclient.dll file. By default, this is the bin subdirectory of the Firebird installation directory.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Account</td>
<td>Enter the database user name supplied by the database administrator (in this example, PROD_ADMIN).</td>
</tr>
<tr>
<td>Password</td>
<td>Enter the database password supplied by the database administrator.</td>
</tr>
</tbody>
</table>

6. Click **OK**.

### 7.2.1.10.2 Connecting to Firebird (JDBC)

This topic provides sample instructions for connecting to a Firebird database server through JDBC.

**Prerequisites:**

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**, see Java **Settings**; b) The JVM path found in the Windows registry; c) The JAVA_HOME environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The Firebird JDBC driver must be available on your operating system (it takes the form of a .jar file which provides connectivity to the database). The driver can be downloaded from the Firebird website (https://www.firebirdsql.org/). This example uses Jaybird 2.2.8.
- You have the following database connection details: host, database path or alias, username, and password.

**To connect to Firebird through JDBC:**

1. **Start the database connection wizard.**
2. **Click** **JDBC Connections**.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the required .jar file is located at the following path: C:\jdbc\firebird\jaybird-full-2.2.8.jar. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also **Configuring the CLASSPATH**).
4. In the “Driver” box, select `org.firebirdsql.jdbc.FBDriver`. Note that this entry is available if a valid .jar file path is found either in the “Classpath” text box, or in the operating system’s CLASSPATH environment variable (see the previous step).

5. Enter the username and password to the database in the corresponding text boxes.
6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

7. Click Connect.

7.2.1.10.3 Connecting to IBM DB2 (ODBC)

This topic provides sample instructions for connecting to an IBM DB2 database through ODBC.

**Prerequisites:**

- IBM Data Server Client must be installed and configured on your operating system (this example uses IBM Data Server Client 9.7). For installation instructions, check the documentation supplied with your IBM DB2 software. After installing the IBM Data Server Client, check if the ODBC drivers are available on your machine (see Viewing the Available ODBC Drivers).
- Create a database alias. There are several ways to do this:
  - From IBM DB2 Configuration Assistant
• From IBM DB2 Command Line Processor
• From the ODBC data source wizard (for this case, the instructions are shown below)

You have the following database connection details: host, database, port, username, and password.

To connect to IBM DB2:

1. **Start the database connection wizard** and select **IBM DB2 (ODBC/JDBC)**.
2. Click **Next**.

### JDBC vs. ODBC

- **JDBC**
  
  Java-based connection that may provide support for more modern features of your database that aren't available via ODBC. The tradeoff for these features is a potential sacrifice of performance.

- **ODBC**
  
  An ODBC connection will generally be faster and less resource-intensive than a JDBC connection, but lacks support for more modern database features (such as native XML types).

3. Select **ODBC**, and click **Next**. If prompted to edit the list of known drivers for the database, select the database drivers applicable to IBM DB2 (see **Prerequisites**), and click **Next**.
4. Select the IBM DB2 driver from the list, and then click **Connect**. (To edit the list of available drivers, click **Edit Drivers**, and then check or uncheck the IBM DB2 drivers you wish to add or remove, respectively.)
5. Enter a data source name (in this example, **DB2DSN**), and then click **Add**.

6. On the **Data Source** tab, enter the user name and password to the database.
7. On the **TCP/IP** tab, enter the database name, a name for the alias, the host name and the port number, and then click OK.
8. Enter again the username and password, and then click **OK**.

![Database connection interface](image)

### 7.2.1.10.4 Connecting to IBM DB2 for i (ODBC)

This topic provides sample instructions for connecting to an *IBM DB2 for i* database through ODBC.

**Prerequisites:**

- *IBM System i Access for Windows* must be installed on your operating system (this example uses *IBM System i Access for Windows V6R1M0*). For installation instructions, check the documentation supplied with your *IBM DB2 for i* software. After installation, check if the ODBC driver is available on your machine (see [Viewing the Available ODBC Drivers](#)).
You have the following database connection details: the I.P. address of the database server, database user name, and password.

Run System i Navigator and follow the wizard to create a new connection. When prompted to specify a system, enter the I.P. address of the database server. After creating the connection, it is recommended to verify it (click on the connection, and select File > Diagnostics > Verify Connection). If you get connectivity errors, contact the database server administrator.

To connect to IBM DB2 for i:

1. Start the database connection wizard.
2. Click ODBC connections.
3. Click User DSN (alternatively, click System DSN, or File DSN, in which case the subsequent instructions will be similar).
4. Click Add +.
5. Select the iSeries Access ODBC Driver from the list, and click User DSN (or System DSN, if applicable).
6. Enter a data source name and select the connection from the System combo box. In this example, the data source name is iSeriesDSN and the System is 192.0.2.0.

![Data Source Configuration]

**Note:** When adding an ODBC data source for an IBM DB2 for i database, a default flag is set which enables query timeouts. This setting must be disabled for MapForce to correctly load mapping files. To disable the setting, select the Performance tab, click Advanced, and clear the Allow query timeout check box.

7. Click Connection Options, select Use the User ID specified below and enter the name of the database user (in this example, DBUSER).
8. Click **OK**. The new data source becomes available in the list of DSNs.
9. Click **Connect**.
10. Enter the user name and password to the database when prompted, and then click **OK**.

### 7.2.1.10.5 Connecting to IBM Informix (JDBC)

This topic provides sample instructions for connecting to an IBM Informix database server through JDBC.

**Prerequisites:**

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**, see **Java Settings**; b) The JVM path found in the Windows registry; c) The **JAVA_HOME** environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The JDBC driver (one or several .jar files that provide connectivity to the database) must be available on your operating system. In this example, IBM Informix JDBC driver version 3.70 is used. For the driver's installation instructions, see the documentation accompanying the driver or the "IBM Informix JDBC Driver Programmer's Guide".
- You have the following database connection details: host, name of the Informix server, database, port, username, and password.
To connect to IBM Informix through JDBC:

1. Start the database connection wizard.
2. Click JDBC Connections.
3. Next to “Classpaths”, enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the required .jar file is located at the following path: C:\Informix_JDBC_Driver\lib\ifxjdbc.jar. Note that you can leave the “Classpaths” text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also Configuring the CLASSPATH).
4. In the “Driver” box, select com.informix.jdbc.IfxDriver. Note that this entry is available if a valid .jar file path is found either in the “Classpath” text box, or in the operating system's CLASSPATH environment variable (see the previous step).
5. Enter the username and password to the database in the corresponding text boxes.
6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.
7. Click Connect.
Connecting to MariaDB (ODBC)

This example illustrates how to connect to a MariaDB database server through ODBC.

Prerequisites:

- The MariaDB Connector/ODBC (https://downloads.mariadb.org/connector-odbc/) must be installed.
- You have the following database connection details: host, database, port, username, and password.

To connect to MariaDB through ODBC:

1. Start the database connection wizard.
2. Select MariaDB (ODBC), and then click Next.
3. Select Create a new Data Source Name (DSN) with the driver, and choose MariaDB ODBC 3.0 Driver. If no such driver is available in the list, click Edit Drivers, and select any available MariaDB drivers (the list contains all ODBC drivers installed on your operating system).
4. Click Connect.
5. Enter name and, optionally, a description that will help you identify this ODBC data source in future.

6. Fill in the database connection credentials (TCP/IP Server, User, Password), select a database, and then click **Test DSN**. Upon successful connection, a message box appears:
7. Click Next and complete the wizard. Other parameters may be required, depending on the case (for example, SSL certificates if you are connecting to MariaDB through a secure connection).

Note: If the database server is remote, it must be configured by the server administrator to accept remote connections from your machine's IP address.

7.2.1.10.7 Connecting to Microsoft Access (ADO)

A simple way to connect to a Microsoft Access database is to follow the wizard and browse for the database file, as shown in Connecting to an Existing Microsoft Access Database. An alternative approach is to set up an ADO connection explicitly, as shown in this topic. This approach is useful if your database is password-protected.

It is also possible to connect to Microsoft Access through an ODBC connection, but there are some limitations in this scenario, so it is best to avoid it.

To connect to a password-protected Microsoft Access database:

1. Start the database connection wizard.
2. Click ADO Connections.
3. Click Build.
4. Select the **Microsoft Office 15.0 Access Database Engine OLE DB Provider**, and then click **Next**.
5. In the Data Source box, enter the path to the Microsoft Access file. Because the file is on
the local network share U:\Departments\Finance\Reports\Revenue.accdb, we will
convert it to UNC format, and namely \server1\dfs\Departments\Finance\Reports
\Revenue.accdb, where server1 is the name of the server and dfs is the name of the
network share.
6. On the All tab, double click the Jet OLEDB:Database Password property and enter the
database password as property value.

Note: If you are still unable to connect, locate the workgroup information file (System.MDW)
applicable to your user profile, and set the value of the Jet OLEDB: System database
property to the path of the System.MDW file.
7.2.1.10.8 Connecting to Microsoft SQL Server (ADO)

This example illustrates how to connect to a SQL Server database through ADO. These instructions are applicable when you use a native SQL Server provider such as SQL Server Native Client 11.0 and also when you use the Microsoft OLE DB Provider for SQL Server provider.

To connect to SQL Server:

1. Start the database connection wizard.
2. Select Microsoft SQL Server (ADO), and then click Next. The list of available ADO drivers is displayed. In this example, the SQL Server Native Client 11.0 is used as provider. You can also select the Microsoft OLE DB Provider for SQL Server provider.

3. Click Next. The Data Link Properties dialog box appears.
4. Select or enter the name of the database server, for example, SQLSERV01. If you are connecting to a named SQL Server instance, the server name looks like SQLSERV01\SOMEINSTANCE. To view the list of all servers on the network, expand the drop-down list.

5. If the database server was configured to allow connections from users authenticated on the Windows domain, select Use Windows NT integrated security. Otherwise, select Use a specific user name and password, clear the Blank password check box, and enter the credentials in the relevant boxes.

6. Select the database to which you are connecting (in this example, Nanonull).
7. To test the connection at this time, click **Test Connection**. This is an optional, recommended step.

    Even if the connection test is successful at this time, you must still persist the security info, see the next step.

8. Click the **All** tab, and change the **Persist Security Info** property to **True**.
9. Click **OK**.

### 7.2.1.10.9 Connecting to Microsoft SQL Server (ODBC)

This example illustrates how to connect to a SQL Server database through ODBC.

**To connect to SQL Server using ODBC:**

1. Start the database connection wizard.
2. Click **ODBC Connections**.
3. Select User DSN (or System DSN, if you have administrative privileges), and then click **Add**.
4. Select **SQL Server** (or **SQL Server Native Client**, if available), and then click **User DSN** (or **System DSN** if you are creating a System DSN).

5. Enter a name and description to identify this connection, and then select from the list the SQL Server to which you are connecting (**SQLSERV01** in this example).
6. If the database server was configured to allow connections from users authenticated on the Windows domain, select **With Windows NT authentication**. Otherwise, select **With SQL Server authentication** and type the user name and password in the relevant boxes.

7. Select the name of the database to which you are connecting (in this example,
7.2.1.10.10  Connecting to MySQL (ODBC)

This example illustrates how to connect to a MySQL database server from a Windows machine through the ODBC driver. The MySQL ODBC driver is not available on Windows, so it must be downloaded and installed separately. This example uses MySQL Connector/ODBC 8.0.

Prerequisites:

- MySQL ODBC driver must be installed on your operating system. Check the MySQL documentation for the driver version recommended for your database server version (see https://dev.mysql.com/downloads/connector/odbc/).
- You have the following database connection details: host, database, port, username, and password.

To connect to MySQL via ODBC:

1. Start the database connection wizard.
2. Select MySQL (ODBC), and then click Next.
3. Select Create a new Data Source Name (DSN) with the driver, and select a MySQL driver. If no MySQL driver is available in the list, click Edit Drivers, and select any available MySQL drivers (the list contains all ODBC drivers installed on your operating system).
4. Click Connect.
5. In the Data Source Name box, enter a descriptive name that will help you identify this ODBC data source in future.

6. Fill in the database connection credentials (TCP/IP Server, User, Password), select a database, and then click **OK**.

**Note:** If the database server is remote, it must be configured by the server administrator to accept remote connections from your machine's IP address. Also, if you click **Details>>**, there are several additional parameters available for configuration. Check the driver's documentation before changing their default values.

### 7.2.10.11 Connecting to Oracle (ODBC)

This example illustrates a common scenario where you connect from MapForce to an Oracle database server on a network machine, through an Oracle database client installed on the local operating system.

The example includes instructions for setting up an ODBC data source (DSN) using the database connection wizard in MapForce. If you have already created a DSN, or if you prefer to create it directly from ODBC Data Source administrator in Windows, you can do so, and then select it when prompted by the wizard. For more information about ODBC data sources, see [Setting up an ODBC Connection](#).
Prerequisites:

- The Oracle database client (which includes the ODBC Oracle driver) must be installed and configured on your operating system. For instructions on how to install and configure an Oracle database client, refer to the documentation supplied with your Oracle software.
- The `tnsnames.ora` file located in Oracle home directory contains an entry that describes the database connection parameters, in a format similar to this:

```plaintext
ORCL =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = server01)(PORT = 1521))
    )
  (CONNECT_DATA =
    (SID = orcl)
    (SERVER = DEDICATED)
    )
  )
```

The path to the `tnsnames.ora` file depends on the location where Oracle home directory was installed. For Oracle database client 11.2.0, the default Oracle home directory path could be as follows:

```
C:\app\username\product\11.2.0\client_1\network\admin\tnsnames.ora
```

You can add new entries to the `tnsnames.ora` file either by pasting the connection details and saving the file, or by running the Oracle Net Configuration Assistant wizard (if available).

To connect to Oracle using ODBC:

1. Start the database connection wizard.
2. Select Oracle (ODBC / JDBC), and then click Next.
3. Select **ODBC**.

4. Click **Edit Drivers**.
5. Select the Oracle drivers you wish to use (in this example, Oracle in OraClient11g_home1). The list displays the Oracle drivers available on your system after installation of Oracle client.

6. Click Back.

7. Select Create a new data source name (DSN) with the driver, and then select the Oracle driver chosen in step 4.
Avoid using the Microsoft-supplied driver called **Microsoft ODBC for Oracle** driver. Microsoft recommends using the ODBC driver provided by Oracle (see [http://msdn.microsoft.com/en-us/library/ms714756(v=vs.85).aspx](http://msdn.microsoft.com/en-us/library/ms714756(v=vs.85).aspx)).

8. Click **Connect**.
9. In the Data Source Name text box, enter a name to identify the data source (in this example, **Oracle DSN 1**).

10. In the TNS Service Name box, enter the connection name as it is defined in the `tnsnames.ora` file (see prerequisites). In this example, the connection name is **ORCL**.

11. Click **OK**.

12. Enter the username and password to the database, and then click **OK**.

7.2.1.10.12 Connecting to Oracle (JDBC)

This example shows you how to connect to an Oracle database server from a client machine, using the JDBC interface. The connection is created as a pure Java connection, using the **Oracle Instant Client Package (Basic)** available from the Oracle website. The advantage of this connection type is that it requires only the Java environment and the .jar libraries supplied by the Oracle Instant Client Package, saving you the effort to install and configure a more complex database client.
Prerequisites:

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application Options, see Java Settings; b) The JVM path found in the Windows registry; c) The JAVA_HOME environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The Oracle Instant Client Package (Basic) must be available on your operating system. The package can be downloaded from the official Oracle website. This example uses Oracle Instant Client Package version 12.1.0.2.0, for Windows 32-bit.
- You have the following database connection details: host, port, service name, username, and password.

To connect to Oracle through the Instant Client Package:

1. Start the database connection wizard.
2. Click JDBC Connections.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the required .jar file is located at the following path: C:\jdbc\instantclient_12_1\odbc7.jar. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also Configuring the CLASSPATH).
4. In the "Driver" box, select either oracle.jdbc.OracleDriver or oracle.jdbc.driver.OracleDriver. Note that these entries are available if a valid .jar file path is found either in the "Classpath" text box, or in the operating system's CLASSPATH environment variable (see the previous step).
5. Enter the username and password to the database in the corresponding text boxes.
6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

   jdbc:oracle:thin:@//host:port:service

7. Click Connect.

7.2.1.10.13  Connecting to PostgreSQL (ODBC)

This topic provides sample instructions for connecting to a PostgreSQL database server from a Windows machine through the ODBC driver. The PostgreSQL ODBC driver is not available on Windows, so it must be downloaded and installed separately. This example uses the psqlODBC driver (version 11.0) downloaded from the official website (see also Database Drivers Overview).

Note: You can also connect to a PostgreSQL database server directly (without the ODBC driver), see Setting up a PostgreSQL Connection.

Prerequisites:

- `psqlODBC` driver must be installed on your operating system.
- You have the following database connection details: server, port, database, user name, and password.
To set up a connection to PostgreSQL using ODBC:

1. [Start the database connection wizard.](#)
2. Click [ODBC Connections](#).
3. Select the [User DSN](#) option.
4. Click [Create a new DSN](#) and select the driver from the drop-down list. If no PostgreSQL driver is available in the list, make sure that the PostgreSQL ODBC driver is installed on your operating system, as mentioned in the prerequisites above.

![Create an ODBC DSN](image)

5. Click [User DSN](#).

![PostgreSQL Unicode ODBC Driver (psqIODBC) Setup](image)

6. Fill in the database connection credentials (these must be supplied by the database owner), and then click [Save](#).

The connection is now available in the list of ODBC connections. To connect to the database, you can either double-click the connection or select it, and then click [Connect](#).
7.2.1.10.14 Connecting to Progress OpenEdge (ODBC)

This topic provides sample instructions for connecting to a Progress OpenEdge database server through the Progress OpenEdge 11.6 ODBC driver.

Prerequisites:

- The ODBC Connector for Progress OpenEdge driver must be installed on your operating system. The Progress OpenEdge ODBC driver can be downloaded from the vendor's website (see also Database Drivers Overview). Make sure to download the 32-bit driver when running the 32-bit version of MapForce, and the 64-bit driver when running the 64-bit version. After installation, check if the ODBC driver is available on your machine (see also Viewing the Available ODBC Drivers).
You have the following database connection details: host name, port number, database name, user ID, and password.

Connecting to Progress OpenEdge through ODBC

1. Start the database connection wizard.
2. Click ODBC Connections.
3. Click User DSN (alternatively, click System DSN, or File DSN, in which case the subsequent instructions will be similar).
4. Click Add.
5. Select the Progress OpenEdge Driver from the list, and click User DSN (or System DSN, if applicable).
6. Fill in the database connection credentials (Database, Server, Port, User Name, password).
Password), and then click **OK**. To verify connectivity before saving the entered data, click **Test Connect**.

7. Click **OK**. The new data source now appears in the list of ODBC data sources.
8. Click **Connect**.

### 7.2.1.10.15 Connecting to Progress OpenEdge (JDBC)

This topic provides sample instructions for connecting to a Progress OpenEdge 11.6 database server through JDBC.

**Prerequisites**

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**, see [Java Settings](#); b) The JVM path found in the Windows registry; c) The **JAVA_HOME** environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The operating system’s **PATH** environment variable must include the path to the `bin` directory of the JRE or JDK installation directory, for example `C:\Program Files (x86)\Java\jre1.8.0_51\bin`.
- The Progress OpenEdge JDBC driver must be available on your operating system. In this example, JDBC connectivity is provided by the `openedge.jar` and `pool.jar` driver component files available in `C:\Progress\OpenEdge\java` as part of the OpenEdge SDK installation.
- You have the following database connection details: host, port, database name, username, and password.

Connecting to OpenEdge through JDBC

1. Start the database connection wizard.
2. Click JDBC Connections.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the required .jar file paths are: `C:\Progress\OpenEdge\java\openedge.jar;C:\Progress\OpenEdge\java\pool.jar`. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also Configuring the CLASSPATH).
4. In the “Driver” box, select `com.ddtek.jdbc.openedge.OpenEdgeDriver`. Note that this entry is available if a valid .jar file path is found either in the "Classpath" text box, or in the operating system’s CLASSPATH environment variable (see the previous step).

![Classpath, Driver, Username, Password, Database URL fields]

5. Enter the username and password to the database in the corresponding text boxes.
6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

```
jdbc:datadirect:openedge://host:port;databaseName=db_name
```

7. Click Connect.
7.2.1.10.16  Connecting to Sybase (JDBC)

This topic provides sample instructions for connecting to a Sybase database server through JDBC.

Prerequisites:

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application Options, see Java Settings; b) The JVM path found in the Windows registry; c) The JAVA_HOME environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- Sybase jConnect component must be installed on your operating system (in this example, jConnect 7.0 is used, installed as part of the Sybase Adaptive Server Enterprise PC Client installation). For the installation instructions of the database client, refer to Sybase documentation.
- You have the following database connection details: host, port, database name, username, and password.

To connect to Sybase through JDBC:

1. Start the database connection wizard.
2. Click JDBC Connections.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the required .jar file path is: C:\sybase\jConnect-7_0\classes\jconn4.jar. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also Configuring the CLASSPATH).
4. In the "Driver" box, select com.sybase.jdbc4.jdbc.SybDriver. Note that this entry is available if a valid .jar file path is found either in the "Classpath" text box, or in the operating system's CLASSPATH environment variable (see the previous step).
5. Enter the username and password to the database in the corresponding text boxes.
6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

```
jdbc:sysbse:Tds:hostName:port/databaseName
```

7. Click Connect.

### 7.2.1.10.17 Connecting to Teradata (ODBC)

This example illustrates how to connect to a Teradata database server through ODBC.

**Prerequisites:**

- The Teradata ODBC driver must be installed (see [https://downloads.teradata.com/download/connectivity/odbc-driver/windows](https://downloads.teradata.com/download/connectivity/odbc-driver/windows). This example uses Teradata ODBC Driver for Windows version 16.20.00.
- You have the following database connection details: host, username, and password.

**To connect to Teradata through ODBC:**

1. Press the **Windows** key, start typing "ODBC", and select **Set up ODBC data sources**
(32-bit) from the list of suggestions. If you have a 64-bit ODBC driver, select **Set up ODBC data sources (64-bit)** and use 64-bit MapForce in the subsequent steps.

2. Click the **System DSN** tab, and then click **Add**.

3. Select **Teradata Database ODBC Driver** and click **Finish**.
4. Enter name and, optionally, a description that will help you identify this ODBC data source in future. Also, enter the database connection credentials (Database server, User, Password), and, optionally, select a database.

5. Click **OK**. The data source now appears in the list.
6. Run MapForce and start the database connection wizard.
7. Click ODBC Connections.
8. Click **System DSN**, select the data source created previously, and then click **Connect**.

Note: If you get the following error: "The driver returned invalid (or failed to return) SQL_DRIVER_ODBC_VER: 03.80", make sure that the path to the ODBC client (for example, **C:\Program Files\Teradata\Client\16.10\bin**, if you installed it to this location) exists in your system's PATH environment variable. If this path is missing, add it manually.

### 7.2.10.18 Connecting to Teradata (JDBC)

This example illustrates how to connect to a Teradata database server through JDBC.

**Prerequisites:**

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in
this order: a) The custom JVM path you may have set in application Options, see Java Settings; b) The JVM path found in the Windows registry; c) The JAVA_HOME environment variable.

- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The JDBC driver (one or more .jar files that provide connectivity to the database) must be available on your operating system. In this example, Teradata JDBC Driver 16.20.00.02 is used. For more information, see https://downloads.teradata.com/download/connectivity/jdbc-driver.
- You have the following database connection details: host, database, port, username, and password.

To connect to Teradata through JDBC:

1. Start the database connection wizard.
2. Click JDBC Connections.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the .jar files are located at the following path: C:\jdbc\teradata\. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also Configuring the CLASSPATH).
4. In the “Driver” box, select com.teradata.jdbc.TeraDriver. Note that this entry is available if a valid .jar file path is found either in the "Classpath" text box, or in the operating system's CLASSPATH environment variable (see the previous step).
5. Enter the username and password to the database in the corresponding text boxes.

6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted value with the one applicable to your database server.

   jdbc:teradata://databaseServerName

7. Click Connect.

### 7.2.2 Introduction to Database Mappings

This section is an introduction to working with databases in MapForce. It shows you how to work with a database after the connection to it is successfully established (see Connecting to a Database). This includes selecting the database objects that you want to appear on the mapping, handling database relationships, and configuring the database settings applicable to the mapping process. Examples of how to achieve specific goals when mapping data to or from database components are also included.
7.2.2.1 Adding Databases to the Mapping

Before adding a database to the mapping, make sure to select a transformation language where database mappings are supported. This can be either the BUILT-IN transformation language, or any of the following languages: C++, C#, Java (see also Selecting a Transformation Language). Note that, if you intend to deploy the mapping to FlowForce Server or execute it with MapForce Server, or use features such as Bulk Transfer and stored procedures, BUILT-IN must be selected as transformation language.

Once the desired transformation language is selected, you can add a database to the mapping in one of the following ways:

- On the Insert menu, click Database.
- Click the Insert Database ( toolbar button.

When you take any of these actions, a database connection wizard appears, guiding you through the steps required to connect to the database.
Note: In some advanced scenarios, databases can also be added to the mapping as variables (see Using Variables). When you choose to add a database structure as a variable, the same database connection wizard mentioned above appears.

For instructions on how to proceed with this wizard so as to set up a connection to any of the databases supported by MapForce, see Connecting to a Database.

Once the database connection is successfully established, you are prompted to select the database objects that should appear on the mapping. See Adding, Editing, and Removing.
Database Objects for further information.

7.2.2.2 Example: Adding the "altova.mdb" Database to the Mapping

This example shows you how to add a sample Microsoft Access database to a mapping. The sample database is called altova.mdb and can be found in the <Documents>\Altova\MapForce2019\MapForceExamples folder. The altova.mdb database supports various database-related actions and concepts described in this documentation.

To add the altova.mdb database to the mapping:

1. On the Insert menu, click Database. Alternatively, click the Insert Database ( ) toolbar button.

2. Click Microsoft Access (ADO), and then click Next.
3. Browse for the `altova.mdb` file available in the `<Documents>\Altova\MapForce2019 \MapForceExamples` folder, and then click Connect.
4. When prompted to select the database objects, select User Tables.
7.2.2.3 Adding, Editing, and Removing Database Objects

Some databases can have a large number of objects (such as schemas, tables, views, and so on). This topic shows you how to get on the mapping only those database objects that are required for mapping purposes. Below, we will be using a sample Access database; the instructions are similar for other database types.

1. On the Insert menu, click Database.
2. Click Connection Wizard, and then click Microsoft Access (ADO).
3. Click Next, and browse for the altova.mdb available in the <Documents>\Altova\MapForce2019\MapForceExamples\ folder.

A dialog box appears, enabling you to select the database objects that you want to be included into the mapping.

To include a database object (for example, a table) it in the mapping, click the check box next to it. For the purpose of this example, click the check box next to User Tables.

The Object Locator button ( ) allows you to find specific database items. Select a particular
object (or type its name) in the combo box which appears in the lower area of dialog box.

The **Filter** button (🔍) allows you to filter objects by name. Once you click the Filter button, a filter icon is available next to objects which supports filtering (in this example, “Tables”).

Click the filter icon to choose whether the object name should begin with, end with, be equal with, or contain the search text.
Now you can enter the search text next to the filter (in this example, "A"):

The **Show checked objects only** button (✓) displays those items where a check box is active.

The **Add/Edit SELECT Statement** button enables you to add or edit custom SELECT statements for the current database. The data returned by such statements then becomes available as mapping source. For more information, see [SQL SELECT Statements as Virtual Tables](#).

The **Add/Edit Relations** button enables you to define local primary and foreign key relationships between fields in the database, in addition to those that may already be present. For more information, see [Defining Local Relationships](#).

The **Add/Edit Recordset Structures** button applies for databases that support stored procedures. It is only enabled if a stored procedure is currently selected from the database tree. For more information, see [Stored Procedures](#).

The **Show Preview** button enables you to quickly preview the data of the currently selected table.
or view. Note that you can also browse and query a database independently of the mapping process, by using the Database Browser. For more information, see Browsing and Querying Databases.

When you are ready to add the database objects to the mapping, click OK. Only the selected tables, views, etc. will appear on the database component, and you can draw mapping connections to or from them in the standard MapForce way.

To change at any time the database objects, right-click the component, and select Add/Remove/Edit Database Objects.

7.2.2.4 Handling Database Relationships

Relational databases, as their name implies, normally have relationships defined between their tables. Taking as example the altova.mdb database found in the folder <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial, several relationships exist in it, for example:

- The sample company (corresponding to the table "Altova") consists of one or more offices (for example, in Brenton and in Vereno). In database terminology, there is a one-to-many
relationship between the "Altova" and "Office" tables. In other words, for each **PrimaryKey** record in "Altova" table there can be multiple **ForeignKey** records in the "Office" table. Any "Office" record where **ForeignKey** value corresponds to **PrimaryKey** value in "Altova" should therefore be considered an office of "Altova".

- Each office consists of one or several departments (for example, "Marketing", "IT", "Development"). Again, there is a one-to-many relationship between "Office" and "Department" tables.
- Finally, each department consists of one or several people. Hence, the one-to-many relationship between the "Department" and "Person" tables.

Table relationships in altova.mdb database (Microsoft Access "Relationships" view)

Relationships between database tables are important for mapping purposes. MapForce keeps track of such database relationships when you add a database to the mapping. This enables you to preserve the database relationships when mapping data to or from a database. To understand this concept better, add the **altova.mdb** database to the mapping (using the **Insert | Database** menu command). Let us call each of the tables below a "root" table:
"Root" tables

Expanding a "root" table displays all related tables beneath it in a tree view. For example, if you expand the **Office** table, notice how the related table hierarchy is displayed:

- A left arrow (←) in front of a table denotes that this is a child table. For example, **Address** is a child of **Office**. **Department** is also a child of **Office**, as well as a "sibling" table of **Address**, so both have the same indentation level. As you can see, the relationship on the mapping corresponds to the "Relationships" diagram above.
- A right arrow (→) in front of a table denotes a parent table. For example, **Altova** is a parent of **Office**.
Tables relationships in MapForce (altova.mdb database)

This hierarchical representation of tables helps you preserve the existing database relationships when your mapping reads from or writes to a database. For example, let's assume you want to get all the records from the Person table into an XML file, grouped by their department. Specifically, your XML file should link every person to a department, similar to the altova.mdb database used in this example:
As illustrated above, the “Administration” department has three people, “Marketing” has two people, “Engineering” has six people, etc.

When mapping data from this database, if you want every person to be distributed to the correct department, it is important that you use Department as "root" table, and then map from the Person table which is child of Department:
The mapping above is a modified DB_Altova_Hierarchical.mfd from the <Documents>\Altova \MapForce2019\MapForceExamples folder. When you preview the mapping, the result is that each person is grouped by department, which was the intended behaviour. That is, "Administration" has three people, "Marketing" has two people, "Engineering" has six people, etc.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Altova xmlns:soap十二条SchemaLocation="file:///C:...">
  <Office>
    <Department>
      <Name>Administration</Name>
      <Person>...</Person>
      <Person>...</Person>
      <Person>...</Person>
    </Department>
    <Department>
      <Name>Marketing</Name>
      <Person>...</Person>
      <Person>...</Person>
      <Person>...</Person>
    </Department>
    <Department>
      <Name>Engineering</Name>
      <Person>...</Person>
      <Person>...</Person>
      <Person>...</Person>
      <Person>...</Person>
      <Person>...</Person>
      <Person>...</Person>
    </Department>
  </Office>
</Altova>
```

Now have a look at the slightly modified mapping below, where connections have been
deliberately drawn so that both Department and Person are "root" tables.

This time, when you preview the mapping, all persons (regardless of their source department) are grouped under each target department, which was not the intended behaviour. That is, "Administration" has 21 people, "Marketing" has 21 people, "Engineering" has 21 people, etc.
In the second example, the database relationships are disregarded, due to the way the connections were made.

Therefore, when you want to preserve database relationships, make sure that connections are drawn to or from the same “root” table, which contains the child tables whose relationships you want to preserve. This works in the same way for both source and target databases. For examples of database mappings which preserve relationships, see the DB_Altova_Hierarchical.mfd and Altova_Hierarchical_DB.mfd files available in the <Documents>\Altova\MapForce2019\MapForceExamples\ folder (see also Inserting Data into Multiple Linked Tables).

There might also be cases when you do not want to preserve database relationships. For example, let’s assume that you want to export all data from the altova.mdb database to a flat XML file adhering to the SQL/XML specification (Part 14 of the Structured Query Language (SQL) specification). This kind of mapping is illustrated by the DB_Altova_SQLXML.mfd sample, available in the <Documents>\Altova\MapForce2019\MapForceExamples\ folder. The goal of the mapping is to get database data as flat XML file. The target SQL/XML schema was generated with XMLSpy, using the Convert | Create XML Schema from DB Structure menu command.
As illustrated above, every database table has a corresponding element in the target XML. When you preview the mapping result, you can see that the actual database rows from each table are written to "row" elements in the target.
As the XML output shows, no hierarchies exist between the XML elements; it is a flat SQL/XML structure. The database relationships were ignored, because we intentionally mapped data from multiple "root" tables.
7.2.2.5  Defining Local Relationships

When database tables do not have explicitly defined relationships between them, you can define such relationships locally in MapForce. In particular, you can create, from MapForce, primary and foreign key relationships between columns of different tables, without affecting the database in any way. Any database columns can be used as primary or foreign keys. Also, new relations can be created, in addition to those existing in the database. Locally defined relationships are saved together with the mapping.

These “on-the-fly” relationships are called Local Relations in MapForce. The following table lists all the possible fields between which you can define local relations. Mixed relationships are possible (for example, mapping the output of a stored procedure to a database column). The fields taking part in the relationship must have the same, or a compatible, data type.

<table>
<thead>
<tr>
<th>Primary/unique key</th>
<th>Foreign key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column of a database table or view</td>
<td>Column of a database table or view</td>
</tr>
<tr>
<td>Output parameter or return value of a stored procedure, see also Stored Procedures</td>
<td>Input parameter of a stored procedure</td>
</tr>
<tr>
<td>Column of a recordset returned by a stored procedure*</td>
<td>Input parameter of a user-defined SELECT statement</td>
</tr>
<tr>
<td>Column of a user-defined SELECT statement, see also SQL SELECT Statements as Virtual Tables.</td>
<td></td>
</tr>
</tbody>
</table>

* Applicable if the stored procedure is called either as data source (without parameters) or as a function (with input and output parameters). In order for the recordset to become available for selection, you must execute the stored procedure once, to retrieve the recordset.

The altova-no-relation.mdb database used in this example is a simplified version of the altova.mdb database supplied with MapForce. The “Person” and “Address” tables, as well as all table relationships have been removed in Microsoft Access. As illustrated below, none of the tables visible in the altova-no-relation tree have any child tables; all tables are on the same “root” level. The content of each table is limited to the fields it contains.
The aim of the example is to display the offices of “Altova” and show the departments in each office. Note that, in the altova-no-relation.mdb, the primary and foreign key relationships do not exist explicitly, as mentioned above. They exist only logically (implicitly), so we will be re-creating them locally in MapForce to achieve the goal of the mapping.

Local relations can be defined while adding a database to the mapping, or by right-clicking an existing database component and selecting Add/Remove/Edit Database Objects from the context menu, as illustrated in the steps below.

1. On the Insert menu, click Database.
2. In the connection wizard, click Microsoft Access (ADO), and then click Next.
3. Browse for the altova-no-relation.mdb database available in the <Documents>\Altova \MapForce2019\MapForceExamples\Tutorial\ folder, and click Connect.
4. Select the User Tables check box.
5. Click the **Add/Edit Relations** button in the icon bar.

6. The "Add/Edit Table Relations" dialog box opens. Click **Add Relation**.

7. Select values from the two drop-down lists so as to create a primary and foreign key relationship between the "Altova" and "Office" tables, as illustrated below. The two drop-down lists allow you to select the tables or database objects you want to create relations for. The left list specifies the object which stores the primary/unique key, while the right one specifies the foreign key object. The Primary/Unique Key object will be the parent object in MapForce, and the Foreign Key object will be shown as child in the database component (see also [Handling Database Relationships](#)).
8. Click OK to complete the local relation definition, and then click the Insert button to insert the database into the mapping area.

At this stage, you have created a local relationship between the PrimaryKey column of the "Altova" table and the ForeignKey column of the "Office" table. As illustrated below, the "Altova" root table is now a parent to the "Office" table. Namely, the Office table is shown as a related table below the Altova table with its own expand icon.

However, the mapping goal is not yet complete. To complete the mapping goal, use the same method to create a relationship between the Office and Department tables, as shown below.
To open again the "Add/Edit Relations" dialog box, right-click the database component, and select Add/Remove/Edit Database Objects from the context menu.

Finally, add the target schema to the mapping as follows:

2. Browse for the Altova_Hierarchical.xsd file available in the <Documents>\Altova\MapForce2019\MapForceExamples folder. When prompted to supply a sample XML file, click Skip. When prompted to select a root element, select "Altova".

Notice that, in order to preserve relationships between tables in the target XML, all connections were drawn from the same "root" table, hierarchically (in this case, "Altova"). For more information, see Handling Database Relationships.
Having defined the mapping as shown above, click the **Output** tab, to preview the result. The mapping result shows the department elements nested under each respective office, which was the intended goal of this mapping.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Altova xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="file:///C:/Altova_Hierarchical.xsd">
    <Office>
        <Name>Nanonull, Inc.</Name>
        <Department>
            <Name>Administration</Name>
        </Department>
        <Department>
            <Name>Marketing</Name>
        </Department>
        <Department>
            <Name>Engineering</Name>
        </Department>
        <Department>
            <Name>IT & Technical Support</Name>
        </Department>
    </Office>
    <Office>
        <Name>Nanonull Partners, Inc.</Name>
        <Department>
            <Name>Administration</Name>
        </Department>
        <Department>
            <Name>Marketing</Name>
        </Department>
        <Department>
            <Name>IT & Technical Support</Name>
        </Department>
    </Office>
</Altova>
```
7.2.2.6 Executing Mappings Which Modify Databases

When a mapping modifies database data in any way (for example, by inserting, updating, or deleting records), the changes are applied to the database by the engine that executes the mapping. The engine that executes the mapping can be MapForce, MapForce Server (both standalone or under FlowForce Server management), or the execution environment of the code generated for C++, C#, or Java.

When you preview the mapping result directly in MapForce (by clicking the Output tab), an update script is displayed. The script is not actually executed against the database until you take this action explicitly; it is available for preview only.

```
/*
 * The following SQL statements are only for preview and may not be executed in another SQL query tool.
 * To execute these statements use function "Run SQL Script" from menu "Output".
 */

Connect to database using the following connection-string:

Data Source=C:/Users/Altova/Documents/Altova/MapForce2019/MapForceExamples/AltovaTarget.mdb Provider=Microsoft.Jet.OLEDB.4.0

-- begin transaction

DELETE FROM [Address]
DELETE FROM [Person]
DELETE FROM [Department]
DELETE FROM [Office]
DELETE FROM [Altova]

SELECT MAX([Altova].[PrimaryKey]) + 1 AS [PrimaryKey] FROM [Altova]

-->> %PrimaryKey1%

INSERT INTO [Altova] ([Name], [PrimaryKey]) VALUES ('Organization Chart', %PrimaryKey1%)
```

Output preview of a mapping which modifies a database (Altova_Hierarchical_DB.mfd)

The script shows pseudo-SQL statements just for information purpose; it does not directly reflect the statements that are to be executed. For example, if multiple actions are defined against a table (such as "Update if... Insert Rest"), only the first action is shown in the preview, since the second action is executed conditionally.

The pseudo-SQL script must not be manually applied to the database using SQL tools other than the execution engines mentioned above. The script may contain formatting of values not "understood" by external SQL editors.

If you want to apply the mapping changes to the database directly from MapForce, click the Run SQL-Script command available in the Output menu. Remember that this action will actually modify the database with immediate effect.

When the mapping is executed with MapForce Server (both standalone or under FlowForce Server management), the changes to the database are applied with immediate effect. The same happens in the generated code: the database changes are applied when you compile and run the code (for example, by clicking the Run command in Visual Studio).
Your MapForce installation includes several example databases (Microsoft Access or SQLite files) available in the `<Documents>\Altova\MapForce2019\MapForceExamples` folder. It is advisable not to apply database changes from MapForce, using the Run SQL-Script command, against any of the example databases supplied with MapForce; this may render the examples unusable. A simple way to avoid overriding original data is to back up the entire `<Documents>\Altova\MapForce2019\MapForceExamples` folder before updating any files in it.

For information about running mappings in execution environments other than MapForce, see:

- Deploying Mappings to FlowForce Server
- Compiling Mappings to MapForce Server Execution Files
- Code Generator

### 7.2.2.7 Replacing Special Characters

When transforming database data, you might need to remove specific special characters, such as the carriage return/line feed (CR/LF) characters, from the data source. This can be done with the help of the MapForce library function `char-from-code`.

Consider a Microsoft Access database consisting of a table "Lines" which has two columns: "ID" and "Description".

![Diagram of Microsoft Access database](image)

The goal is to extract each description to a CSV file (one description per line); therefore, a mapping to achieve this goal could look as follows:

![Diagram of MapForce mapping](image)

However, because each "Description" row in Access contains multiple lines separated by CR/LF characters, the mapping output includes line breaks also, which is not the intended result:
To overcome this problem, we are going to add to the mapping the `char-from-code` and `replace` functions from the MapForce built-in library (see also Working with Functions). Every description must be processed so that, whenever the characters above are encountered, they should be replaced by a space character.

In the Unicode chart (http://www.unicode.org/charts/), the LF and CR characters correspond to `hex 0A | dec 10` and `hex 0D | dec 13` characters, respectively. Therefore, the mapping has to be modified to convert the decimal Unicode values 13 and 10 to a string, so as to allow further processing by the `replace` function.

If you preview the mapping now, notice that the CR/LF characters within each database field have been replaced by a space.

```
"This is our new company policy."
"It will be implemented immediately."
```

### 7.2.2.8 Handling Null Values

To check at mapping runtime whether a database field is null, use the `is-null` and `is-not-null` MapForce library functions. To see from MapForce if a table has null fields, query it using the Database Browser (see Browsing and Querying Databases).

To set a database field to null, use the `set-null` function.

To replace null database values with a string, use the `substitute-null` function. A sample mapping that illustrates this is `DB_ApplicationList.mfd` available in the `<Documents>\Altova`
For information about handling NULL value comparisons in mappings which update databases, see Handling Nulls in Database Table Actions.

For information about handling nulls when mapping database to or from XML documents, see Nil Values / Nillable.

### 7.2.2.9 Generating Sequential and Unique Values

When inserting data or updating a database, sometimes you might need to create “on-the-fly” sequential or unique values for those database fields which do not have any input from the mapped source. For such cases, use the following built-in MapForce library functions:

- **auto-number** (available in the "core | generator functions" library). This function is generally used to generate primary key values for a numeric field.
- **create-guid** (available in the "lang | generator functions" library). This function creates a globally-unique identifier (as a hex-encoded string) for the specific field.

Note that values for database fields can also be written using database-generated values. This option is available on the Database Table actions dialog box (see Database Table Actions Settings) and is particularly useful when generating primary keys.

### 7.2.2.10 SQL Auto-Completion Suggestions

When you type SQL statements in certain contexts, MapForce may suggest text entries automatically. Auto-completion is available in the following contexts:

- SQL Editor (see Browsing and Querying Databases)
- "Custom SQL" text box in the "Database Table Actions" dialog box (see Database Table Actions Settings)
- "Enter a SQL SELECT statement" dialog box (see Creating SELECT Statements)

Auto-completion

Use the Up and Down keyboard keys to navigate through the list of suggestions. To pick a suggested entry, click it or press Enter.

To disable auto-completion suggestions:

1. On the Tools menu, click Options (or press Ctrl+Alt+O).
2. Under Database, click SQL Editor.
3. Under Entry Helpers, clear the Automatically open check box.
To invoke auto-completion suggestions manually as and when required:

- Press Ctrl+Space.

7.2.2.11 Database Component Settings

After you add a database component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component settings dialog box in one of the following ways:

- Select the component, and then, on the Component menu, click Properties.
- Double-click the component header.
- Right-click the component, and then click Properties.

The available settings are as follows.

Database

This group displays database connection information. Click Change to select a different database, or to redefine the database objects in the existing database component. Connectors to tables of the same name will be retained. You can also change the tables in the component, by right clicking a database component and selecting Add/Remove/Edit Database Objects.

| Data Source | Specifies the name of the current data source. For file-based databases, this can be a path on the file system. |
Use this setting to determine whether a file-based database was added to the mapping using an absolute or relative path. In case of relative paths, “Data Source” contains a path; in case of absolute paths, it contains just the database filename.

**Connection Name**
- Specifies the name of the connection. This name is generated automatically by MapForce. Typically it is the same as the data source name, but it may also be a so-called “alias” name if you are connecting with Altova Global Resources. If there are multiple database components with the same connection name on the mapping, then the name will take the form “<connection1>”, “<connection2>”, etc.

**Database Kind**
- Specifies the kind of the database.

**Connection String**
- Displays the current database connection string. This read-only field is generated based on the information you supply when creating or changing the database connection.

**Login Settings**
The login settings are used for all code generation targets and the built-in execution engine.

| **User** | Enables you to change the user name for connecting to the database. Mandatory if the database requires a user name to connect. |
| **Password** | Enables you to change the password for connecting to the database. Mandatory if the database requires a password to connect. |

**JDBC-Specific Settings**
These settings are used to connect to the database if the mapping contains a JDBC connection and is executed by generated Java code or by MapForce Server.

- **ADO, ADO.NET, and ODBC connections are converted to JDBC (and the JDBC settings below apply) when the mapping is run on a Linux or macOS machine, see Database mappings in various execution environments.**

| **JDBC Driver** | Displays the currently active driver for the database component. The default driver is automatically entered when you define the database component. You can change the driver entered here to suit your needs. Make sure that the syntax of the entry in the Database URL field conforms to the specific driver you choose. |
| **Database URL** | URL of the currently selected database. Make sure that this entry conforms to the JDBC driver syntax, of the specific driver entered in the JDBC-driver field. |
ADO/OLEDB-Specific Settings
These settings are used to connect to the database if the mapping contains an ADO connection and it is executed by generated C# or C++ code, or by MapForce Server running on Windows, see Database mappings in various execution environments. The Data Source and Catalog settings are not used by the built-in execution engine.

<table>
<thead>
<tr>
<th>Data Source</th>
<th>Displays the name of the ADO data source.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catalog</td>
<td>Displays the name of the ADO catalog.</td>
</tr>
<tr>
<td>Provider</td>
<td>Displays the currently active provider for the database component.</td>
</tr>
<tr>
<td>Add. Options</td>
<td>Displays any additional database options.</td>
</tr>
</tbody>
</table>

Generation settings
Generation settings apply to all code generation targets as well as the built-in execution engine.

| Strip schema names from table names | Allows you to strip database schema names from generated code, only retaining the table names for added flexibility. Note that this option works only for SQL Select statements generated by MapForce. User-defined SQL-Statements, when creating virtual tables, will not be modified. |

Timeout for statement execution
When a database is used as a target component, execution timeouts can occur due to server availability, traffic, long-running triggers, and other factors. This setting allows you to define how long the timeout period can be before the database connection is closed. The setting takes effect when querying database data as well as in generated C#, Java, and C++ code.

<table>
<thead>
<tr>
<th>Timeout</th>
<th>Defines the time period, in seconds, that the execution engine must wait for a database response before aborting the execution of the database statement. The default setting for the execution timeout is 60 seconds.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infinite</td>
<td>When enabled, this option instructs the execution engine to never time out.</td>
</tr>
</tbody>
</table>

Note: Timeout for statement execution is not applicable to SQLite databases.

Database transaction handling

| Use transactions | Enables transaction processing when using a database as a target. Transaction processing is enabled for all tables of the database component when you select this option. For more |

Traces

<table>
<thead>
<tr>
<th>Trace level</th>
<th>Applicable for target database components. When tracing is enabled, the actions performed by the mapping against the database are logged in a trace file. You can choose to log all actions, only errors, or disable tracing completely. For more information, see Database Tracing and Error Logging.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace file</td>
<td>Specifies the file to which database trace information will be written when the mapping runs. The trace file is in XML format, see Trace File Structure. This path can be either absolute or relative, and is influenced by the Save all file paths relative to MFD file checkbox.</td>
</tr>
</tbody>
</table>

Save all file paths relative to MFD file

When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. Use relative paths if you intend to run the mapping with MapForce Server on a different operating system. See also Using Relative Paths on a Component.

7.2.3 Mapping Data to Databases

This section provides instructions and examples for transferring data from any mapping source supported by MapForce (for example, an XML file) to a target database. Use the following roadmap for a summary of available options.

<table>
<thead>
<tr>
<th>I want to...</th>
<th>Read this topic...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert data into a target database table based on data supplied by the mapping...</td>
<td>• Inserting Data into a Table</td>
</tr>
</tbody>
</table>
| Control how primary key values are to be created... | • Inserting Data into a Table  
• Inserting Data into Multiple Linked Tables |
| Run a "preliminary" SQL statement to be executed before a table is modified by the mapping (for example, delete all records in the table, or a custom SQL statement)... | • Inserting Data into Multiple Linked Tables  
• Database Table Actions Settings |
| Preserve the hierarchical relationship of records in tables linked by foreign keys... | • Inserting Data into Multiple Linked Tables |
| Update a table conditionally... | • Updating a Table |
| Merge records into a database table (update some records, and also insert other records into the same table), based on a condition... | • "Update if... Insert Rest" Action  
• MERGE statements |
I want to... | Read this topic...
---|---
Preserve database integrity when updating tables that are linked to other tables through foreign key relationships... | • Options for Child Tables When Updating a Parent Table
Define multiple actions against the same table (for example, delete a record if a condition is satisfied, otherwise insert a new record)... | • "Delete if..." Action
• "Ignore if..." Action
Preserve data integrity in case of failed mapping execution... | • Handling Database Exceptions
Insert multiple records into a database table in bulk (combine multiple INSERT statements in one query)... | • Bulk Inserts (MapForce Server)
Avoid undesired results when mapping data to target database tables that contain null values... | • Handling Nulls in Database Table Actions
View an example of how to create a mapping which updates a database. | • Example: Mapping Data from XML to SQLite

7.2.3.1 Inserting Data into a Table

A mapping can insert data into a database table from any of the source components supported by MapForce, including other databases. You can flexibly configure how the primary key of newly inserted records should be created. For example, the primary key can be taken from the mapping, generated by the database, or calculated based on existing key values in the database table.

This example shows you how to insert new records into an existing database table from an XML file. You will also configure how the primary key is to be generated. The example uses the following files:

- altova-cmpy.xml — contains the source data to be inserted into the database.
- Altova_Hierarchical.xsd — the schema used to validate the instance file above.
- altova.mdb — the target database to be updated.

All files are available in the `<Documents>\Altova\MapForce2019\MapForceExamples` folder. Below, the complete path to them will be omitted, for simplicity.

The mapping in this example modifies a sample database file. It is strongly recommended to back up the original database and start with a new copy before following the steps below. This ensures that the original examples are not overridden and that you get the same results as below. For more information, see Executing Mappings Which Modify Databases.

The goal of the mapping is to insert companies found in the altova-cmpy.xml as new records in the "Altova" table of the altova.mdb database. If you open the source XML file, you will notice that it contains only one company, called "Microtech OrgChart". Therefore, the mapping must add a new record to the "Altova" table with the name "Microtech OrgChart". Also, a new primary key
must be generated for it.

To achieve the mapping goal, we will take the steps below.

**Step 1: Insert the source XML component**
- On the Insert menu, click XML Schema/File, and browse for Altova_Hierarchical.xsd. When prompted to supply an instance file, browse for Altova-cmpy.xml.

**Step 2: Insert the target database**
- On the Insert menu, click Database, and go through the wizard steps to connect to altova.mdb (see Example: Adding the "altova.mdb" Database to the Mapping).

**Step 3: Draw the connections**
- Draw the mapping connections as shown below.

![Diagram of mapping connections]

**Note:** If unwanted connections are automatically drawn for descending items, the option "Auto-connect children" is active. In this case, to undo the last action, select the menu option Edit | Undo. To disable the auto-connect option, select the menu option Connection | Auto-connect matching children.

**Step 4: Configure the Insert action**
1. On the target component, notice the Action: Insert (Alt) button. This button appears for each table that has a connection from the mapping (in this case, the "Altova" table). Click this button to configure in more detail the database action to be executed (in this case, the insert action). The Database Table Actions dialog box appears.
2. In the Database Table Actions dialog box, under Insert All, next to PrimaryKey, select the max() + 1 option.
The options available in this list have the following meaning:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapped value</td>
<td>Allows source data to be mapped to the database field directly, and is the</td>
</tr>
<tr>
<td></td>
<td>standard setting for all database fields. It is also possible to use a stored</td>
</tr>
<tr>
<td></td>
<td>procedure to supply a key value by defining a relation, see Using Stored</td>
</tr>
<tr>
<td></td>
<td>Procedures to Generate Keys.</td>
</tr>
<tr>
<td>max() + 1</td>
<td>Generates the key values based on the existing keys in the database. For</td>
</tr>
<tr>
<td></td>
<td>example, if the table has three records, with primary keys 1, 2, and 3,</td>
</tr>
<tr>
<td></td>
<td>then max() + 1 is 4.</td>
</tr>
<tr>
<td></td>
<td>In this example, the &quot;Altova&quot; table has only one record with primary key 1,</td>
</tr>
<tr>
<td></td>
<td>so max() + 1 is 2, which is the expected value of the new primary key.</td>
</tr>
<tr>
<td>DB-generated</td>
<td>The database uses the Identity function to generate key values.</td>
</tr>
</tbody>
</table>

The option mapped value next to "Name" signifies that this column will get the value directly from the mapping. For reference to other options available on the Database Table Actions dialog box, see Database Table Actions Settings.

Step 5: Preview the mapping and update the database

Click the Output tab to preview the mapping. A SQL script is generated, containing actions to be executed against the database. The script has not modified the database yet; it is only for preview.

```
1 /*
2 The following SQL statements are only for preview and may not be executed in another SQL query tool
3 To execute these statements use function "Run SQL-script" from menu "Output"
4 Connect to database using the following connection-string:
5 Data Source=\wep\6DBDocumentation\PublicExampleFiles\EN\MapForce\DB Insertaltova.mdb;Provider=Microsoft.Jet.OLEDB.4.0
6 */
7 SELECT IF(MAX([Altova].[PrimaryKey]) IS NULL,MAX([Altova].[PrimaryKey]) + 1 AS [PrimaryKey] FROM [Altova]
8 --> %PrimaryKey1%
9
10 INSERT INTO [Altova] ([Name], [PrimaryKey]) VALUES ('Microtech OrgChart', '%PrimaryKey1%')
```

To run the script against the database:

- On the Output menu, click Run SQL-Script.

Note: Running the SQL script directly from MapForce is just one of the ways to update the database, see also Executing Mappings Which Modify Databases.

To see the result, open the altova.mdb database in DatabaseSpy or Access. Notice that a new "Microtech OrgChart" record has been added to the "Altova" table with the new primary key 2. The data for this record originated in the input XML instance.
You have now finished creating a mapping which inserts data into a database table. For a mapping example which inserts data both into the current table and a dependent child table, see Inserting Data into Multiple Related Tables.

7.2.3.2 Inserting Data into Multiple Linked Tables

A database table may be a "parent" table; that is, it might be referred by other tables in the database through foreign key relationships. In such scenarios, you can configure the mapping to insert records not only into the parent table, but also into dependent child tables. For example, when inserting a new "company" record into a database table, you can also insert records for offices linked to this company, as well as their children departments, people, and so on.

This example shows you how to insert data into several tables while preserving the database relationships. It is a slightly more elaborate version of the previous example, Inserting Data into a Table. The example is accompanied by a sample mapping, and it uses the following files:

- Altova_Hierarchical.mfd — the actual mapping file.
- Altova_Hierarchical.xml — contains the source data to be inserted into the database.
- Altova_Hierarchical.xsd — the schema used to validate the instance file above.
- AltovaTarget.mdb — the target database to be updated.

All files are available in the <Documents>\Altova\MapForce2019\MapForceExamples\ folder. Below, the complete path to them will be omitted, for simplicity.

The mapping in this example modifies a sample database file. It is strongly recommended to back up the original database and start with a new copy before following the steps below. This ensures that the original examples are not overridden and that you get the same results as below. For more information, see Executing Mappings Which Modify Databases.

The goal of the mapping is to replace data in the target database (AltovaTarget.mdb) with data from a source XML file. The XML file structure roughly corresponds to the hierarchical structure of tables in the database. It is an organization chart, structured as follows: the top element is a company which contains two offices. Each office contains departments, and each department contains people. The same hierarchy exists in the AltovaTarget.mdb, where the "Altova" table corresponds to the company. This table is linked, through a foreign key relationship, to records in the "Office" table. Likewise, the "Office" links to "Department", and "Department" links to "Person". To view a relationship diagram of the AltovaTarget.mdb database, open it in the “Relationships” view of Access (see also Handling Database Relationships).
To achieve the mapping goal, we will take the steps below.

Step 1: Insert the source XML component
- On the Insert menu, click XML Schema/File, and browse for Altova_Hierarchical.xsd. When prompted to supply an instance file, browse for Altova_Hierarchical.xml.

Step 2: Insert the target database
- On the Insert menu, click Database, and go through the wizard steps to connect to AltovaTarget.mdb. The instructions for connecting to this database are the same as for altova.mdb (see Example: Adding the "altova.mdb" Database to the Mapping).

Step 3: Draw the connections
- Draw the mapping connections as shown below. Notice that the primary and foreign keys are not mapped; they will be generated on the fly, as shown below.
Step 4: Configure the Insert actions

1. On the target component, click the **Action: Insert** button next to the “Altova” table and configure the \( \text{max()} + 1 \) setting of the primary key as shown below. This setting was explained in more detail in the previous example, see *Inserting Data into a Table*. 

**Note:** If unwanted connections are automatically drawn for descending items, the option “Auto-connect children” is active. In this case, to undo the last action, select the menu option **Edit | Undo**. To disable the auto-connect option, select the menu option **Connection | Auto-connect matching children**.
Also, notice that the **DELETE all records** option is enabled. This clears all existing records from the table, before new ones are entered, which is the desired behavior in this example. If you disable this option, new records (with a new primary key) will be added to the database in addition to existing ones, every time you run the mapping, which is not the desired behavior.

For the scope of this example, the option **also delete all records from child tables** is also enabled. This ensures that not only records from the "Altova" table are deleted, but also all records in tables that are linked to "Altova" table through a foreign key relationship. If the child tables have their own child tables, those will also be deleted, and so on, down to the last table in the dependency tree. If you attempted to delete only records from the root "Altova" table, this would violate the database integrity, and the mapping execution would fail.

For reference to other options available on the Database Table Actions dialog box, see **Database Table Actions Settings**.

2. Click **OK** to close the dialog box. Notice that, on the mapping area, the appearance of the button has now changed to **DEL.A[in]** This indicates that a "Delete" statement is configured to take place before the "Insert" action.

3. Click the **A[in]** button next to the "Office" table and configure the **max() + 1** setting of the primary key.
4. Perform step 3 for each table descending from "Office", namely: "Address", "Department", and "Person". Make sure that all these tables are immediate descendants of the root "Altova" table. For an explanation of what is a "root" table and why it is necessary, see Handling Database Relationships.

Step 5: Preview the mapping and update the database

Click the Output tab to preview the mapping. A SQL script is generated, containing actions to be executed against the database. The script has not modified the database yet; it is only for preview.

To run the script against the database:

- On the Output menu, click Run SQL-Script.

Note: Running the SQL script directly from MapForce is just one of the ways to update the database, see also Executing Mappings Which Modify Databases.

To see the result, open the "Altova" table in Microsoft Access, and observe how relationships from the XML file have now been propagated to the database, from the "Altova" table down to the "Person" table.
You have now finished creating a mapping which inserts data into multiple database tables, while preserving the table integrity relationships.

7.2.3.3 Updating a Table

This example shows you how to update data of an existing database table with data coming from an XML source. The example uses the following files:

- `altova-cmpy.xml` — contains the source data to be inserted into the database.
- `Altova_Hierarchical.xsd` — the schema used to validate the instance file above.
- `altova.mdb` — the target database to be updated.

All files are available in the `<Documents>\Altova\MapForce2019\MapForceExamples` folder. Below, the complete path to them will be omitted, for simplicity.

The mapping in this example modifies a sample database file. It is strongly recommended to back up the original database and start with a new copy before following the steps below. This ensures that the original examples are not overridden and that you get the same results as below. For more information, see Executing Mappings Which Modify Databases.

The goal of the mapping is to update all records in "Person" table with instances of "Person" from the XML document. Each person in the XML file has a PrimaryKey child element. Each person in the "Person" table has a PrimaryKey column. Only those records where a person's PrimaryKey in the XML file corresponds to a person's PrimaryKey in the database must be updated.

To achieve the mapping goal, we will take the steps below.

**Step 1: Insert the source XML component**

- On the **Insert** menu, click **XML Schema/File**, and browse for **Altova_Hierarchical.xsd**. When prompted to supply an instance file, browse for **altova-cmpy.xml**.
Step 2: Insert the target database
- On the Insert menu, click Database, and go through the wizard steps to connect to altova.mdb (see Example: Adding the "altova.mdb" Database to the Mapping).

Step 3: Draw the connections
- Draw the mapping connections as shown below.

Step 4: Configure the Update action
1. On the target component, click the Action: Insert (A In) button next to the “Person” table.
2. Next to Action on record, select Update if... This changes the database table action to a conditional update action. That is, the current record will only be updated when a condition is satisfied (see next step).
3. Next to PrimaryKey, select the value equal, as shown below. This defines the update condition: that is, the database record will be updated only when its PrimaryKey value is equal to the PrimaryKey value coming from the mapping.
In this example, the equality operator is applied to the PrimaryKey field, which is a likely scenario when updating databases. Note that conditions can also be defined on other fields which are not necessarily primary keys. For example, by selecting equal next to the First and Last fields, you would update only those records where both the first and last name is equal to that in the source XML.

Optionally, select the NULL equal check box. In this example, this check box is irrelevant, because neither the PrimaryKey field in the source XML nor the PrimaryKey field in the database contains any null values. However, you should select this check box if your data contains null values, and if you want to treat such null values as equal; otherwise, you may get undesired results. For more information, see Handling Nulls in Database Table Actions.

4. Click OK to close the dialog box. Notice that, back on the mapping, the Action: Insert button has now changed to an Action: Update (Update) button. This indicates that an update action is configured to take place for this table.

Step 5: Preview the mapping and update the database

Click the Output tab to preview the mapping. A SQL script is generated, containing actions to be executed against the database. The script has not modified the database yet; it is only for preview.
To run the script against the database:

- On the Output menu, click Run SQL-Script.

**Note:** Running the SQL script directly from MapForce is just one of the ways to update the database, see also Executing Mappings Which Modify Databases.

### 7.2.3.4 "Update if... Insert Rest" Action

Sometimes, it is necessary not only to update existing records, but also to insert new records into the same database table. For such cases, MapForce provides an "Update if... Insert Rest" action. This works as follows:

- If the **Update if** condition is true, then the existing database record is updated with data from the mapping.
- If the **Update if** condition is false, and an **Insert Rest** condition exists, then a new record is inserted.
- If records exist in the database with no counterpart in the source file, then these records remain unchanged.

#### MySQL ODBC note

If the target database is MySQL through ODBC, the option **Return matched rows instead of affected rows** must be enabled in the **Cursor/Results** tab of MySQL ODBC Connector. Alternatively, if you enter the connection string manually through the Database Connection wizard, add **Option=2** to the connection string, for example: `Dsn=mydsn;Option=2;`

To enable this option from MySQL ODBC Connector:

1. Press the **Windows** key and start typing "ODBC".
2. Run the ODBC Data Sources Administrator (either 32-bit or 64-bit, depending on the platform of the installed MySQL ODBC Connector).
3. Click the Data Source Name (DSN) used by the MapForce mapping, and then click **Configure**.

4. Click **Details >>** to make the advanced options available.
5. Click the **Cursors/Results** tab, and then select the check box **Return matched rows instead of affected rows.**

**Example**

The following example shows you how to merge (both update and insert) data from an XML source into a database table. The example uses the following files:

- **altova-cmpy-extra.xml** — contains the source data to be inserted into the database.
- **Altova_Hierarchical.xsd** — the schema used to validate the instance file above.
- **altova.mdb** — the target database to be updated.

All files are available in the folder `<Documents>\Altova\MapForce2019\MapForceExamples\`. Below, the complete path to them will be omitted, for simplicity.

The mapping in this example modifies a sample database file. It is strongly recommended to back up the original database and start with a new copy before following the steps below. This ensures that the original examples are not overridden and that you get the same results as below. For more information, see Executing Mappings Which Modify Databases.

The goal of the mapping is to merge all records from a source XML document into a target "Person" table. Namely, for each record in the source XML data, the mapping must do the
following:

- If the person’s PrimaryKey in the XML file corresponds to a person’s PrimaryKey in the database, then update the record.
- Any existing records in the Person table which do not meet the above condition must not be affected.
- If the person’s PrimaryKey in the XML file does not have a match in the target database table, then add a new record to the database table.

To achieve the mapping goal, we will take the steps below.

**Step 1: Insert the source XML component**
- On the **Insert** menu, click **XML Schema/File**, and browse for *Altova_Hierarchical.xsd*. When prompted to supply an instance file, browse for *altova-cmpy-extra.xml*.

**Step 2: Insert the target database**
- On the **Insert** menu, click **Database**, and go through the wizard steps to connect to *altova.mdb* (see **Example: Adding the "altova.mdb" Database to the Mapping**).

**Step 3: Draw the connections**
- Draw the mapping connections as shown below.
Step 4: Configure the "Update if... Insert Rest" actions

1. On the target component, click the **Action: Insert** button next to the "Person" table.
2. Next to **Action on record**, select **Update if...**. This changes the database table action to a conditional update action. That is, the current record will only be updated when a condition is satisfied (see next step).
3. Next to **PrimaryKey**, select the value **equal**, as shown below. This defines the update condition: that is, the database record will be updated only when its **PrimaryKey** value is equal to the **PrimaryKey** value coming from the mapping.

```
<table>
<thead>
<tr>
<th>Action on record</th>
<th>NULL equal</th>
<th>Update if...</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrimaryKey</td>
<td></td>
<td>equal</td>
</tr>
<tr>
<td>ForeignKey</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMail</td>
<td></td>
<td></td>
</tr>
<tr>
<td>First</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Last</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PhoneExt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

4. Optionally, select the **NULL equal** check box. In this example, this check box is irrelevant, because neither the **Primary Key** field in the source XML nor the **PrimaryKey** field in the database contains any null values. However, you should select this check box if your data contains null values, and if you want to treat such null values as equal; otherwise, you may get undesired results. For more information, see [Handling Nulls in Database Table Actions](#).
5. Click **Append Action**. This adds a new action to the right of the existing **Update If** action. Configure the new action as **Insert Rest**:

```
<table>
<thead>
<tr>
<th>Action on record</th>
<th>NULL equal</th>
<th>Update if...</th>
<th>Insert Rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrimaryKey</td>
<td></td>
<td>equal</td>
<td>mapped value</td>
</tr>
<tr>
<td>ForeignKey</td>
<td></td>
<td></td>
<td>mapped value</td>
</tr>
<tr>
<td>EMail</td>
<td></td>
<td></td>
<td>mapped value</td>
</tr>
<tr>
<td>First</td>
<td></td>
<td></td>
<td>mapped value</td>
</tr>
<tr>
<td>Last</td>
<td></td>
<td></td>
<td>mapped value</td>
</tr>
<tr>
<td>PhoneExt</td>
<td></td>
<td></td>
<td>mapped value</td>
</tr>
<tr>
<td>Title</td>
<td></td>
<td></td>
<td>mapped value</td>
</tr>
</tbody>
</table>
```

In the image above, the database table actions have been configured in accordance with the goals of the mapping. That is, only when the **Update If**... condition is satisfied will the record be updated; otherwise, it will be inserted. The option "mapped value" specifies that values from the mapping will be used to populate all fields of the record.

It is also possible to define more than two actions against the same database table (this is not necessary in this example, however). At mapping runtime, actions are executed from left to right. The last **Insert** action is considered final; any other actions added after it will be ignored.

Note that the **Append Action** button on the dialog box adds the new action after the
selected one. **Insert Action** adds the new action *before* the selected one. To delete an existing action, click anywhere inside it, and then click **Delete Action**.

6. Click **OK** to close the dialog box. Notice that, back on the mapping, the **Action: Insert** button has now changed to an **Action: Update; Insert** button. This indicates that both an update and an insert action is configured to take place for this table.

**Step 5: Preview the mapping and update the database**

Click the **Output** tab to preview the mapping. A SQL script is generated, containing actions to be executed against the database. The script has not modified the database yet; it is only for preview.

```sql
/*
The following SQL statements are only for preview and may not be executed in another SQL query tool.
To execute these statements use function "Run SQL-script" from menu "Output".
Connect to database using the following connection string:
Data Source=Altova\AltovaMapForce\AltovaMapForceDB Update if... Insert
RestAltova.mdb;Provider=Microsoft.Jet.OLEDB.4.0
*/

UPDATE [Person] SET [ForeignKey] = 1, [Email] = 'A.Aldrich@microtech.com', [First] = 'Albert', [Last] = 'Aldrich', [PhoneExt] = 502, [Title] = 'Manager' WHERE ([Person][PrimaryKey] = 1)


```

**SQL script (partial view) before updating the database**

You may notice that no INSERT statements are visible in the preview script. This is normal behavior, because records are inserted conditionally, and the INSERT statements depend on the result of the **Update If** action (which is not known before the mapping runs).

**Note:** For certain database types, MapForce creates MERGE statements instead of UPDATE statements. For further information, see [MERGE statements](#).

To run the script against the database:

- On the **Output** menu, click **Run SQL-Script**.

Now that the mapping has been executed and the script applied against the database, notice that INSERT statements are visible in the **Output** tab.
The following SQL statements were executed during "Generate output" function.

Every single result is written right to the "-->" string.

These statements are only for preview and may not be executed in another SQL query tool.

The database was connected using the following connection string:

Data Source=\wmgf\00\Documentation\Public\ExampleFiles\EN\MapForceDB Update if... insert
Restaltova.mdb;Provider=Microsoft.Jet.OLEDB.4.0

```
/*
 * SQL script (partial view) after updating the database
 */

UPDATE [Person] SET [ForeignKey] = 1, [Email] = 'A.Aldrich@microtech.com', [First] = 'Albert', [Last] = 'Aldrich',
[PhoneExt] = 562, [Title] = 'Manager' WHERE ([Person].[PrimaryKey] = 1)
--> OK. 1 row(s).

UPDATE [Person] SET [ForeignKey] = 1, [Email] = 'b.bander@microtech.com', [First] = 'Bert', [Last] = 'Bander', [PhoneExt]
= 471, [Title] = 'Accounts Receivable' WHERE ([Person].[PrimaryKey] = 2)
--> OK. 1 row(s).

UPDATE [Person] SET [ForeignKey] = 1, [Email] = 'c.clvisi@microtech.com', [First] = 'Clive', [Last] = 'Clovis', [PhoneExt] =
963, [Title] = 'Accounting Manager' WHERE ([Person].[PrimaryKey] = 3)
--> OK. 1 row(s).

[PhoneExt] = 765, [Title] = 'HR' WHERE ([Person].[PrimaryKey] = 30)
--> OK. 0 row(s).

INSERT INTO [Person] ([PrimaryKey], [ForeignKey], [Email], [First], [Last], [PhoneExt], [Title]) VALUES (30, 1,
'c.Cicada@microtech.com', 'Camila', 'Cicada', 765, 'HR')
--> OK. 1 row(s).
```

**Note:** Running the SQL script directly from MapForce is just one of the ways to update the database, see also Executing Mappings Which Modify Databases.

If you open the "Person" table in the DB query tab of MapForce (see Browsing and Querying Databases), you can see the result of the mapping as follows:

- All database records which had corresponding primary keys in the XML file have been updated. Examples are records with primary key 1, 2, 3, 4, and 5.
- All database records which had no corresponding keys in the XML file remained unaffected. Examples are records with primary key 6, 7, 8, and 9.
- New records have been inserted to the "Person" table (where key did not already exist in the database). Examples are records with primary key 30 and 31.
MERGE Statements

7.2.3.5

For certain mappings which both update and insert data into a database table (see also "Update if... Insert Rest! Action"), MapForce generates MERGE statements to be executed against the database at mapping runtime. The execution engine may not necessarily be MapForce, see Executing Mappings Which Modify Databases.

MERGE statements are supported for the following database types:

- SQL Server 2008 and later
- Oracle
- DB2
- Firebird

MERGE statements reduce the number of database server calls, since they combine the INSERT and UPDATE statements into one. Also, in case of MERGE statements, the consistency check is done by the database.

To see whether the mapping will execute database MERGE statements at runtime (as opposed to applying a combination of INSERT and UPDATE statements):

The "Person" table after updating the database
1. Create a mapping which uses an **Update if...** as well as an **Insert Rest** action. For an example, see "**Update if... Insert Rest**" Action.
2. Preview the mapping, by clicking the **Output** tab.

If MERGE is supported by the database type, the generated SQL script includes MERGE statements, for example:

```sql
/*
The following SQL statements are only for preview and may not be executed in another SQL query tool!
To execute these statements use function "Run SQL script" from menu "Output".
Connect to database using the following connection-string:

Catalog=DB MERGE;MultipleActiveResultSets=True;Password=******;Persist Security Info=True;User ID=
*/
SET QUOTED_IDENTIFIER ON
UPDATE [dbo].[Users] SET [FirstName] = (CAST('Despine' AS nvarchar(50))), [LastName] = 
(CAST('Butter' AS nvarchar(50))) WHERE ([dbo].[Users].[UserID] = 1)
SELECT UserID FROM [dbo].[Users] WHERE ([UserID] = 1)
---------- >userid1
MERGE INTO [dbo].[Addresses] AS T USING ( VALUES ('%userid1%', 1, 1, (CAST('Home' AS nvarchar(20)) ), (CAST('Louisville' AS nvarchar(50))), (CAST('12th St' AS nvarchar(50))); (CAST('12' AS nvarchar(20)))) ) AS S ( [UserID], [isShipping], [isBilling], [AddressType], [City], [Street], [Number] ) ON ( S.[UserID] = T.[UserID] ) WHEN NOT MATCHED THEN UPDATE SET [isShipping] = S. [isShipping], [isBilling] = S.[isBilling], [AddressType] = S.[AddressType], [City] = S.[City], [Street] = S.[Street], [Number] = S.[Number] WHEN NOT MATCHED THEN INSERT ( [UserID], [isShipping], [isBilling], [AddressType], [City], [Street], [Number] ) VALUES ( S.[UserID], S.[isShipping], S. [isBilling], S.[AddressType], S.[City], S.[Street], S.[Number] );
```

If MERGE is not supported by the database type, the generated SQL script includes UPDATE statements only. No INSERT statements are visible for preview, since those are to be executed only if the **Update If...** condition is not satisfied (and this is not known before the mapping execution).

**Notes**

- MapForce creates MERGE statements automatically when it detects a supported database type; it is not possible to manually influence whether MapForce should create a MERGE statement.
- If you are updating multiple tables having parent-child relationships, merges are created only for "leaf" tables. A "leaf" table is the deepest child table that is mapped. For example, in the mapping below, **Update If... Insert Rest...** actions have been defined for both **Users** table and **Addresses** table. However, MERGE statements will be generated only for the leaf table, **Addresses**. The parent table, **Users**, gets UPDATE or INSERT statements instead of MERGE.
With MERGE statements, the "Bulk Transfer" option (see Bulk Inserts (MapForce Server) is supported only for ODBC and JDBC database connections.

7.2.3.6 Options for Child Tables When Updating a Parent Table

When the mapping updates a table which is a "parent" table (that is, it has foreign key relationships to other tables), you can configure how the dependent records should be treated both in the source data and in the target table. For example, let's assume that you want to update the "Department" table in the altova.mdb database. Because every person is linked to a department by means of a foreign key, you will likely want to take action against the "Person" table as well (which could be an insert, update, or delete). Doing so would help you maintain the database integrity and avoid mapping errors.

This topic discusses the options available for the "Person" table when you update the parent "Department" table. It makes use of the following example files:

- altova-cmpy-extra.xml — contains the source data to be inserted into the database.
- Altova_Hierarchical.xsd — the schema used to validate the instance file above.
- altova.mdb — the target database to be updated.

All files are available in the <Documents>\Altova\MapForce2019\MapForceExamples folder. Below, the complete path to them will be omitted, for simplicity.

The mapping in this example modifies a sample database file. It is strongly recommended to back up the original database and start with a new copy before following the steps below. This ensures that the original examples are not overridden and that you get the same results as below. For more information, see Executing Mappings Which Modify Databases.

First, add the source XML schema and instance as well as the target database to the mapping
As illustrated above, the mapping updates the "Department" table in the target database. The "Department" table is chosen as "root" table. For more information about what a root table is and why it is necessary, see Handling Database Relationships. The action to be taken against the child "Person" table is the subject of this topic.

The following tables illustrate various configuration options and the corresponding mapping result. These options can be selected from the Database Table Actions dialog box of the parent "Department" table and the child "Person" table.
### Configuration A

<table>
<thead>
<tr>
<th>Settings</th>
<th>Mapping result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Department</strong> table</td>
<td>• Updates Department records where PrimaryKey in the source XML corresponds to the PrimaryKey in the database table.</td>
</tr>
<tr>
<td></td>
<td>• Does not update existing Department records which do not have a counterpart in the input XML file (no such key exists in the source).</td>
</tr>
<tr>
<td></td>
<td>• Inserts, from the input XML instance, all Person records that do not already exist in the database.</td>
</tr>
<tr>
<td></td>
<td>• Deletes child data (Person records) of those Department records which satisfy the Update if... condition.</td>
</tr>
<tr>
<td><strong>Person</strong> table</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Configuration B

<table>
<thead>
<tr>
<th>Settings</th>
<th>Mapping result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action on record</td>
<td>The mapping fails with an SQL execution error. The reason is that the mapping attempts to insert new Person records with the same primary key as the existing Person records. If you want to insert records from the input XML in addition to those already in the database, see the next option.</td>
</tr>
<tr>
<td>PrimaryKey</td>
<td>equal</td>
</tr>
<tr>
<td>ForeignKey</td>
<td>equal</td>
</tr>
<tr>
<td>Name</td>
<td>equal</td>
</tr>
<tr>
<td>Delete data in child tables</td>
<td></td>
</tr>
<tr>
<td>Ignore input child data</td>
<td></td>
</tr>
</tbody>
</table>

**“Department” table**

| Action on record              | Insert All                                                                                                                                   |
| PrimaryKey                    | mapped value                                                                                                                               |
| ForeignKey                    | foreign key                                                                                                                                |
| EMail                         | mapped value                                                                                                                               |
| First                         | mapped value                                                                                                                               |
| Last                          | mapped value                                                                                                                               |
| PhoneExt                      | mapped value                                                                                                                               |
| Title                         | mapped value                                                                                                                               |

**“Person” table**
### Configuration C

<table>
<thead>
<tr>
<th><strong>Settings</strong></th>
<th><strong>Mapping result</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Action on record</strong></td>
<td>Updates Department records where PrimaryKey in the source XML corresponds to the PrimaryKey in the database table.</td>
</tr>
<tr>
<td></td>
<td>Does not update existing Department records which do not have a counterpart in the input XML file (no such key exists in the source).</td>
</tr>
<tr>
<td></td>
<td>New Person records (with generated primary keys) are inserted into the Person table in addition to existing ones.</td>
</tr>
<tr>
<td><strong>Delete data in child tables</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Ignore input child data</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Person</strong></td>
<td></td>
</tr>
</tbody>
</table>

*“Department” table*

<table>
<thead>
<tr>
<th><strong>Action on record</strong></th>
<th><strong>Update if...</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PrimaryKey</strong></td>
<td>equal</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td></td>
</tr>
</tbody>
</table>

*“Person” table*
### Configuration D

<table>
<thead>
<tr>
<th>Settings</th>
<th>Mapping result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Action on record</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Department</strong> table</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Ignore input child data</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Person</strong> table</td>
<td></td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>Action on record</th>
<th>Insert All</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrimaryKey</td>
<td>mapped value</td>
</tr>
<tr>
<td>ForeignKey</td>
<td>foreign key</td>
</tr>
<tr>
<td>EMail</td>
<td>mapped value</td>
</tr>
<tr>
<td>First</td>
<td>mapped value</td>
</tr>
<tr>
<td>Last</td>
<td>mapped value</td>
</tr>
<tr>
<td>PhoneExt</td>
<td>mapped value</td>
</tr>
<tr>
<td>Title</td>
<td>mapped value</td>
</tr>
</tbody>
</table>
```
**Configuration E**

<table>
<thead>
<tr>
<th><strong>Settings</strong></th>
<th><strong>Mapping result</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Action on record</strong></td>
<td><strong>Update if...</strong></td>
</tr>
<tr>
<td>Departments</td>
<td>Updates Department records where PrimaryKey in the source XML corresponds to the PrimaryKey in the database table.</td>
</tr>
<tr>
<td>Person records</td>
<td>Does not update existing Department records which do not have a counterpart in the input XML file (no such key exists in the source).</td>
</tr>
<tr>
<td>Person records linked to a Department that did not meet the Update if... condition remain in the database.</td>
<td></td>
</tr>
<tr>
<td>No records in the Person table are updated.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>“Department” table</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Action on record</td>
</tr>
<tr>
<td>PrimaryKey</td>
</tr>
<tr>
<td>ForeignKey</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>“Person” table</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Action on record</td>
</tr>
<tr>
<td>PrimaryKey</td>
</tr>
<tr>
<td>EMail</td>
</tr>
<tr>
<td>First</td>
</tr>
<tr>
<td>Last</td>
</tr>
<tr>
<td>PhoneExt</td>
</tr>
<tr>
<td>Title</td>
</tr>
</tbody>
</table>
### Configuration F

<table>
<thead>
<tr>
<th>Settings</th>
<th>Mapping result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>“Department” table</strong></td>
<td>• Updates Department records where PrimaryKey in the source XML corresponds to the PrimaryKey in the database table.</td>
</tr>
<tr>
<td></td>
<td>• Does not update existing Department records which do not have a counterpart in the input XML file (no such key exists in the source).</td>
</tr>
<tr>
<td></td>
<td>• Deletes all Person records linked to a Department which has a corresponding PrimaryKey in the source XML. The reason is that the Delete data in child tables option is enabled for the parent Department table.</td>
</tr>
<tr>
<td></td>
<td>• Person records linked to a department that did not meet the Update if... condition remain in the database.</td>
</tr>
<tr>
<td>Action on record</td>
<td>Delete if...</td>
</tr>
<tr>
<td>PrimaryKey</td>
<td>equal</td>
</tr>
<tr>
<td>ForeignKey</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>Delete data in child tables</td>
<td>checked</td>
</tr>
<tr>
<td>Ignore input child data</td>
<td></td>
</tr>
<tr>
<td>Person</td>
<td></td>
</tr>
</tbody>
</table>

| **“Person” table**         |                                                                                  |
| Action on record          | Delete if...                                                                    |
| PrimaryKey                | equal                                                                           |
| ForeignKey                | foreign key                                                                     |
| EMail                     |                                                                                 |
| First                     |                                                                                 |
| Last                      |                                                                                 |
| PhoneExt                  |                                                                                 |
| Title                     |                                                                                 |
7.2.3.7 "Delete if..." Action

The table action Delete if... is used to delete data from a database table conditionally. You can define this action from the Database Table Actions dialog box (see Database Table Actions Settings). For example, when mapping data from a source XML to a target database, you can configure a Delete if... condition to check whether a certain field in the source XML is equal to a field in the target database record (typically, a primary key value). If the Delete if... condition is true (that is, the two fields are equal), the database record will be deleted when the mapping runs.

Note: The Delete if... table action should not be confused with the Delete data in child tables option available in the Database Table Actions dialog box. The Delete if... table action only affects the table for which the action is defined; no other tables are affected.

This example shows you how to delete data from a database table conditionally, and also insert records into the same database table based on the outcome of the delete condition.

This example uses the following files:

- altova-cmpy-extra.xml — contains the source data to be inserted into the database.
- Altova_Hierarchical.xsd — the schema used to validate the instance file above.
- altova.mdb — the target database to be updated.

### Configuration G

<table>
<thead>
<tr>
<th>Settings</th>
<th>Mapping result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action on record</td>
<td>Updates Department records where PrimaryKey in the source XML corresponds to the PrimaryKey in the database table. Does not update existing Department records which do not have a counterpart in the input XML file (no such key exists in the source). Deletes all Person records which satisfy both of the following conditions:</td>
</tr>
<tr>
<td>PrimaryKey</td>
<td>equal</td>
</tr>
<tr>
<td>ForeignKey</td>
<td>equal</td>
</tr>
<tr>
<td>Name</td>
<td>equal</td>
</tr>
<tr>
<td>Delete data in child tables</td>
<td>a. The Person record is linked to a Department which has a corresponding PrimaryKey in the source XML, and b. The Person record has a corresponding PrimaryKey in the source XML.</td>
</tr>
<tr>
<td>Ignore input child data</td>
<td></td>
</tr>
<tr>
<td>Person</td>
<td></td>
</tr>
</tbody>
</table>

**"Department" table**

<table>
<thead>
<tr>
<th>Action on record</th>
<th>Delete if...</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrimaryKey</td>
<td>equal</td>
</tr>
<tr>
<td>ForeignKey</td>
<td>foreign key</td>
</tr>
<tr>
<td>EMail</td>
<td>equal</td>
</tr>
<tr>
<td>First</td>
<td>equal</td>
</tr>
<tr>
<td>Last</td>
<td>equal</td>
</tr>
<tr>
<td>PhoneExt</td>
<td>equal</td>
</tr>
<tr>
<td>Title</td>
<td>equal</td>
</tr>
</tbody>
</table>

**"Person" table**
All files are available in the <Documents>\Altova\MapForce2019\MapForceExamples folder. Below, the complete path to them will be omitted, for simplicity.

The mapping in this example modifies a sample database file. It is strongly recommended to back up the original database and start with a new copy before following the steps below. This ensures that the original examples are not overridden and that you get the same results as below. For more information, see Executing Mappings Which Modify Databases.

The goal of the mapping is as follows:

- If any person records with the same PRIMARY KEY exist both in the source XML and the target Person table, they must be deleted from the Person table.
- All other records from the source XML must be inserted into the Person table.

To achieve the mapping goal, we will take the steps below.

**Step 1: Insert the source XML component**

**Step 2: Insert the target database**
- On the Insert menu, click Database, and go through the wizard steps to connect to altova.mdb (see Example: Adding the "altova.mdb" Database to the Mapping).

**Step 3: Draw the connections**
- Draw the mapping connections as shown below.
Step 4: Configure the "Delete if... Insert Rest" actions

1. On the target component, click the **Action: Insert** button next to the "Person" table.

2. Next to **Action on record**, select **Delete if...**. This changes the database table action to a conditional delete action. That is, the current record will only be deleted when a condition is satisfied (see next step).

3. Next to **PrimaryKey**, select the value **equal**, as shown below. This defines the update condition: that is, the database record will be deleted only when its **PrimaryKey** value is equal to the **PrimaryKey** value coming from the mapping.
4. Optionally, select the **NULL equal** check box. In this example, this check box is irrelevant, because neither the **PrimaryKey** field in the source XML nor the **PrimaryKey** field in the database contains any null values. However, you should select this check box if your data contains null values, and if you want to treat such null values as equal; otherwise, you may get undesired results. For more information, see **Handling Nulls in Database Table Actions**.

5. Click **Append Action**. This adds a new action to the right of the existing **Delete If** action. Configure the new action as **Insert Rest**:

   ![Actions to execute for each record](image)

   In the image above, the database table actions have been configured in accordance with the goals of the mapping. That is, only when the **Delete If...** condition is satisfied will the record be deleted; otherwise, it will be inserted. The option "mapped value" specifies that values from the mapping will be used to populate all fields of the record.

6. Click **OK** to close the dialog box. Notice that, back on the mapping, the **Action: Insert** button has now changed to an **Action: Delete; Insert** button. This indicates that both a delete and an insert action is configured for this table.

---

**Step 5: Preview the mapping and update the database**

Click the **Output** tab to preview the mapping. A SQL script is generated, containing actions to be executed against the database. The script has not modified the database yet; it is only for preview. To run the script against the database:

- On the **Output** menu, click **Run SQL-Script**.

**Note:** Running the SQL script directly from MapForce is just one of the ways to update the database, see also **Executing Mappings Which Modify Databases**.

If you open the "Person" table in the DB query tab of MapForce (see **Browsing and Querying Databases**), you can see the result of the mapping as follows:

- All database records which had corresponding primary keys in the XML file have been deleted. Examples are records with primary key 1, 2, 3, 4, and 5.
- All database records which had no corresponding keys in the XML file remained unaffected. Examples are records with primary key 6, 7, 8, 9, 10, 11, 12, and 13.
- New records have been inserted to the "Person" table (where key did not already exist in the database). Examples are records with primary key 30 and 31.
The "Person" table after updating the database

<table>
<thead>
<tr>
<th>PrimaryKey</th>
<th>ForeignKey</th>
<th>Email</th>
<th>First</th>
<th>Last</th>
<th>PhoneExt</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td><a href="mailto:f.landis@nanonull.com">f.landis@nanonull.com</a></td>
<td>Fred</td>
<td>Landis</td>
<td>951</td>
<td>Program Manager</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td><a href="mailto:m.landis@nanonull.com">m.landis@nanonull.com</a></td>
<td>Michelle</td>
<td>Butler</td>
<td>654</td>
<td>Software Engineer</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td><a href="mailto:l.little@nanonull.com">l.little@nanonull.com</a></td>
<td>Ted</td>
<td>Little</td>
<td>852</td>
<td>Software Engineer</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td><a href="mailto:a.way@nanonull.com">a.way@nanonull.com</a></td>
<td>Ann</td>
<td>Way</td>
<td>951</td>
<td>Technical Writer</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td><a href="mailto:l.gardner@nanonull.com">l.gardner@nanonull.com</a></td>
<td>Liz</td>
<td>Gardner</td>
<td>753</td>
<td>Software Engineer</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td><a href="mailto:p.smith@nanonull.com">p.smith@nanonull.com</a></td>
<td>Paul</td>
<td>Smith</td>
<td>334</td>
<td>Software Engineer</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td><a href="mailto:a.martin@nanonull.com">a.martin@nanonull.com</a></td>
<td>Alex</td>
<td>Martin</td>
<td>778</td>
<td>IT Manager</td>
</tr>
<tr>
<td>8</td>
<td>13</td>
<td><a href="mailto:g.hammer@nanonull.com">g.hammer@nanonull.com</a></td>
<td>George</td>
<td>Hammer</td>
<td>223</td>
<td>Web Developer</td>
</tr>
<tr>
<td>9</td>
<td>30</td>
<td><a href="mailto:c.cicada@microtech.com">c.cicada@microtech.com</a></td>
<td>Camila</td>
<td>Cicada</td>
<td>765</td>
<td>HR</td>
</tr>
<tr>
<td>10</td>
<td>31</td>
<td><a href="mailto:c.corrigan@microtech.com">c.corrigan@microtech.com</a></td>
<td>Carol</td>
<td>Corrigan</td>
<td>629</td>
<td>Admin</td>
</tr>
</tbody>
</table>

7.2.3.8 "Ignore if..." Action

The table action Ignore if... is used to prevent certain records in a database table from being updated, based on a defined condition. The Ignore if... action is only meaningful when used in combination with another database table action (such as the Insert Rest action). For example, when mapping data from a source XML to a target database, you can configure an Ignore if... condition to check whether a certain field in the source XML is equal to a field in the target database record (typically, a primary key value). If the Ignore if... condition is true (that is, the two fields are equal), the database record will be ignored when the mapping runs, and the next defined action (Insert Rest, for example) will be executed.

This example shows you how insert records into a database table based on the outcome of the Ignore if... condition. It uses the following files:

- altova-cmpy-extra.xml — contains the source data to be inserted into the database.
- Altova_Hierarchical.xsd — the schema used to validate the instance file above.
- altova.mdb — the target database to be updated.

All files are available in the <Documents>\Altova\MapForce2019\MapForceExamples\ folder. Below, the complete path to them will be omitted, for simplicity.

The mapping in this example modifies a sample database file. It is strongly recommended to back up the original database and start with a new copy before following the steps below. This ensures that the original examples are not overridden and that you get the same results as below. For more information, see Executing Mappings Which Modify Databases.

The goal of the mapping is as follows:

- If any person records with the same PrimaryKey exist both in the source XML and the target Person table, no action must be taken against them (that is, they must be ignored).
- If any person records which do not meet the above condition exist in the Person table, no
action must be taken against them either.

- Records from the source XML which do not have a counterpart (no primary key) in the Person table must be treated as new and inserted into the Person table with a new primary key.

To achieve the mapping goal, we will take the steps below.

**Step 1: Insert the source XML component**

- On the **Insert** menu, click **XML Schema/File**, and browse for `Altova_Hierarchical.xsd`. When prompted to supply an instance file, browse for `altova-cmpy-extra.xml`.

**Step 2: Insert the target database**

- On the **Insert** menu, click **Database**, and go through the wizard steps to connect to `altova.mdb` (see Example: Adding the “altova.mdb” Database to the Mapping).

**Step 3: Draw the connections**

- Draw the mapping connections as shown below.
Step 4: Configure the "Ignore if... Insert Rest" actions
1. On the target component, click the Action: Insert ( ) button next to the "Person" table.
2. Next to Action on record, select Ignore if... . This changes the database table action to a conditional ignore action. That is, the current record will only be ignored when a condition is satisfied (see next step).
3. Next to PrimaryKey, select the value equal, as shown below. This defines the ignore condition: that is, the database record will be ignored only when its PrimaryKey value is equal to the PrimaryKey value coming from the mapping.

4. Optionally, select the NULL equal check box. In this example, this check box is irrelevant, because neither the PrimaryKey field in the source XML nor the PrimaryKey field in the database contains any null values. However, you should select this check box if your data contains null values, and if you want to treat such null values as equal; otherwise, you may get undesired results. For more information, see Handling Nulls in Database Table Actions.
5. Click Append Action. This adds a new action to the right of the existing Ignore If action. Configure the new action as Insert Rest, with the primary key set to max() + 1, as shown below:

   In the image above, the database table actions have been configured in accordance with the goals of the mapping. That is, only when the Ignore If... condition is satisfied will the record be skipped; otherwise, it will be inserted. The option "mapped value" specifies that values from the mapping will be used to populate all fields of the record. The option max() + 1 generates a unique, new primary key value for the record.

6. Click OK to close the dialog box. Notice that, back on the mapping, the Action: Insert ( ) button has now changed to an Action: Ignore; Insert ( ) button. This indicates that both the ignore and insert actions are configured for this table.

Step 5: Preview the mapping and update the database
Click the Output tab to preview the mapping. A SQL script is generated, containing actions to be
executed against the database. The script has not modified the database yet; it is only for preview. To run the script against the database:

- On the Output menu, click **Run SQL-Script**.

**Note:** Running the SQL script directly from MapForce is just one of the ways to update the database, see also Executing Mappings Which Modify Databases.

If you open the "Person" table in the DB query tab of MapForce (see Browsing and Querying Databases), you can see the result of the mapping as follows:

- All database records which had corresponding primary keys in the XML file satisfied the Ignore if... and remained unaffected. Examples are records with primary key 1, 2, 3, 4, and 5.
- All database records which had no corresponding keys in the XML file did not satisfy the Ignore if... condition but nevertheless remained unaffected. Examples are records with primary key 6, 7, 8, 9, 10, 11, 12, and 13.
- New records have been inserted to the "Person" table (where key did not already exist in the database). Examples are records with primary key 30 and 31 in the source XML file. These were inserted into the database with the new primary key 22 and 23, respectively.

<table>
<thead>
<tr>
<th>PrimaryKey</th>
<th>ForeignKey</th>
<th>Email</th>
<th>FirstName</th>
<th>LastName</th>
<th>PhoneExt</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td><a href="mailto:v.callaby@nanonull.com">v.callaby@nanonull.com</a></td>
<td>Vernon</td>
<td></td>
<td></td>
<td>Office Manager</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td><a href="mailto:f.further@nanonull.com">f.further@nanonull.com</a></td>
<td>Frank</td>
<td></td>
<td></td>
<td>Accounts Receivable</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td><a href="mailto:l.melise@nanonull.com">l.melise@nanonull.com</a></td>
<td>Loby</td>
<td></td>
<td></td>
<td>Accounting Manager</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td><a href="mailto:j.firstbreed@nanonull.com">j.firstbreed@nanonull.com</a></td>
<td>Joe</td>
<td>Firstbreed</td>
<td></td>
<td>Marketing Manager Europe</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td><a href="mailto:s.sanna@nanonull.com">s.sanna@nanonull.com</a></td>
<td>Susi</td>
<td>Sanna</td>
<td></td>
<td>Art Director</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td><a href="mailto:f.landis@nanonull.com">f.landis@nanonull.com</a></td>
<td>Fred</td>
<td>Landis</td>
<td></td>
<td>Program Manager</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td><a href="mailto:m.landis@nanonull.com">m.landis@nanonull.com</a></td>
<td>Michelle</td>
<td>Butler</td>
<td></td>
<td>Software Engineer</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td><a href="mailto:t.little@nanonull.com">t.little@nanonull.com</a></td>
<td>Ted</td>
<td>Little</td>
<td></td>
<td>Software Engineer</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td><a href="mailto:a.way@nanonull.com">a.way@nanonull.com</a></td>
<td>Ann</td>
<td>Way</td>
<td></td>
<td>Technical Writer</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td><a href="mailto:l.gardner@nanonull.com">l.gardner@nanonull.com</a></td>
<td>Liz</td>
<td>Gardner</td>
<td></td>
<td>Software Engineer</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td><a href="mailto:p.smith@nanonull.com">p.smith@nanonull.com</a></td>
<td>Paul</td>
<td>Smith</td>
<td></td>
<td>Software Engineer</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td><a href="mailto:a.martin@nanonull.com">a.martin@nanonull.com</a></td>
<td>Alex</td>
<td>Martin</td>
<td></td>
<td>IT Manager</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td><a href="mailto:g.hammer@nanonull.com">g.hammer@nanonull.com</a></td>
<td>George</td>
<td>Hammer</td>
<td></td>
<td>Web Developer</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td><a href="mailto:j.bond@nanonull.com">j.bond@nanonull.com</a></td>
<td>Jessica</td>
<td>Bond</td>
<td></td>
<td>Support Engineer</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td><a href="mailto:l.king@nanonull.com">l.king@nanonull.com</a></td>
<td>Lui</td>
<td>King</td>
<td></td>
<td>Support Engineer</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
<td><a href="mailto:a.moier@nanonull.com">a.moier@nanonull.com</a></td>
<td>Steve</td>
<td>Moier</td>
<td></td>
<td>Office Manager</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td><a href="mailto:t.bone@nanonull.com">t.bone@nanonull.com</a></td>
<td>Theo</td>
<td>Bone</td>
<td></td>
<td>Accounts Receivable</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td><a href="mailto:m.natta@nanonull.com">m.natta@nanonull.com</a></td>
<td>Max</td>
<td>Natta</td>
<td></td>
<td>PR &amp; Marketing Manager US</td>
</tr>
<tr>
<td>19</td>
<td>7</td>
<td><a href="mailto:v.bass@nanonull.com">v.bass@nanonull.com</a></td>
<td>Valentin</td>
<td>Bass</td>
<td></td>
<td>IT Manager</td>
</tr>
<tr>
<td>20</td>
<td>7</td>
<td><a href="mailto:c.franken@nanonull.com">c.franken@nanonull.com</a></td>
<td>Carl</td>
<td>Franken</td>
<td></td>
<td>Support Engineer</td>
</tr>
<tr>
<td>21</td>
<td>7</td>
<td><a href="mailto:m.redgreen@nanonull.com">m.redgreen@nanonull.com</a></td>
<td>Mark</td>
<td>Redgreen</td>
<td></td>
<td>Support Engineer</td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td><a href="mailto:c.Cicco@microtech.com">c.Cicco@microtech.com</a></td>
<td>Camilla</td>
<td>Cicco</td>
<td></td>
<td>HR</td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td><a href="mailto:c.corrigan@microtech.com">c.corrigan@microtech.com</a></td>
<td>Carol</td>
<td>Corrigan</td>
<td></td>
<td>Admin</td>
</tr>
</tbody>
</table>

*The "Person" table after updating the database*
7.2.3.9 Bulk Inserts (MapForce Server)

The **Use Bulk Transfer** option allows you to insert data at very high speed from a MapForce component (TXT, CSV, DAT, etc.) into a database table. Using this option dramatically speeds up the Insert process, as only one statement needs to be executed instead of many.

The **Use Bulk Transfer** option can be enabled in MapForce, at mapping design time, as shown below. A mapping where this option is enabled can be executed in MapForce, but no bulk insert applies at this stage. The actual bulk transfer of data occurs when the mapping is run by MapForce Server.

Bulk transfer is supported when the following conditions are true:

- The mapping transformation language is set to BUILT-IN. For further information, see Selecting a Transformation Language.
- The mapping is run by MapForce Server (either standalone or under FlowForce Server management). This means that the mapping must be either compiled to .mfx format or deployed to FlowForce Server. For further information, see Compiling Mappings to MapForce Server Execution Files and Deploying Mappings to FlowForce Server.
- The MapForce Server license is not limited to "single thread execution" on a multi-core machine. That is, the Limit to single thread execution check box in the "Server Management" tab of Altova LicenseServer must be inactive.
- The database action is "Insert all", see also Database Table Actions Settings.
- The table into which the data is to be bulk loaded must be a "leaf" table, that is, on the lowest hierarchy of the database. There should not be any related tables, views, or stored procedures referencing the table in the mapping.
- The database driver supports bulk insert on WHERE conditions.

The following table summarizes support for bulk inserts depending on the database kind and the driver used.

<table>
<thead>
<tr>
<th>Database</th>
<th>ADO</th>
<th>ODBC</th>
<th>JDBC</th>
<th>ADO.NET</th>
<th>Native</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>No</td>
<td>No</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>DB2</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Firebird</td>
<td>n/a</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>n/a</td>
</tr>
<tr>
<td>Informix</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a</td>
</tr>
<tr>
<td>iSeries</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a</td>
</tr>
<tr>
<td>MariaDB</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>MySQL</td>
<td>n/a</td>
<td>Yes*</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a</td>
</tr>
<tr>
<td>Oracle</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>n/a</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a</td>
<td>Yes</td>
</tr>
<tr>
<td>Progress</td>
<td>n/a</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>
This example shows you how create a mapping which bulk loads data from a sample `source.txt` file into a target database. The example uses SQL Server 2014 and the AdventureWorks 2014 database. The latter can be downloaded from the AdventureWorks samples page on GitHub (https://github.com/Microsoft/sql-server-samples/releases/tag/adventureworks).

<table>
<thead>
<tr>
<th>Location</th>
<th>A,15.3,39</th>
<th>B,46,34</th>
<th>C,56.33,0</th>
<th>D,0,399</th>
<th>E,0,97.43</th>
</tr>
</thead>
</table>

`source.txt`:

To define a bulk insert:

1. Set the transformation language to BUILT-IN (BUILT-IN).
2. Connect to the AdventureWorks 2014 database and add the "Production.Location" table to the mapping. For more information, see Adding Databases to the Mapping.

3. On the **Insert** menu, click **Text**, and add a source text file (such as the `source.txt` sample above) to the mapping. For more information, see CSV and Text Files. Make sure that the data types of both the source and the target components are compatible. Data types are visible on component when the **Show Data Types** toolbar button must be enabled.

Select SQL Server Database Support

<table>
<thead>
<tr>
<th>SQL Server</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>n/a</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLite</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>No</td>
</tr>
<tr>
<td>Sybase</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Teradata</td>
<td>n/a</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

* MySQL version 5 or later is required.
4. Draw the mapping connections as shown below. Note that the **Database Actions** button is now visible to the right of the table name.

5. Click the **Database Actions** button, select the **Use Bulk Transfer** check box, and click OK to confirm.
In the dialog box above, notice that the "Action on record" is "Insert All". The **Batch size** field defines the number of records to be inserted per action.

**Note:** When the **Use Bulk Transfer** option is enabled, the **Use Transactions** option becomes disabled, and vice versa. If you want to enable transaction processing, click to clear the **Use Bulk Transfer** check box.

Now that bulk insert is enabled, the next step is to execute the mapping in MapForce Server (either standalone or under FlowForce Server management). For further information, see [Compiling Mappings to MapForce Server Execution Files](#) and [Deploying Mappings to FlowForce Server](#).

### 7.2.3.10 Handling Nulls in Database Table Actions

When a mapping updates a target database by means of table actions such as "Ignore If", "Update If", "Delete If", MapForce compares the source data against the target data and generates internal database update queries as a result. These internal queries are available for preview in the **Output** pane of MapForce, see [Executing Mappings Which Modify Databases](#). The generated queries reflect the comparison conditions that were defined from the "Database Table
If your source or the target data contains nullable fields, you have two options to compare null values from the source with those in the target data:

1. treat null values as equal
2. treat null values as not equal

For this purpose, the dialog box above displays a **NULL equal** check box next to some fields. Selecting or not selecting this check box might affect the mapping result, and is the subject of this topic. Be aware that the check box can be selected only for fields which are nullable, and when at least one table action has an "equal" or "equal (ignore case)" condition.

To avoid undesired results, you should select the **NULL equal** check box if all of the following conditions are true:

1. The "Database Table Actions" dialog box contains "Ignore if", "Update if", "Delete if" actions, and
2. These actions are taken against records that may contain null values, and
3. Your mapping requires that null values in the source must be treated as equal with null values in the target.

By default, the **NULL equal** check box is not selected. If the conditions above are true and the check box is not selected, the target database table might not be updated as expected (for example, more rows would be inserted than necessary). This happens because null values affect the data comparison. For example, in the image above, let's suppose that a record has a null **email** both in the source and target. If you select the **NULL equal** check box, that record will satisfy the **Ignore If...** condition and will be ignored (skipped). However, if you don't select the **NULL equal** check box, the record will no longer satisfy the **Ignore If...** condition and will be inserted into the database.
Example

To better understand null awareness in mappings, let's analyze an example where comparison of null data occurs. This example uses a Microsoft SQL Server database; however, it is also applicable for any other supported database type. Optionally, if you have Microsoft SQL Server, you can create the tables and data used in this example by running the following database script:

```sql
<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial \CreateNullableFields.sql
```

For convenience, the database tables are illustrated below. Both tables store people data and have the same columns. Also, the column `email` can contain null data in both tables.

<table>
<thead>
<tr>
<th>id</th>
<th>firstname</th>
<th>lastname</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Toby</td>
<td>Hughey</td>
<td><a href="mailto:t.hughey@nanonull.com">t.hughey@nanonull.com</a></td>
</tr>
<tr>
<td>2</td>
<td>Mia</td>
<td>Dahill</td>
<td>NULL</td>
</tr>
<tr>
<td>3</td>
<td>Fred</td>
<td>Weinstein</td>
<td><a href="mailto:f.weinstein@nanonull.com">f.weinstein@nanonull.com</a></td>
</tr>
</tbody>
</table>

The `SOURCE` table

<table>
<thead>
<tr>
<th>id</th>
<th>firstname</th>
<th>lastname</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mia</td>
<td>Dahill</td>
<td>NULL</td>
</tr>
<tr>
<td>2</td>
<td>Fred</td>
<td>Weinstein</td>
<td><a href="mailto:f.weinstein@nanonull.com">f.weinstein@nanonull.com</a></td>
</tr>
</tbody>
</table>

The `TARGET` table

Let's suppose your task is to merge data from the `SOURCE` table into the `TARGET` table. Only the new records must be inserted into the `TARGET` table (in this example, "Tobie Hughey"). The records which exist in both tables ("Mia Dahill" and "Fred Weinstein") must be ignored.

The task can be accomplished as follows.

1. On the **Insert** menu, select **Database**. Follow the wizard steps to connect to the database (see also Connecting to a Database). When prompted to add database objects, select the table `SOURCE`.
2. On the **Insert** menu, select **Database**. Connect to the database again and add the table `TARGET` to the mapping.
3. Draw the mapping connections between the source and target components.
4. Click the **Action: Insert** button and configure the database table actions as follows:

As illustrated above, a combination of "Ignore if.. Insert Rest" actions are defined. This configuration means that, for each record, the mapping checks if:

- `firstname` in the source is equal to `firstname` in the target, AND
- `lastname` in the source is equal to `lastname` in the target, AND
- `email` in the source is equal to `email` in the target.

If all the conditions above are true, the record is ignored (according to the requirement). Otherwise, a new record is inserted into the target table. The `id` of the new record is generated by the database, while the other fields (`firstname`, `lastname`, `email`) are populated with values mapped from the source.

Importantly, the check box next to `email` enables or disables NULL-aware comparison for this field. This check box must be selected, because `email` can contain null values (namely, "Mia Dahill" has a null email address). To see the role played by this check box, try updating the database two times: first time, with the check box selected, and a second time with the cleared check box.

To update the database, click the **Output** tab and run the menu command **Output | Run SQL-Script**.

If the check box is selected, MapForce has explicit indication that you want to treat the null values as equal. Therefore, the record "Mia Dahill" is not inserted in the target table, which is the intended result.

If the check box is not selected, the record "Mia Dahill" is inserted in the target table (despite that fact that it exists already), which is not the intended result. The reason is that no explicit indication was given to MapForce that you want to treat null values as equal. A similar situation would occur if you ran the following query against the database (this query retrieves no records...
because the null value is compared with the "=" operator so it is not NULL-aware):

```
SELECT firstname, lastname, email FROM TARGET WHERE firstname = 'Mia' AND lastname = 'Dahill' AND email = NULL;
```

In order to be NULL-aware, the query above would have to be rewritten as follows:

```
SELECT firstname, lastname, email FROM TARGET WHERE firstname = 'Mia' AND lastname = 'Dahill' AND email IS NULL;
```

**Note:** The queries above are only for illustrative purposes and do not reflect the actual syntax of internal queries generated by MapForce. MapForce adapts the syntax of generated queries according to the database type (since various database vendors have different approaches to handling null comparisons).

### 7.2.3.11 Database Table Actions Settings

Whenever you create a mapping connection to a database table, a Database Actions button appears next to the affected table. Clicking this button opens Database Table Actions dialog box, from where you can configure the database insert, update, and delete actions, as well as other options.
Below is a description of the settings available on the Database Table Actions dialog box.

**SQL statement to execute before first record**

In this group box, you can define SQL statements that are executed before any actions defined under **Actions to execute for each record**. Select the desired radio button:

- **None** — No action is carried through. This is the default setting.
- **DELETE all records** — All records from the selected table are deleted before any specific table action defined in the **Actions to execute for each record** group box is performed. Activate the also delete all records in all child tables check box if you also want to get rid of the data stored in child tables of the selected table. For an example, see [Inserting Data into Multiple Linked Tables](#).
- **Custom SQL** — Write a custom SQL statement to affect the complete table. Note that support for multiple SQL statements in one query depends on the database, connection
method, and the driver used.

Actions to execute for each record

This group of settings specify the database actions that are to take place against this table when the mapping runs. To manage table actions, click the **Append Action**, **Insert Action**, or **Delete Action** buttons. Multiple actions can be defined if necessary (for example, an "Update if..." action followed by an "Insert Rest" action).

The defined table actions are processed from left to right. In the example above, the "Update if..." action is processed first. If the update condition is not satisfied then the following action is processed (in this example, the "Insert Rest" action). Note the following:

- All the defined conditions of one action must be satisfied for the table action to be executed. When this is the case, all those fields are updated where a connection exists between the source and target items on the mapping. Any subsequent table actions (to the right of an action whose condition matched) are ignored for that record.
- If the defined condition is not satisfied, then the table action is skipped, and the next action (to the right) is processed.
- If none of the conditions are satisfied, no table action takes place.

| Any table actions defined after "Insert All" or "Insert Rest" actions will never be executed, because no column conditions exist for insert actions. A dialog box appears if this is the case, stating that the subsequent table action columns will be deleted. |

The **NULL Equal** check box next to each record, where applicable, instructs MapForce that null values in the source record must be treated as equal with null values in the target record. Not selecting this check box may lead to incorrect results, see [Handling Nulls in Database Table Actions](#).

When the mapping updates a table which has foreign key relationships to other tables, the following options can be used:

<table>
<thead>
<tr>
<th>Delete data in child tables</th>
<th>This option is meaningful when you select the &quot;Update if...&quot; action for a parent table. It might be necessary if the number of records in the source file is different from the number of records in the target database, and you want to keep the database synchronized (no orphaned data in child tables). See also <a href="#">Options for Child Tables When Updating a Parent Table</a>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ignore input child data</td>
<td>Use this option when you want to update a target parent table, without affecting any of the child tables/records of that table. See also <a href="#">Options for Child Tables When Updating a Parent Table</a>.</td>
</tr>
</tbody>
</table>

For examples which illustrate various combinations of actions, see:

- [Inserting Data into a Table](#)
- [Inserting Data into Multiple Linked Tables](#)
- [Updating a Table](#)
- [Options for Child Tables When Updating a Parent Table](#)
7.2.3.12 **Example: Mapping Data from XML to SQLite**

This example walks you through the steps required to create a MapForce mapping which reads data from an XML file and writes it to a SQLite database. The example is accompanied by a sample mapping design (.mfd) file. If you want to look at the sample file before starting this example, you can open it from the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\XMLtoSQLite.mfd`.

The goal of the example is to insert data from an XML file into a SQLite database. To accomplish the goal of the example, you will need an empty SQLite database to which data will be written. As illustrated below, you can create and explore the SQLite database either with Altova DatabaseSpy, or with the command-line shell available from the official SQLite website.

**To create the SQLite database:**

If DatabaseSpy is installed on your computer (either standalone or as part of Altova MissionKit), you can create the new SQLite database as follows:

1. Run DatabaseSpy.
2. On the File menu, click **Create a Database Connection**.
3. Click **Connection Wizard**, and then click **SQLite**.
4. Click **Create a new SQLite database**, enter `c:\sqlite\articles.sqlite` as path, and then click **Connect**.
5. When prompted to set a data source name, leave the default name as is.
6. Open a new SQL editor (Ctrl+N) and run the following query against the database:

   ```sql
   create table articles (number smallint, name varchar(10), singleprice real);
   ```

Otherwise, follow the steps below to create the database:

1. Download the SQLite command-line shell for Windows from the SQLite download page ([http://www.sqlite.org/download.html](http://www.sqlite.org/download.html)) and unpack the .zip archive to a directory on your local machine (for the scope of this example, use `c:\sqlite`).
2. Run `c:\sqlite\sqlite3.exe` and enter the following statement:
create table articles (number smallint, name varchar(10), singleprice real);

This creates the table `articles` in the in-memory database. The table `articles` consists of three columns: `number`, `name`, and `singleprice`. The purpose of these columns is to store data from the elements with the same name defined in the source XML schema. Each column is declared with a data type suitable for the data expected to be stored in that column.

3. Run the command:

```
.save articles.sqlite
```

This saves the in-memory database to the current working path: `c:\sqlite\articles.sqlite`. Note that you will need to refer to this path in subsequent steps.

You have now finished creating the sample SQLite database required for this example.

To create the XML to SQLite mapping design:

1. Run MapForce and make sure that the transformation language is set to BUILT-IN (use the menu command `Output | Built-in Execution Engine`).
2. Add to the mapping area the file `Articles.xml` located in the `<Documents>`\Altova\MapForce2019\MapForceExamples\Tutorial folder (use the menu command `Insert | XML Schema/File`).
3. Add to the mapping area the database `articles.sqlite` created in previous steps (use the menu command `Insert | Database`), and then select SQLite.
4. Click `Connect`. When prompted to choose the database objects, select the `articles` table.
5. Draw the connections as shown below:

![Diagram showing connections between two components]

6. Click the A:In button on the database component and select the **Delete All records** option. This ensures that, every time the mapping is executed, all existing database rows are first deleted, in order to prevent duplication.
7. Click the **Output** tab of the main mapping window. MapForce executes the mapping using the built-in execution engine and displays the create SQL query in the Output window.

```sql
/*
The following SQL statements are only for preview and may not be executed in another SQL query tool!
To execute these statements use function 'Run SQL-script' from menu 'Output'.
Connect to database using the following connection-string:
C:/sqlite/articles.sqlite
*/

PRAGMA foreign_keys = ON;

DELETE FROM 'articles'

INSERT INTO 'articles' (number, name, singleprice) VALUES (1, 'T-Shirt', 25)

INSERT INTO 'articles' (number, name, singleprice) VALUES (2, 'Socks', 2.3)

INSERT INTO 'articles' (number, name, singleprice) VALUES (3, 'Pants', 54)

INSERT INTO 'articles' (number, name, singleprice) VALUES (4, 'Jacket', 57.5)
```

8. Run the SQL script to populate the database (use the menu command **Output | Run SQL-Script**). If MapForce does not encounter any runtime errors, the records are inserted into the SQLite database.
To check whether data was correctly inserted into the SQLite database:

1. Click the **DB Query** tab of MapForce.
2. Select the **articles** entry from the drop-down list at the top of the **DB Query** pane.
3. Enter the following query in the SQL Editor:

   ```sql
   select * from articles;
   ```

Alternatively, follows the steps below:

1. Run the file `c:sqite\sqlite3.exe` and open the database with the command:

   ```shell
   .open articles.sqlite
   ```

2. Run the following select statement:

   ```sql
   select * from articles;
   ```

Regardless of the approach you choose to select the table data (MapForce or SQLite shell), the query should return four rows now. This corresponds to the number of records in the source XML file, which was the intended goal of this example.

### 7.2.4 Joining Database Data

In mappings that read data from databases, you can join database objects such as tables or views by adding a Join component to the mapping. For example, you could combine data from
two or more tables bound by foreign key relationships, which is the typical way data is stored in relational databases. The result would be the same as if you ran against the database an SQL query where two or more tables are joined by means of an INNER JOIN operation.

Depending on the kind of data connected to the join component, the join operation can happen either in standard (non-SQL) mode, or in SQL mode. Joins in non-SQL mode are undertaken by MapForce, while joins in SQL mode are undertaken by the database from which the mapping reads data.

Joins in non-SQL mode are more flexible because they support more component types as input (for example, the join can be between tables from different databases, or between XML structures and database tables). For an example of a non-SQL join, see Example: Join XML Structures. On the other hand, a non-SQL join causes the mapping engine to perform memory-costly comparisons (because the total number of comparisons represents the cross-join, or Cartesian product, of all joined structures). Usually this process takes place very fast and is negligible in mappings which are not data-intensive; however, if the joined data sources consist of a huge number of records, then the mapping will require significant time to execute. If your mappings must process a very large number of records, consider licensing MapForce Server Advanced Edition, which includes dedicated join optimization to speed up the mapping execution.

A join in SQL mode accepts only eligible database objects as input (such as tables or views), so it is not as flexible as a non-SQL join. However, it offers better mapping performance because it is executed natively by the database. For further information, see About Joins in SQL Mode.

Note: Using a Join component is not the only way to join database tables or views. Joins applicable to databases can also be performed by using SQL SELECT statements, see SQL SELECT Statements as Virtual Tables. A major difference between SQL SELECT statements and Join components is that the former are written by hand so they might provide more flexibility. Join components are a simpler alternative if you do not feel comfortable writing SQL statements by hand.

To add a Join component:

1. Set the mapping transformation language to BUILT-IN (to do this, either click the toolbar button, or use the Output | Built-In Execution Engine menu command).
2. On the Insert menu, click Join. Alternatively, click the Join ( toolbar button. The Join component appears on the mapping. By default, it accepts data from two structures, so it has two nodes/rows inputs. If necessary, you can add new inputs to the join by clicking the Add Input ( ) button, see Joining Three or More Structures.

3. Connect the structures that are to be joined to the nodes/rows items of the join component.
4. Add the condition for the join (or multiple conditions). To do this, right-click the Join
component and select **Properties**. Join conditions can also be added directly from the mapping, by connecting the Boolean result of some function to the **condition** item of the Join component. In certain cases when database tables are joined, the join condition (or conditions) can be created automatically by MapForce. For further information, see **Adding Join Conditions**.

**Notes:**

- Join components are supported when the target language of the mapping is set to BUILT-IN. Code generation in C#, C++, or Java is not supported.
- When a structure is not a valid or supported input source for the join, MapForce displays hints either immediately directly on the mapping, or in the Messages window, when you validate the mapping (see **Validating Mappings**).
- Join components should not be connected to input parameters or results of inline user-defined functions. If such connections exist, validation errors will occur during mapping validation.
- When you connect eligible database components (such as tables or views) directly to a Join component, an **SQL mode** (SQL) button automatically appears at the top-right corner of the Join component. When enabled, this button provides special SQL features applicable to the join operation (see **About Joins in SQL Mode**).
- It is not possible to connect the output of the **joined** item to another Join component. If necessary, however, you can connect a partial result of one join to another one.

### 7.2.4.1 About Joins in SQL Mode

When you connect eligible database components (such as tables or views) directly to a join component, an **SQL mode** (SQL) button automatically appears at the top-right corner of the join component. When SQL mode is enabled, the join operation is undertaken by the database from where the mapping reads data. In other words, MapForce will internally send to the database a query with the appropriate SQL syntax to select and combine data from all tables that take part in the join. Importantly, you do not need to write any SQL; the required query is produced based on how you visually designed the Join component on the mapping, as you will see in subsequent examples.

**Note:** From a database and SQL perspective, MapForce-generated joins are always INNER joins. That is, only records which satisfy the condition in both input sets are returned by the Join component.

For SQL mode to be possible, the following conditions must be met:

1. Both objects (tables or views) that are to be joined must be from the same database.
2. Both objects that are to be joined must originate from the same MapForce component. (Note that you can quickly add/remove database objects in a component as follows: right-click the database component, and select **Add/Remove/Edit Database Objects** from the context menu.)
3. The Join condition (or conditions) must defined only from the component properties (by right-clicking the header of the join component, and selecting **Properties**), and not on the mapping (see also **Adding Join Conditions**).

**Note:** When database tables are joined in SQL mode, MapForce will create the join condition (or conditions) automatically, based on foreign key relationships detected between tables. For automatic join conditions to happen, the database tables must be in a child-parent
relationship on the MapForce component (that is, one table must be "parent" or "child" of another one on the component), see Example: Join Tables in SQL Mode.

4. All database tables must not yet be in the current target context. For example, if the mapping is designed in such a way that tables are queried by the mapping before the join operation, this could make the join impossible. For more information about how a mapping is executed, see Mapping Rules and Strategies.

You can view or control the SQL mode through the SQL (SQL) button at the top-right corner of the join component, as follows:

- SQL mode is disabled (join will be executed by MapForce (or, if applicable, by MapForce Server).
- SQL mode is enabled (join will be executed by the database).

If the SQL button is missing, this means that SQL mode is not meaningful or not supported for the data that is being joined.

In certain cases, the SQL mode must be explicitly disabled (SQL), for example:

- When your mapping requires join conditions outside of the join component properties (that is, conditions defined on the mapping and connected to the condition item of the join component).
- When you want to join tables from different databases. Use a standard (non-SQL) join if you need to join tables from different databases.

It is often the case that joined database tables or views contain identical field names in both joined structures. When SQL mode is enabled, such items appear on the component prefixed by the keyword "AS". For example, if two joined tables contain an "id" field, this field appears as "id" on the first joined table and as "id AS id2" on the second joined table. Joined tables can also produce alias names, for example, if the same table is joined to itself.

The alias field or table names are important if you need to refer to them subsequently on a mapping. For example, imagine a case when you want to filter or sort the result of the join. To achieve this, the output of the join component can be connected to a SQL WHERE/ORDER component, where you would enter the SQL WHERE and ORDER BY clauses.

To refer to a field from the WHERE clause, write the table name, followed by a dot (.) character, followed by the field name. To refer to a table alias, use the alias name as it appears on the Join component. In the ORDER BY clause, you can either use the same technique (table.field), or write just the alias field name (the name that appears after "AS").

For an example mapping which uses SQL WHERE/ORDER clauses, see Example: Join Tables in SQL Mode.

**Note:** SQL WHERE/ORDER components are not allowed between a database table and the join component; they can be added only after (but not before) a join component. For more information about SQL WHERE/ORDER components, see Filtering and Sorting Database Data (SQL WHERE/ORDER).
7.2.4.2 Example: Join Tables in SQL Mode

This example illustrates how to join data from two database tables, using a MapForce join component. The join operation is performed in SQL mode, as described in About Joins in SQL Mode. Note that joining three or more tables works in a very similar way, see also Example: Create CSV Report from Multiple Tables.

The example is accompanied by a mapping sample which is available at the following path:

<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\JoinDatabaseTables.mfd

The purpose of the mapping above is to combine data from two source database tables into a single target CSV file. As illustrated in the database diagram below, the first table (users) stores people's addresses and the second table (addresses) stores people names and email addresses. The two tables are linked by a common field (id in users corresponds to user_id in addresses). In database terminology, this kind of relation is also known as a "foreign key relationship".
For convenience, the image below illustrates the actual data in both tables.

Each user record in the `users` table can have zero or more addresses in the `addresses` table. For
example, a user may have one address of type "home", or two addresses (one of type "home" and another of type "work"), or no address at all.

The goal of the mapping is to retrieve full data (name, surname, email, city, street, number) of all users that have at least one address in the addresses table. It should also be possible to easily retrieve only addresses of a specific kind (for example, only home addresses, or only work addresses). The kind of addresses to retrieve ("home" or "work") should be supplied as a parameter to the mapping. The retrieved people records must be sorted alphabetically by last name.

The mapping requirement will be accomplished with the help of a Join component, as illustrated in the steps below.

**Note:** Using a Join component is not the only way to join database tables or views. Joins applicable to databases can also be performed by using SQL SELECT statements, see SQL SELECT Statements as Virtual Tables. A major difference between SQL SELECT statements and Join components is that the former are written by hand so they might provide more flexibility. Join components are a simpler alternative if you do not feel comfortable writing SQL statements by hand.

**Step 1: Add the source database**

1. On the Insert menu, click Database. (Alternatively, click the Insert Database toolbar button).
2. Select "SQLite" as database kind, and click Next.
3. Browse for the Nanonull.sqlite file available in the folder: <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial, and click Connect.
4. When prompted, select the addresses and users tables.
Step 2: Add the join component

1. On the Insert menu, click Join. (Alternatively, click the Join toolbar button).
2. Draw a connection from the users table to the first input of the join component.
3. Expand the users table and draw a connection from the addresses table (child of users) to the second input of join component. The button enables you to add more tables if necessary; however, in this example, only two tables are being joined.

Note: It is also possible to add the connection directly from the addresses table (the one which is not child of users); however, in this case, the join conditions would have to be defined manually, as described in Adding Join Conditions. For the purpose of this example, make sure to create the connections as shown above. This ensures the required join condition is created automatically.

4. Click the Define Join Condition button available on the join component. Notice that the join condition has been created automatically (users.id = addresses.user_id).
Step 3: Add the target CSV component

1. On the Insert menu, click Text File. (Alternatively, click the Insert Text File toolbar button).
2. When prompted to choose a text processing mode, select Use simple processing for standard CSV.
3. Click Append Field several times to create seven CSV fields. Leave all other settings as is.
4. Double-click the title cell of each field to give it a descriptive name (this will make your mapping easier to read).
5. Draw the mapping connections between the Join component and the CSV component as shown below. The connection between the joined item of the join component and the Rows item of the target component means "create as many records (rows) in the target as there are records that meet the join condition".
Step 4: Add the SQL WHERE/ORDER condition and input parameter

1. Right-click the connection between the joined item of the Join component and the Rows item of the target CSV component, and select **Insert SQL-WHERE/ORDER**.
2. Enter the WHERE and ORDER BY clauses as shown below.
3. On the mapping, add an input component (using the Insert | Insert Input menu command) and connect its output to the address_type parameter created in the previous step.

4. Double-click the input component and configure it as shown below. A design-time value is required (in this case, "home") to preview the mapping output in MapForce. If you want the preview to retrieve work addresses, replace this value with "work".
The mapping explained

The join condition automatically created in step 2 ensures that only records which satisfy the join condition `users.id = addresses.user_id` are copied to the target. The join condition was added automatically because the two tables are bound by a foreign key relationship and the mapping connections were drawn accordingly (for more information about table relationships, see Handling Database Relationships). Although not applicable to this example, it is also possible to define join conditions manually, see Example: Create CSV Report from Multiple Tables.

The two source tables are from the same database and from the same component, so this join benefits from the SQL (\*) mode. Since SQL mode is enabled, the join operation is undertaken by the database, not by MapForce. In other words, an INNER JOIN statement is generated internally by MapForce and sent to the database for execution.

The SQL WHERE/ORDER component added in step 4 enables filtering (to retrieve either home or work addresses) and sorting the recordset. Notice that the WHERE clause created a parameter `:address_type` of type `string`. This parameter makes it possible to supply the address kind (home or work) from the mapping. For more information about SQL WHERE/ORDER, see Filtering and Sorting Database Data (SQL WHERE/ORDER).

Finally, the input component makes it possible to supply the actual parameter value when the mapping runs. Note that, when the mapping runs outside MapForce (for example, when it is executed by MapForce Server on a different machine), the input must be supplied at mapping runtime as a command-line parameter, so the design-time value mentioned above is ignored. For more information, see Supplying Parameters to the Mapping.

7.2.4.3 Example: Create CSV Report from Multiple Tables

This example illustrates how to join multiple database tables for the purpose of extracting data into a single report in CSV format. The database used in this example is called `Nanonull.sqlite` and is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\`. This database stores information about a fictitious business
(which includes orders, products, users and their addresses). As is typically the case with relational databases, the information is normalized and spread across multiple tables. For example, the users table stores user personal data (which includes first name, last name, and email). The database also stores information about products ordered by users, in two different tables: orders (which includes the unique ID of the order, and the time when it took place) and orderedproducts (which includes a list of products ordered, and their quantity). Furthermore, the names of the products themselves is stored in a separate table called products.

The goal of the example is to produce a report based on data extracted from various tables, so as to make it clear who ordered certain products, when, and in which quantity. To achieve the mapping goal, follow the steps below:

1. On the Insert menu, click Database.
2. When prompted to select a database kind, click SQLite, and then click Next.
3. Browse for the Nanonull.sqlite database mentioned above, and click Connect.
4. When prompted, select the tables orderedproducts, orders, products, and users, and click OK.

5. Add a Join component to the mapping and create four nodes/rows items by clicking the Add input (a) button.
6. Connect the four tables from the database component to the corresponding input items of the Join component.
Note: In an alternative scenario, you could connect to the Join component the table orderedproducts, then the table orders (the one which is nested under it, not the one at the same level), and so on, so that all joined tables are nested under the same "root" table, see also Handling Database Relationships. The mapping result would be the same if you joined tables this way. The difference is that in this example the join conditions must be created manually, as shown below, whereas in the alternative scenario the join conditions would be created automatically by MapForce. For an example of joining tables without having to define join conditions manually, see Example: Join Tables in SQL Mode. Another mapping where all joined tables are under the same "root" table is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\DB_Denormalize.mfd`.

In this example, the tables connected to the Join component have the following order:

1. orderedproducts
2. orders
3. products
4. users

This order affects how the respective structures are displayed on the "Define Join Condition" dialog box, when you click the Define Join Condition (吸入) button. Namely, the first table (orderedproducts) appears by default under Structure 1, and the table immediately after it (orders) appears under Structure 2.

To define the first join condition, click the order_id item in the left pane and the id item in the
right pane. Now the fields `orderedproducts.order_id` and `orders.id` are paired:

So far, only two tables have been joined. To define join conditions which involve a third table, select the desired table from the drop-down list available above the right pane. The left pane displays in this case all tables that occur before it on the Join component. For example, if you select `products` on the right side, then the left side displays `orderedproducts` and `orders` (since these tables occur before `products` on the Join component). You can now pair fields of table `products` with fields of tables preceding it (in this case, `orderedproducts.product_id` and `products.id`).

To join a fourth table (`users`), select the `users` table from the drop-down list. You can now pair the fields `orders.user_id` and `users.id`.

To define a condition, select an item from each structure to compare for equality.
Now that all required join conditions have been defined, items of the Join component can be further mapped to a target component. To finish the mapping, add a CSV component (see CSV and Text Files), and connect items from the Join component to the target CSV component as illustrated below:
The mapping illustrated above produces a report (in CSV format) compiled from all four tables involved in the join, as follows:

- ID of the order (taken from the `orderedproducts` table)
- Quantity of ordered items (taken from the `orderedproducts` table)
- Time when the order took place (taken from the `orders` table)
- Name of the product ordered (taken from the `products` table)
- First name and last name of the user who ordered the product (taken from the `users` table).

### 7.2.5 Filtering and Sorting Database Data (SQL WHERE/ORDER)

When you need to filter and sort database data, use an SQL WHERE/ORDER component. This enables you to manually enter, from the MapForce graphical user interface, a SQL WHERE clause that filters data. Optionally, you can also specify an ORDER BY clause if you want to sort the recordset by a particular database field, in ascending or descending order.

The SQL WHERE/ORDER component must be connected to a table or field of a database mapping component. It is also possible to connect the SQL WHERE/ORDER to a Join component, if you need to filter the joined set or records (see [Joining Database Data](#)).
Adding a SQL WHERE/ORDER component to the mapping

1. On the **Insert** menu, click **SQL WHERE/ORDER**. By default, the SQL WHERE/ORDER component has the following structure:

   ![SQL WHERE/ORDER component](image)

   - **table/field**
   - **result**

2. Connect a source database table or field to the **table/field** item of the SQL WHERE/ORDER. For an example, open the mapping **DB_PhoneList.mfd** from the folder `<Documents>\Altova\MapForce2019\MapForceExamples`. In this mapping, the SQL WHERE/ORDER is used to filter from the source table "Person" all records where the last name begins with letter "B".

   ![Navigation panel](image)

3. Double-click the header of the SQL WHERE/ORDER component (or right-click it and select **Properties** from the context menu). This opens the "SQL WHERE/ORDER Properties" dialog box.
4. Type the SQL WHERE clause in the text box at the top. Optionally, type the ORDER BY clause. The image above illustrates the WHERE and ORDER BY clauses defined in the **DB_PhoneList.mfd** mapping (these settings are further explained below). For more examples, see Creating WHERE and ORDER BY Clauses.

**Supplying parameters to a SQL WHERE/ORDER**

The SQL WHERE/ORDER component used in the mapping **DB_PhoneList.mfd** defines a WHERE clause as follows:

```
Last LIKE :Name
```

"Last" refers to the name of a database field in the connected table. "LIKE" is an SQL operator. ":Name" creates a parameter called "Name" on the mapping.

Parameters in SQL WHERE/ORDER components are optional; they are useful if you want to pass a value to the WHERE clause from the mapping. Without parameters, the WHERE clause above could have been written as follows:

```
Last LIKE "B%"
```
This would retrieve all persons whose last name begins with letter "B". In order to make this query even more flexible, we added a parameter instead of "B%". This makes it possible to supply any other letter from the mapping (for example, "C", and thus retrieve people whose last name begins with "C" simply by changing a constant, or a mapping input parameter).

**Appearance of SQL WHERE/ORDER components**

An important thing about SQL WHERE/ORDER components is that they change appearance depending on the settings defined in them. This way you can quickly view directly from the mapping what the SQL WHERE/ORDER component does, for example:

<table>
<thead>
<tr>
<th>A WHERE clause has been defined.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A WHERE clause with a parameter has been defined. The parameter &quot;Name&quot; is visible under the &quot;table/field&quot; item.</td>
</tr>
<tr>
<td>A WHERE clause with a parameter has been defined. Additionally, an ORDER BY clause has been defined. The sorting is indicated by the A-Z sort icon.</td>
</tr>
</tbody>
</table>

Placing the mouse cursor over the SQL WHERE/ORDER header opens a tooltip displaying the various clauses that have been defined.

### 7.2.5.1 Creating WHERE and ORDER BY Clauses

After an SQL WHERE/ORDER component is added to the mapping, it needs a WHERE condition (clause) through which you specify how exactly you want to filter the data connected to the component. The WHERE condition must be entered in the "SQL WHERE/ORDER Properties" dialog box of MapForce, as shown in the previous section.

Writing a WHERE condition from MapForce is similar to writing the same SQL clause outside MapForce. Use the syntax applicable to the SQL dialect of the corresponding database. For example, you can use operators, wildcards, as well as sub-selects or aggregate functions. To create parameters that you can pass from the mapping, enter a semi-colon character (:) followed by the parameter name.

**Note:** When you finish writing the WHERE clause and click OK, MapForce validates the integrity of the final SQL statement. A dialog box prompts you if there are syntax errors.

The table below lists some typical operators that can be used in the WHERE clause:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal</td>
</tr>
<tr>
<td>Operator</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than/equal</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than/equal</td>
</tr>
<tr>
<td>IN</td>
<td>Retrieves a known value of a column</td>
</tr>
<tr>
<td>LIKE</td>
<td>Searches for a specific pattern</td>
</tr>
<tr>
<td>BETWEEN</td>
<td>Searches between a range</td>
</tr>
</tbody>
</table>

Use the % (percentage) wildcard to represent any number of characters in a pattern. For example, to retrieve all records ending in "r" from a field called lastname, use the following expression:

```
lastname = "%r"
```

When querying databases that support storing and querying of XML database data (for example, IBM DB2, Oracle, SQL Server), you can use XML functions and keywords applicable to that particular database, for example:

```
xmlexists('$c/Client/Address[zip">55116"]' passing USER.CLIENTS.CONTACTINFO AS "c")
```

See also Example: Extracting Data from IBM DB2 XML Type Columns.

Optionally, if you want to sort the retrieved recordset by a particular field, add an ORDER BY clause in the corresponding text box of the "SQL WHERE/ORDER Properties" dialog box. To sort by multiple fields, separate the field names by commas. To change the sort order, use the ASC and DESC keywords. For example, the following ORDER BY clause retrieves records ordered by lastname, and then by firstname, in descending order:

```
lastname, firstname DESC
```

**Example 1**

The following WHERE condition is attached to the Person table of the altova.mdb database component. It retrieves those records where First and Last are greater than the letter "C". In other words, it retrieves all names from "Callaby" onwards.

```
First > "C" AND Last > "C"
```

Note how the connections are placed:

- The connection to table/field originates in the table that you want to query, "Person" in this case.
- The result output is connected to a "parent" item of the fields that are queried/filtered, in
Example 2
The following WHERE condition creates a parameter Name which then appears in the SQL WHERE/ORDER component on the mapping.

```
Last LIKE :Name
```

The constant component %S supplies the value of the Name parameter. The wildcard % denotes any number of characters. This causes the mapping to search for a pattern in the column "Last" (all last names ending in "S").

Example 3
The following WHERE condition creates two parameters, PhoneUpper and PhoneLower, to which
the current values of PhoneExt are compared. The upper and lower values are supplied by two constant components shown in the diagram below.

PhoneExt < :PhoneUpper and PhoneExt > :PhoneLower

The WHERE condition in this example could also be written using the BETWEEN operator:

PhoneExt BETWEEN :PhoneUpper and :PhoneLower

7.2.6 SQL SELECT Statements as Virtual Tables

MapForce supports the creation of SQL SELECT statements with parameters in database components. These are table-like structures that contain the fields of the result set generated by the SELECT statement. These structures can then be used as a mapping data source, like any table or view defined in the database.

- When using Inner/Outer joins in the SELECT statement, fields of all tables are included in the component.
- Expressions with correlation names (using the SQL "AS" keyword) also appear as a mappable items in the component.
- Database views can also be used in the FROM clause.
- SELECT statements can contain parameters which use the same syntax as the SQL WHERE/ORDER component.

Once the SELECT statement has been added to a database component, the fields returned by it are available for mapping, for example:
The number of visible lines of the SELECT statement is configurable. To define the number of lines you want to see on the component, select the menu command Tools | Options, click the General tab and enter the number of lines in the Mapping View group.

### 7.2.6.1 Creating SELECT Statements

You can create SELECT statements on any mapping which contains a database component. If your mapping does not contain a database yet, add a database first (see Connecting to a Database). For the scope of this example, select the menu command Insert | Insert Database and follow the wizard steps to connect to the *altova-products.mdb* file available in the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\` folder.

To create a SELECT statement:

1. Right-click the title of the database component, and select Add/Remove/Edit Database Objects. (As an alternative, select the database component, and then select the menu command Component | Add/Remove/Edit Database Objects).
2. Do one of the following:
   o To generate the SELECT statement from an existing table, right-click any table and
     select **Generate and add an SQL statement** from the context menu. You will be able to edit
     the generated statement afterwards.
   o To write a custom SELECT statement, click the **Add/Edit SELECT Statement**
     button.

3. Edit or create the statement as required. For example, the SELECT statement below is
   valid for the `altova-products.mdb` file available in the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\` folder. The **Price** field is the product of the two fields, **Quantity** and **UnitPrice**, and is declared as a correlation name (**AS** **Price**).

   ```sql
   SELECT *, (Quantity*UnitPrice) AS Price
   From Orders
   INNER JOIN Products
   ON Orders.ProductID = Products.ProductID
   Where Orders.Quantity > 2
   ```

4. Click **Add SELECT Statement**. Notice that the SELECT statement is now visible as a
   database object, similar to how tables, views, and procedures are visible.
5. Click OK. The SELECT statement is also displayed on the database component, and you can map data from any of the fields returned by the SELECT query.

Important notes:

- All calculated expressions in the SELECT statement must have a unique correlation name (like "AS Price" in this example) to be available as a mappable item.
- If you connect to an Oracle or IBM DB2 database using JDBC, the SELECT statement must have no final semicolon.
To remove a previously added SELECT statement:

1. Right-click the title of the database component, and select Add/Remove/Edit Database Objects.
2. Right-click the SELECT statement you want to delete, and select Remove Select Statement.

7.2.6.2 Example: SELECT with Parameters

This example shows you how to create a MapForce mapping which reads data from a Microsoft Access database and writes it to a CSV file. In particular, the mapping described in this example uses a custom database SELECT query with a parameter. The SELECT statement combines data from multiple tables. Then, the results are supplied to the mapping for further processing.

The example is accompanied by a mapping design (.mfd) file available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\select-component.mfd`. You might want to open this sample file and analyze it first, or follow the steps below to create it from scratch.

Although this example uses a Microsoft Access database, the process works in the same way for other database types. For information about connecting to other databases, see Connecting to a Database.

The goals are as follows:

1. We must select from the database only those orders where the number of ordered items exceeds a custom value. This custom value should be supplied as a parameter to the mapping. To achieve this goal, we will create a custom database SELECT statement that takes an input parameter.
2. In the Access database, the date format is `YYYY-MM-DD HH-MI-SS`. In the CSV file, the time part must be stripped, so the format should be `YYYY-MM-DD`. To achieve this goal, we use the date-from-datetime function available in MapForce.
3. The resulting CSV file must have the name `OrdersReport.csv`.

Step 1: Add the SELECT structure

1. On the Insert menu, click Database.
2. Select Microsoft Access (ADO), and follow the wizard steps to connect to the `altova-products.mdb` file available in the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\` folder.
3. On the Insert Database Objects dialog box, click Add/Edit SELECT Statement, and enter the following query:

```sql
SELECT *, (Quantity * UnitPrice) AS Price
FROM Orders
INNER JOIN Products
ON Orders.ProductID = Products.ProductID
WHERE Orders.Quantity > :Quantity
```

This query uses a join between the Orders and Products tables, and retrieves all fields (*), and a computed value (AS Price). The query also specifies the :Quantity parameter in the WHERE clause.
4. Click **Add SELECT statement**.

5. Click **OK**. The `altova-products` component has now been added to the mapping area.

6. On the `altova-products` component, click `Edit` and select **Insert Call with Parameters**.
A new structure (SELECT_Statement) is now available on the mapping. It is split into two parts: the left part supplies input connectors and the right part supplies output connectors. Notice that the left part also includes the Quantity parameter defined previously.

Step 2: Add the input parameter

1. On the Insert menu, click Insert Input.
2. Type "Quantity" as name.
3. Under Design-time Execution, enter a parameter value to be used for executing the mapping during the design phase (in this example, "2"). For more information, see Supplying Parameters to the Mapping.
You can now connect the input parameter to the database call structure, as shown below.

Step 3: Add the target CSV component

1. On the **Insert** menu, click **Text File**.
2. Select **Use simple processing for standard CSV...**, and then click **Continue**.
3. On the Component Settings dialog box, click **Append Field** and add nine new fields. It is recommended that you give to the CSV fields the same name as the name of the database fields, as shown below. This will help you save time later when drawing mapping connections. For more information about these settings, see **Setting the CSV Options**.

4. Create a connection between the **result** node of the SELECT structure and the **Rows**
Because most of the fields in the CSV component have the same name as their database equivalent, mapping connections will likely be drawn automatically when you connect `result` to `Rows`. If this does happen, select the `Connection` menu and make sure that the `Auto Connect Matching Children` option is enabled. The only mapping item that you have to connect manually is `ProductID`, since there is no field with this name in the SELECT structure.

**Step 4: Convert the date**

In the Libraries window, search for the `date-from-datetime` function and drag it to the mapping area. Then connect its input and output as shown below.

**Step 5: Set the name of the output file**

To set the name of the output file to `OrdersReport.csv`, double-click the CSV component, and enter the value in the Output File box.
7.2.7 Mapping XML Data to / from Database Fields

MapForce enables you to map data to or from database fields (columns) that store XML content. This means that XML data stored by the database field (column) can be extracted and written to any other structure supported by MapForce, and the other way round. You can map data as follows:

1. To or from fields of a dedicated XML type (for example, `Xml` in SQL Server, `XMLType` in Oracle). Reading or writing XML to/from dedicated XML fields is applicable to databases that have native support for XML (such as IBM DB2, Oracle, and SQL Server).
2. To or from text fields storing XML content (for example, `Text`, `Varchar`). This applies to any database where the text field has sufficient length to store an XML document.

In either of the cases, a valid XML schema must exist for each database column to/from which you want to map data. When a database column stores XML, MapForce provides you with the choice to assign an XML schema directly from the database (if supported by the database), or select a schema from an external file. You can assign one XML schema per database column. If the schema has multiple root elements, you can select a single root element of that schema.

When XML is stored as a string field in a database, the character encoding of the XML document is that of the underlying string field. If the database field does not store text as Unicode, some characters cannot be represented.

Some databases support XML encoding for XML fields (which may not necessarily be the same as that of the database character set). If supported by the database, the XML document encoding declaration is assumed to be the one declared in the XML field. For information about the XML encoding support provided by various databases, refer to their documentation.

7.2.7.1 Assigning an XML Schema to a Database Field

This topic illustrates how to assign a schema to a field that is natively defined as XML type in the database. The instructions below use SQL Server 2014 and the AdventureWorks 2014 database. The latter can be downloaded from the AdventureWorks samples page on GitHub (https://github.com/Microsoft/sql-server-samples/releases/tag/adventureworks). Note that mapping of data to or from XML fields works in the same way with other database types that support XML fields.
To add the Adventure Works 2014 database as a mapping component:

1. On the **Insert** menu, click **Database**, and follow the wizard to connect to the database using your preferred method (ADO or ODBC). For more information, see Connecting to Microsoft SQL Server (ADO) and Connecting to Microsoft SQL Server (ODBC). NOTE: If you use the SQL Server Native Client driver, you might need to set the Integrated Security property to a space character (see Setting up the SQL Server Data Link Properties).

2. On the **Insert Database Object** dialog box, expand the **Production** schema, and then select the **ProductModel** table.

3. Click OK.

The database table has now been added to the mapping area. Notice that this table has two fields of XML type: **CatalogDescription** and **Instructions**.

For the structure of the XML fields to appear on the mapping, the XML schema of the field content is required. Right-click the **Instructions** field and select **Assign XML Schema to Field** from the context menu.
In this particular example, you will assign a schema to the Instructions field directly from the database. To do this, select the Production.ManuInstructionsSchemaCollection item next to the Database option, and then click OK.

The structure of the XML field now appears on the component. You can now draw connections.
(and map data) to or from this field.

7.2.7.2 Example: Writing XML Data to a SQLite Field

This example walks you through the steps required to create a MapForce mapping which reads data from multiple XML files and writes it to a SQLite database. The goal of the mapping is to create, for each source XML file, a new database record in the SQLite database. Each record will store the XML document as a TEXT field.

All the files used in this example are available at the following path: <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial. The file names are as follows:

<table>
<thead>
<tr>
<th>The mapping design file</th>
<th>XmlToSqliteField.mfd</th>
</tr>
</thead>
<tbody>
<tr>
<td>The source XML files</td>
<td>bookentry1.xml</td>
</tr>
<tr>
<td></td>
<td>bookentry2.xml</td>
</tr>
<tr>
<td></td>
<td>bookentry3.xml</td>
</tr>
<tr>
<td>The XML schema used for validation</td>
<td>books.xsd</td>
</tr>
<tr>
<td>The target SQLite database</td>
<td>Library.sqlite</td>
</tr>
</tbody>
</table>
To achieve the goal of the mapping, the following steps will be taken:

1. Add the XML component and configure it to read from multiple files.
2. Add the SQLite database component and assign an XML schema to the target TEXT field.
3. Create the mapping connections and configure the database INSERT action.

Step 1: Add the XML component

1. On the Insert menu, click XML Schema/File and browse for the books.xsd schema located in the <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ directory. When prompted to supply a sample XML file, click Skip. When prompted to select a root element, select Books.
2. Double-click the component header and type bookentry*.xml in the Input XML File box. This instructs MapForce to read all XML files whose name begins with "bookentry-" in the source directory. For more information about this technique, see Processing Multiple Input or Output Files Dynamically.

Step 2: Add the SQLite component

On the Insert menu, click Database, and follow the wizard to connect to the Library.sqlite database file from the <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ directory (see also Connecting to an Existing SQLite Database). When prompted to select the database objects, select the BOOKS table.

The database field where XML content will be written is called metadata. To assign an XML schema to this field, right-click it and select Assign XML Schema to Field from the context menu.
In this tutorial, the schema assigned to the `metadata` field is the same one used to validate the source XML files. Click **Browse** and select the `books.xsd` schema from the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\` directory:

The `books.xsd` schema has two elements with global declaration: `book` and `books`. In this example, we will set `book` as the root element of the XML written to the database field. Click **Choose**, and select `book` as root element:

**Step 3:** Create the mapping connections and configure the database `INSERT` action
Create the mapping connections as follows:
As shown above, the connection from book to book is a "Copy-All" connection, since both the source and target use the same schema and the names of child elements are the same. For more information about such connections, see Copy-all connections.

The topmost connection (books to BOOKS) iterates through each book element in the source and writes a new record in the BOOKS table. Click the A:In button on the database component and set the database update settings as shown below:

The DELETE all records option instructs MapForce to delete the contents of the BOOKS table before inserting any records.

The Insert All actions specify that a database INSERT query will take place. The field id is generated from the database itself, while the field metadata will be populated with the value provided by the mapping.

Make sure to save the mapping before running it.

To run the mapping and view the generated output, click the Output tab. Note that this action does not update the database immediately. When you are ready to run the generated database
script, select the menu command **Output** | **Run SQL Script** (or click the toolbar button).

### 7.2.7.3  Example: Extracting Data from IBM DB2 XML Type Columns

This example illustrates how to extract data from IBM DB2 database columns of XML type and write it to a target CSV file. It also illustrates how to use XQuery statements embedded into SQL in order to retrieve XML content conditionally. The example requires access to an IBM DB2 database where you have permission to create and populate tables.

First, let's prepare the database so that it actually contains XML data. This can be done either in a database administration tool specific to your database, or directly in MapForce. To do this directly in MapForce, follow the steps below:

1. Create a new mapping and click the **DBQuery** tab.
2. Click **Quick Connect** and follow the wizard steps to create a new database connection (see also Database Connection Examples).
3. Paste the following text into the SQL Editor. This SQL query creates a database table called **ARTICLES** and populates it with data.

```sql
-- Create the table
CREATE TABLE ARTICLES (
  id INTEGER NOT NULL,
  article XML
);
-- Populate the table
INSERT INTO ARTICLES VALUES
(1, '<Article>
  <Number>1</Number>
  <Name>T-Shirt</Name>
  <SinglePrice>25</SinglePrice>
</Article>'),
(2, '<Article>
  <Number>2</Number>
  <Name>Socks</Name>
  <SinglePrice>230</SinglePrice>
</Article>'),
(3, '<Article>
  <Number>3</Number>
  <Name>Pants</Name>
  <SinglePrice>34</SinglePrice>
</Article>'),
(4, '<Article>
  <Number>4</Number>
  <Name>Jacket</Name>
  <SinglePrice>5750</SinglePrice>
</Article>');</s
```

4. Click the **Execute** button. The query execution result is displayed in the Query Results window. If the query is executed successfully, four rows are added to the newly created table.

Next, we will create a mapping which retrieves XML data from the **ARTICLES** table created above.
conditionally. The goal is to retrieve from the ARTICLES column only articles with a price greater than 100.

Step 1: Add the database
1. Click the **Mapping** tab to switch back to the mapping pane.
2. On the **Insert** menu, click **Database**, and follow the wizard steps to connect to the database.
3. When prompted to select the database objects, select the **ARTICLES** table created previously.

![Connection to ARTICLES table](image)

Step 2: Assign the schema to the XML type field
1. Right-click the **ARTICLE** item of the component, and select **Assign XML Schema to field** from the context menu.

![Assign XML Schema to Field](image)

2. Select **File**, and browse for the following schema: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\DB2xsd.xsd`.

Step 3: Add the SQL WHERE/ORDER component
1. On the **Insert** menu, click **SQL WHERE/ORDER**.
2. Connect the **ARTICLE** XML type column to the input of the SQL WHERE/ORDER.
3. In the SQL-WHERE/ORDER Properties dialog box, enter the following text:

```sql
XML</code><code>EXIST</code><code>('<$a/Article[SinglePrice>100]' PASSING ARTICLE as "a")
```

The text above represents the "WHERE" part of the SQL query. At mapping runtime, it will be combined with the "SELECT" part displayed on the dialog box. This statement uses the `XML</code><code>EXIST</code><code>` function and syntax specific to IBM DB2 databases.

**Step 4: Add the target CSV file**

1. On the **Insert** menu, click **Text File**.
2. When prompted, select **Use simple processing for standard CSV...** , and click **Continue**.
3. Click **Append Field** three times to add three fields which will store the article number, name, and price, respectively. Leave all other settings as is.
4. Draw the mapping connections as shown below.
Data Sources and Targets

You can now preview the mapping result, by clicking the **Output** tab. As expected, only articles with price greater than 100 are shown in the output.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2, Socks, 230</td>
</tr>
<tr>
<td>2</td>
<td>4, Jacket, 575</td>
</tr>
</tbody>
</table>

### 7.2.8 Browsing and Querying Databases

MapForce has a dedicated Database Query pane (also called **DB Query**) that allows you to query a database independently of the mapping process. Such direct queries are not saved together with the mapping *.mfd* file but provide a convenient way to browse or modify the contents of a database directly from MapForce.

A separate **DB Query** pane exists for each currently active mapping. You can create multiple active connections, to different databases, within each **DB Query** pane. Note that the connections created from the **DB Query** pane are not part of the mapping and thus not preserved after you close MapForce, unless you define them as Global Resources (see **Global Resources**).
The Database Query pane consists of the following parts:

- **Database Browser**, which displays connection info and database tables
- **SQL Editor**, in which you write your SQL queries
- **Results tab**, which displays the query results in tabular form
- **Messages tab**, which displays warnings or error messages.

The upper area of the Database Query pane contains the connection controls allowing you to define the working databases, as well as the connection and database schemas.

### 7.2.8.1 Selecting or Connecting to a Database

For each database that you want to query, a database connection must be created. If your mapping already includes a database component, you can select the existing database connection from the upper area of the **DB Query** pane (by default, the connection is “Offline”) and start exploring the database objects and run queries.

If your mapping does not include any database component, or if you want to connect to a new database, click **Quick Connect** ( ) and follow the wizard steps to create a new database connection (see **Examples** ). You can also select an existing database connection from Global Resources, if one has been defined as such (see **Global Resources** ).

Once you are connected to the database, you can create database queries using one of the following methods:
• Import the SQL query into the SQL Editor pane from an existing SQL file.
• Write the query in the SQL Editor pane.
• Right-click an object in the Database Browser pane and generate a query (typically, SELECT).

When you are ready to run the query displayed in the SQL Editor pane, click the **Execute** button. The database data is retrieved and displayed in the Results tab in tabular form. Note that the status bar displays the "Finished Retrieval" message, and other pertinent information about the query results.

Once the "Finished Retrieval" message is displayed, you can search, sort, or copy to clipboard the search results (see Copying, Sorting, and Searching the Query Results).

7.2.8.2 **Creating and Editing SQL Statements**

The SQL Editor is used to write and execute SQL statements. It displays any SQL statements that you may have generated automatically, loaded from existing SQL scripts, or written manually. The SQL Editor supports autocompletion (see Auto-Completion), regions, and line or block comments.

The SQL Editor toolbar provides the following buttons:

- **Toggle Browser**: Toggles the Browser pane on and off.
- **Toggle Result**: Toggles the Result pane on and off.
- **Execute (F5)**: Clicking this button executes the SQL statements that are currently selected. If multiple statements exist and none are selected, then all are executed.
- **Undo**: Allows you to undo an unlimited number of edits in the SQL window.
- **Redo**: Allows you to redo previously undone commands. You can step backward and forward through the undo history using both these commands.
- **Import SQL file**: Opens an SQL file in the SQL Editor, which can then be executed.
- **Export SQL file**: Saves SQL queries for later use.
7.2.8.2.1 Generating SQL Statements

SQL statements can be generated automatically from the Database Browser, loaded from scripts, or entered manually.

To generate SQL SELECT statements from the Database Browser, do one of the following:

- Click a database object (such as a table or view), or a folder, in the Database Browser and drag it into the SQL Editor.

- Right-click a database object in the Database Browser and select Show in SQL Editor | Select.

To create SQL statements manually:

1. Start entering the SQL statement in the SQL Editor. If autocompletion is set to occur automatically, a drop-down list with suggestions appears while you enter statement.

2. Use the cursor Up and Down keys to select a suggestion, and then press Enter to insert the highlighted option (see also SQL Auto-Completion Suggestions).
7.2.8.2.2 Executing SQL Statements

The SQL statements that appear in the SQL Editor can be executed against the database, with immediate effect. The result of the SQL query and the number of affected rows is displayed in the Messages pane of the DB Query pane.

When multiple SQL statements appear in the SQL Editor, only the selected statements will be executed. You can select individual SQL statements as follows:

- Holding the left mouse button clicked, drag the cursor over a specific statement.
- Click a line number in the SQL Editor.
- Triple-click a specific statement.

To execute a SQL statement:

1. Enter or select the SQL statement in the SQL Editor (see Generating SQL Statements).
2. Click the Execute button.

7.2.8.2.3 Importing and Exporting SQL Scripts

You can save any SQL that appears in an SQL Editor window to a file and re-use the script file later on.

To export the contents of the SQL Editor pane to a file:

- Click Export SQL file, and enter a name for the SQL script.

To import a previously saved SQL file:

- Click Import SQL file, and select the SQL file you want to open.

7.2.8.2.4 Adding and Removing SQL Comments

The SQL Editor allows you to comment out statements, parts of statements, or groups of statements. These statements, or the respective parts of them, are skipped when the SQL script is being executed.

To comment out a section of text:

1. Select a statement or part of a statement.
2. Right-click the selected statement and select Insert / Remove Block Comment.
To comment out text line by line:

- Right-click at the position you want to comment out the text and select **Insert / Remove Line Comment**. The statement is commented out from the current position of the cursor to the end of the statement.

To remove a block comment or a line comment:

1. Select the part of the statement that is commented out. If you want to remove a line comment, it is sufficient to select only the comment marks `--` before the comment.
2. Right-click and select **Insert / Remove Block (or Line) Comment**.

### 7.2.8.2.5 Using Bookmarks

Bookmarks are used to mark items of interest in long scripts.

To add a bookmark:

- Right-click the line you want to have bookmarked and select **Insert/Remove Bookmark** from the context menu.

```sql
/*
SELECT [PrimaryKey], [ForeignKey], [city], [state], [street], [zip] FROM [Address];
SELECT [PrimaryKey], [Name] FROM [Altova];
SELECT [PrimaryKey], [ForeignKey], [Name] FROM [Department];*/

SELECT [PrimaryKey], [ForeignKey], [Desc], [EMail], [Established], [Fax], [Name], [Phone] FROM [Office];
SELECT [PrimaryKey], [ForeignKey], [EMail], [First], [Last], [PhoneExt], [Title] FROM [Person];
```
A bookmark icon is displayed in the margin at the beginning of the bookmarked line.

**To remove a bookmark:**

- Right-click the line from where you want to remove the bookmark and select **Insert/Remove Bookmark** from the context menu.

**To navigate between bookmarks:**

- To move the cursor to the next bookmark, right-click and select **Go to Next Bookmark**.
- To move the cursor to the previous bookmark, right-click and select **Go to Previous Bookmark**.

**To remove all Bookmarks:**

- Right-click and select **Remove all Bookmarks**.

### 7.2.8.2.6 Inserting Regions

Regions are sections of text that you mark and declare as a unit to structure your SQL scripts. Regions can be collapsed and expanded to display or hide parts of SQL scripts. It is also possible to nest regions within other regions.

When you insert a region, an expand/collapse icon and a **--region** comment are inserted above the selected text.

**Note:** You can change the name of a region by appending descriptive text to the **--region** comment. The word "region" must not be deleted, e.g. **--region DB2query**.

**To create a region:**

1. In the SQL Editor, select the statements you want to make into a region.
2. Right-click and select **Add Region** from the context menu. The selected statements become a region which can be expanded or collapsed.

```sql
1  SELECT [PrimaryKey], [ForeignKey], [city], [state],
    [street], [zip] FROM [Address];
2  -- region
3  SELECT [PrimaryKey], [Name] FROM [Altova];
4  SELECT [PrimaryKey], [ForeignKey], [Name] FROM [Department];
5   -- endregion
6  SELECT [PrimaryKey], [ForeignKey], [Desc], [EMail],
    [Established], [Fax], [Name], [Phone] FROM [Office];
7  SELECT [PrimaryKey], [ForeignKey], [EMail], [First],
    [Last], [PhoneExt], [Title] FROM [Person];
```
3. Click the + or - box to expand or collapse the region.

To remove a region:

- Delete the -- region and -- endregion comments.

7.2.8.3 Browsing Database Objects

When you are connected to one or several databases, the Database Browser pane gives a full overview of the objects in each database, including tables, views, procedures, and so on, up to the most detailed level. For databases with XML support, the Database Browser additionally shows registered XML schemas in a separate folder.

For custom navigation through database objects, the Database Browser pane includes several predefined database display layouts. The predefined layouts are available in the top area of the Database Browser.

To select a layout, click the Folders Layout drop-down button and select an entry from the list. Note that the button changes with the selected layout.

- The Folders layout organizes database objects into folders based on object type in a hierarchical tree, this is the default setting.
- The No Schemas layout is similar to the Folders layout, except that there are no database schema folders; tables are therefore not categorized by database schema.
- The No Folders layout displays database objects in a hierarchy without using folders.
- The Flat layout divides database objects by type in the first hierarchical level. For example, instead of columns being contained in the corresponding table, all columns are displayed in a separate Columns folder.
- The Table Dependencies layout categorizes tables according to their relationships with other tables. There are categories for tables with foreign keys, tables referenced by foreign keys and tables that have no relationships to other tables.

In addition to layout navigation, you can use the Database Browser for the following tasks:

- Generate SQL statements (see Generating SQL Statements).
- Filter and search the displayed database objects (see Filtering and Searching Database Objects).
- Sort the tables into "System" and "User" tables.
- Refresh the root object of the active data source.

**To sort tables into User and System tables:**

- In the **Database Browser**, right-click the "Tables" folder, and then select **Sort into User and System Tables**.

  **Note:** This function is available when one of the following layouts is selected: **Folders**, **No Schemas** or **Flat**.

**To refresh the root object of the active data source:**

- At the top of the **Database Browser**, click **Refresh ( )**.

### 7.2.8.3.1 Filtering and Searching Database Objects

You can filter any database objects (schemas, tables, views, etc) displayed in the **Database Browser** by name or part of a name. Objects are filtered as you type in the characters. Filtering is case-insensitive by default. Filtering is not supported if you have selected the "No Folders" layout.

**Filtering database objects**

1. At the top of the Database Browser, click **Filter Folder contents ( )**. Filter icons appear next to all folders in the currently selected layout.

2. Click the filter icon next to the folder you want to filter, and select the filtering option from the context menu (for example, **Contains**).
3. In the empty field which appears next to the filter icon, enter the search text (for example, "G"). The results are adjusted as you type.

![Database Browser Screenshot]

Searching database objects
To find a specific database item by its name, you can either use filtering functions or the Object Locator. To find database elements using the Object Locator:

1. At the top of the Database Browser, click Object Locator ( ).
2. In the drop-down list that appears, enter the search text (for example, "Off").
3. Click an object in the list to select it in the Database Browser.

7.2.8.3.2 Context Options in Database Browser

The context menu options available in the Database Browser depend on the object you have selected, for example:

- Right-clicking the "root" object allows you to Refresh the database.
- Right-clicking a folder always presents the same choices: Expand | Siblings | Children and Collapse | Siblings | Children.
Right-clicking a database object reveals the **Show in SQL Editor** command and the submenu items discussed below.

To select multiple database objects, press either **Shift + Click** or **Ctrl + Click**.

**Note:** The syntax of the SQL statements may vary depending on the database you are using. The syntax below applies to Microsoft SQL Server 2014.

The following options are available under the **Show in SQL Editor** context menu for the root object:

- **CREATE**: Creates a CREATE statement for the selected database root object, for example:
  ```sql
  CREATE DATABASE [MYDB]
  ```

- **DROP**: Creates a DROP statement for the selected database root object, for example:
  ```sql
  DROP DATABASE [MYDB]
  ```

The following options are available under the **Show in SQL Editor** context menu for tables and views:

- **SELECT**: Creates a SELECT statement that retrieves data from all columns of the source table, for example:
  ```sql
  SELECT [DepartmentID], [Name], [GroupName], [ModifiedDate] FROM [MYDB].[HumanResources].[Department]
  ```

- **Name**: Returns the name of the table.

- **Path**: Returns the full path of the tables, in the format `DataSourceName.DatabaseName.SchemaName.TableName`.

  If you selected multiple tables, the names or paths are printed on separate lines, separated by commas.

The following options are available under the **Show in SQL Editor** context menu for columns:

- **SELECT**: Creates a SELECT statement that retrieves data from the selected column(s) of the parent table, for example:
  ```sql
  SELECT [DepartmentID] FROM [MYDB].[HumanResources].[Department]
  ```

- **Name**: Returns the name of the selected column.

- **Path**: Returns the full path of the column, in the format `DataSourceName.DatabaseName.SchemaName.TableName.ColumnName`.

  If you selected multiple columns, the names or paths are printed on separate lines, separated by commas.

The following options are available under the **Show in SQL Editor** context menu for constraints:

- **Name**: Returns the name of the selected constraint.

- **Path**: Returns the full path of the constraint, in the format `DataSourceName.DatabaseName.SchemaName.TableName.ConstraintName`.

  If you selected multiple constraints, the names or paths are printed on separate lines, separated by commas.

The following options are available under the **Show in SQL Editor** context menu for indexes:

- **Name**: Returns the name of the selected index.

- **Path**: Returns the full path of the index, in the format

  `DataSourceName.DatabaseName.SchemaName.TableName.IndexName`
If you selected multiple indexes, the names or paths are printed on separate lines, separated by commas.

If the database has support for XML Schemas, the following options are available for every schema displayed under the "XML Schemas" folder:

- **View in XMLSpy**: Opens the database schema in XMLSpy, provided that the latter is installed.
- **Manage XML Schemas**: Opens a dialog box where you can add new or drop existing database XML schemas.

### 7.2.8.4 Copying, Sorting, and Searching the Query Results

The **Results** tab of the **DB Query** pane shows the recordset retrieved as a result of a database query.

![Database query results](image)

The toolbar buttons enable navigation between results and SQL statements and facilitate searching within the query results.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Find</strong></td>
<td>Searches a specific text within the displayed results. Press F3 to go to the next occurrence of the search term.</td>
</tr>
<tr>
<td><strong>Go to statement</strong></td>
<td>Jumps to the SQL Editor and highlights the SQL statement that produced the current result. This might be particularly useful when the SQL Editor contains multiple statements.</td>
</tr>
</tbody>
</table>

**To select cells from the query results:**

- Click a column header to select the entire column
- Click a row number to select the entire row
- Click individual cells. Holding down the Ctrl key while clicking allows you to make multiple selections. If a column or cell contains XML data then this data can also be copied.

Note: The context menu can also be used to select data, **Selection | Row | Column** |
To copy the selected cells to clipboard:

- Right-click and select **Copy selected cells** from the context menu.

To sort data:

- Right-click anywhere in the column to be sorted and select **Sorting | Ascending** or **Descending**
- Click the sort icon in the column header

The data is sorted according to the contents of the sorted column.

To restore the default sort order:

- Right-click anywhere in the table and choose **Sorting | Restore default** from the context menu.

7.2.8.5 Viewing the Status of Executed Queries

The **Messages** tab of the **DB Query** pane provides specific information about the last executed SQL statement and reports errors or warning messages.

You can use different filters to customize the view of the **Messages** tab or use the **Next** and **Previous** buttons to browse data row by row. The buttons at the top are used to navigate the messages, copy text to clipboard, and hide certain parts of the message. These options are also available in the context menu, when you right-click anywhere inside the **Messages** tab.

**Filter:** Opens a pop-up menu from where you can filter out the individual message types (**Summary**, **Success**, **Warning**, **Error**, **Autoinsertion**, **Progress**). "Autoinsertion" refers to those messages that may be triggered when SQL statements or SQL
constructs are inserted automatically in SQL Editor. "Progress" messages report the database connection result, as well as the outcome of SQL parsing and data structure loading.

You can check all or none of these options with a single mouse click by selecting either Check All or Uncheck All from the pop-up menu.

| Next: Jumps to and highlights the next message. |
| Previous: Jumps to and highlights the previous message. |
| Copy selected message to the clipboard |
| Copy selected message including its children to the clipboard |
| Copy all messages to the clipboard |
| Find: Opens the Find dialog box. |
| Find previous: Jumps to the previous occurrence of the string specified in the Find dialog box. |
| Find next: Jumps to the next occurrence of the string specified in the Find dialog box. |
| Clear: Removes all messages from the Message tab of the SQL Editor window. |

7.2.8.6 Database Query Settings

This section includes information about configuring miscellaneous settings applicable to SQL statements entered or loaded in SQL Editor, as well as the query results displayed after a query is executed.

7.2.8.6.1 SQL File Encoding Settings

You can specify the encoding options for SQL files created or opened with SQL Editor as follows:

1. Click the DB Query tab.
2. At the top of the pane, click Options and then click General | Encoding.
Default encoding for new SQL files

Define the default encoding for new files so that each new document includes the encoding specification that you specify here. If a two- or four-byte encoding is selected as the default encoding (for example, UTF-16, UCS-2, or UCS-4), you can also choose between little-endian and big-endian byte ordering for the SQL files.

The encoding of existing files is not affected by this setting.

Open SQL files with unknown encoding as

You can select the encoding with which to open an SQL file with no encoding specification or where the encoding cannot be detected.

Note: SQL files which have no encoding specification are saved with a UTF-8 encoding.

7.2.8.6.2 SQL Editor General Settings

You can change the general settings applicable to the SQL Editor as follows:

1. Click the DB Query tab.
2. At the top of the pane, click Options and then click SQL Editor.
Data Sources and Targets Databases and MapForce

**General**

*Syntax coloring* emphasizes different elements of SQL syntax using different colors.

Activate the **Connect datasource on execute** check box to connect to the corresponding data source automatically whenever a SQL statement is executed and its data source is not connected.

**Retrieval**

Specify the maximum amount of time permissible for SQL execution (**Execution timeout**) in seconds.

Activating the **Show timeout dialog** check box allows you to change the time-out settings when the permissible execution period is exceeded.

**Entry Helper Buffer**

To enable auto-completion suggestions as you start typing SQL statements, select the **Automatically open** check box (see also [Auto-Completion](#)).

The entry helper buffer for auto-completion can be filled either when you connect to a data source or when it is used for the first time. Note that filling the buffer may take some time. Use the **Clear Buffer** button to reset the buffer.

**Text View Settings**

Allows you to define the specific **Text view settings**: Margins, Tabs, Visual aids, as well as
showing you the Text view navigation hotkeys.

7.2.8.6.3 **SQL Statement Generation Settings**

You can specify the SQL statement generation syntax for various database kinds as follows:

1. Click the **DB Query** tab.
2. At the top of the pane, click **Options** and then click **SQL Editor**.

![SQL Editor: Generation](image)

To define the syntax preferences for a specific database, select it from the list, and then enable or disable the three check boxes to the right.

To define a unique syntax for all databases, select **Apply to all databases**, and then enable or disable the three check boxes to the right. Note that using common settings for all databases may cause inability to edit data in Oracle and IBM DB2 and iSeries databases via a JDBC connection.

7.2.8.6.4 **Query Result View Settings**

You can configure the appearance of the **Results** tab of the **DB Query** pane as follows:

1. Click the **DB Query** tab.
2. At the top of the pane, click **Options** and then click **SQL Editor | Result View**.
Select the **Show grid with alternating colors** check box to display rows in Result tabs as simple grid or with alternating white and colored rows. The alternating color is configurable.

The **Display Options** group lets you define how horizontal and vertical grid lines, as well as line numbers and the **Result** toolbar, are displayed. You can switch any of these options off by deactivating the respective check box.

The **Data Editing** group lets you define the transaction settings, if the cells are to be filled with default values and if a hint is to be displayed when data editing is limited.

### 7.2.8.6.5 SQL Editor Font Settings

You can configure color and font settings of SQL statements that appear in SQL Editor as follows:

1. Click the **DB Query** tab.
2. At the top of the pane, click **Options** and then click **SQL Editor | Fonts**.
The font settings listed in the Font Settings list box are elements of SQL statements. You can choose the common font face, style, and size of all text that appears in SQL Editor. Note that the same font and size is used for all text types.

Only the style can be changed for individual text types. This enables the syntax coloring feature. Click the **Reset to default** button to restore the original settings.

### 7.2.9 Stored Procedures

Stored procedures are programs that are hosted and run on a database server. Stored procedures can be called by client applications and they are often written in some extended dialect of SQL. Some databases support also implementations in Java, .NET CLR, or other programming languages.

Typical uses of stored procedures include querying a database and returning data to the calling client, or performing modifications to the database after additional validation of input parameters. Stored procedures can also perform other actions outside the database, such as sending e-mails.

A stored procedure may have zero or more input and output parameters, and may optionally return zero or more recordsets, in addition to the default return value. Consequently, in MapForce, you can call a stored procedure in various ways:

- Call a stored procedure in order to retrieve data, as if it were a source component on the mapping. This is applicable for procedures that do not take input parameters. When the mapping runs, the procedure is called, and it returns some recordset or output parameters. You can map the recordset, or the output parameters, or both, to any other data type supported by MapForce. For an example, see **Stored Procedures as Data Source**.
- Call a stored procedure as a function-like call, with parameters. In this case, you supply all required input parameters from the mapping, and you can also map the returned
Call a stored procedure as if it were a target component on the mapping. The typical use case is calling a stored procedure with parameters in order to modify the database (for example, insert a record). This approach is suitable if you do not need any output from the stored procedure. Also, in this approach you can execute the stored procedure within a database transaction that can be rolled back in case of an error. For an example, see Stored Procedures in Target Components.

There are also cases where you may need to call stored procedures or perform actions on database tables in a specific order (first insert, then update, and so on). For example, you may need to pass the output parameter of a stored procedure to another stored procedure. Or you may need to combine data returned by a stored procedure with data from a table. Such actions are possible with the help of local relations defined in MapForce, even when the underlying database does not enforce primary/foreign key relationships between tables. For more information, see Stored Procedures and Local Relations.

**Note:** To illustrate how MapForce implements stored procedures, this chapter uses Microsoft SQL Server 2016 and the "AdventureWorks 2016" database. The latter can be downloaded from https://github.com/Microsoft/sql-server-samples/releases/tag/adventureworks.

**Support notes**

- Stored procedures can be used only in the BUILT-IN execution engine. Code generation in C++, C#, or Java is not supported.
- User-defined types, cursor types, variant types and many "exotic" database-specific data types (such as arrays, geometry, CLR types) are generally not supported as input or output parameter types.
- Procedure and function overloading (multiple definitions of routines with the same name and different parameters) is not supported.
- Some databases support default values on input parameters, this is currently not supported. You cannot omit input parameters in the mapping to use the default value.
- Stored procedures returning multiple recordsets are supported depending on the combination of driver and database API (ODBC/ADO/ADO.NET/JDBC). Only procedures that return the same number of recordsets with a fixed column structure are supported.
- Whenever possible, use the latest version of the database native driver maintained by the database vendor. Avoid using bridge drivers, such as ODBC to ADO Bridge, or ODBC to JDBC Bridge.
- You can optionally enable database transactions for stored procedures that are called as data target, see Stored Procedures in Target Components. Transactions are not supported for stored procedures that are called as a data source (without input parameters), or those that are called like a function (with both input and output).

The following table lists the database-specific support notes.

<table>
<thead>
<tr>
<th>Database</th>
<th>Support notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Stored procedures in Microsoft Access databases have very limited functionality and are not supported in MapForce.</td>
</tr>
<tr>
<td>DB2</td>
<td>Supported in MapForce: stored procedures, scalar functions, table-valued functions.</td>
</tr>
<tr>
<td>Database</td>
<td>Support notes</td>
</tr>
<tr>
<td>----------</td>
<td>---------------</td>
</tr>
<tr>
<td><strong>Database</strong></td>
<td><strong>Support notes</strong></td>
</tr>
</tbody>
</table>
| | • Return values from DB2 stored procedures are not supported because they cannot be read via the database APIs used in MapForce.  
| | • Row-valued functions (RETURNS ROW) are not supported.  
| | • It is recommended to install at minimum "IBM_DB2 9.7 Fix Pack 3a" to avoid a confirmed JDBC driver issue when reading errors/warnings after execution. This also fixes an issue with the ADO provider that causes one missing result set row. |
| Firebird | • Supported in MapForce: stored procedures, table-valued functions |
| Informix | • Supported in MapForce: stored procedures, table-valued functions. |
| MariaDB | • Supported in MapForce: stored procedures, scalar functions |
| MySQL | • Supported in MapForce: stored procedures, scalar functions  
| | • MySQL includes complete support for stored procedures and functions starting with version 5.5. If you are using an earlier version, functionality in MapForce is limited. |
| Oracle | • Supported in MapForce: stored procedures, scalar functions, table-valued functions.  
| | • It is recommended to use a native Oracle driver instead of the **Microsoft OLE DB Provider for Oracle**.  
| | • Oracle has a special way to return result sets to the client by using output parameters of type REF CURSOR. This is supported by MapForce for stored procedures, but not for functions. The names and number of recordsets is therefore always fixed for Oracle stored procedures. |
| PostgreSQL | • Supported in MapForce: scalar functions, row-valued functions, table-valued functions.  
| | • In PostgreSQL, any output parameters defined in a function describe the columns of the result set. This information is automatically used by MapForce - no detection by execution or manual input of recordsets is needed. Parameters of type `refcursor` are not supported. |
| Progress OpenEdge | • Supported in MapForce: stored procedures. |
| SQL Server | • Supported in MapForce: stored procedures, scalar functions, table-valued functions.  
| | • It is recommended to use the latest **SQL Server Native Client** driver instead of the **Microsoft OLE DB Provider for SQL Server**.  
| | • The ADO API has limited support for some data types introduced with SQL Server 2008 (datetime2, datetimeoffset). If you encounter data truncation issues with these temporal types when using ADO with the SQL Server Native Client, you can set the connection string argument `DataTypeCompatibility=80` or use ODBC.  
| | • SQL Server Procedures have an implicit return parameter of type... |
### Adding Stored Procedures to the Mapping

On the mapping area, stored procedures are shown as part of the database component where they belong. In order for stored procedures to be visible on the database component, you must explicitly select them when adding the database component to the mapping, as shown below. In this example, we connect to the "AdventureWorks" database running on SQL Server. Instructions are similar for other database types.

**To add stored procedures to the mapping:**

1. Do one of the following:
   - On the **Insert** menu, click **Database**.
   - Click the **Insert Database** ( 
   ![Insert Database](image)

2. Follow the database wizard steps until you get to the "Insert Database Objects" dialog box. For detailed instructions applicable to each database type, see **Database Connection Examples**.

3. Select the check boxes next to the database objects that you need to be displayed on the mapping, and click **OK**. In this example, we have selected all the tables, views, and stored procedures available in the "HumanResources" schema.

### Database Support Notes

<table>
<thead>
<tr>
<th>Database</th>
<th>Support notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLite</td>
<td>• SQLite does not use stored procedures.</td>
</tr>
</tbody>
</table>
| Teradata | • Supported in MapForce: stored procedures, macros.  
• Scalar functions, aggregate functions and table functions are not supported  
• Known issue: The Teradata ODBC driver refuses to populate output parameter values after a procedure call. |

*int null*, which is available for mapping. If the procedure omits a RETURN statement, the resulting value is 0.
The database component is now added to the mapping. Notice that stored procedures are identified by the icon. In addition, tables, views and procedures are sorted alphabetically in the database component.

The **Show Context Menu** button next to each stored procedure lets you configure how the stored procedure is to be called, and other procedure-related settings, as follows:

<table>
<thead>
<tr>
<th>Option</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Show Nodes as Source</strong></td>
<td>Select this option if you want to call a stored procedure <em>without parameters</em> in order to retrieve data from a database and map it to another component supported by MapForce (XML, text, EDI, and so on). For an example, see Stored Procedures as Data Source.</td>
</tr>
<tr>
<td><strong>Show Nodes as Target</strong></td>
<td>Select this option if you want to call a stored procedure in order to modify the database or perform another specific action where you don't need the output of the stored procedure. For an example, see Stored Procedures in Target Components.</td>
</tr>
<tr>
<td><strong>Insert Call with Parameters</strong></td>
<td>Select this option if you want to call a stored procedure <em>with parameters</em> and want to map the returned data to another component supported by MapForce. For an example, see</td>
</tr>
<tr>
<td>Stored Procedures with Input and Output Parameters.</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Edit Recordset Structures</strong></td>
<td></td>
</tr>
<tr>
<td>Applicable for stored procedures that return recordsets. Select this option to execute the stored procedure once, so that MapForce can determine the structure of the returned recordset and display it on the mapping. Alternatively, if you don't want to execute the stored procedure at design time, you can define the recordset structure manually.</td>
<td></td>
</tr>
<tr>
<td><strong>Procedure Settings</strong></td>
<td></td>
</tr>
<tr>
<td>Applicable only for stored procedures that were configured as &quot;target&quot; (that is, those that update the database). Select this option to configure additional procedure-related settings, such as running a custom SQL query before calling the procedure, or enabling database transactions.</td>
<td></td>
</tr>
</tbody>
</table>

### 7.2.9.2 Stored Procedures as Data Source

This example shows you how to call a procedure that takes no input parameters and just retrieves some data from the database. In this scenario, the stored procedure acts as a source component to the mapping, and you can map data retrieved by it to any other target component supported by MapForce. If you need to call a stored procedure with input parameters, see Stored Procedures with Input and Output.

Let us first create the demo stored procedure in the "AdventureWorks" database. To do this, run the script below against the database. You can do this from a query window of Microsoft SQL Server Management Studio, or directly from the DB Query tab of MapForce, see Browsing and Querying Databases. In either case, make sure that your database user account has permission to create stored procedures.

```sql
CREATE PROCEDURE HumanResources.uspGetAllEmployees
AS
    SELECT LastName, FirstName, JobTitle, Department
    FROM HumanResources.vEmployeeDepartment
```

The stored procedure above returns employee information from the vEmployeeDepartment view. The following steps show you how to create a mapping that consumes data returned by this procedure.

1. Connect to the "AdventureWorks" database from MapForce and add the stored procedure to the mapping, as described in Adding Stored Procedures to the Mapping. Make sure that your database user account has permission to view and execute stored procedures.
2. Click the Show Context Menu button next to the stored procedure and select Show Nodes as Source.
3. Click the **Show Context Menu** button again and select **Edit Recordset Structures**. The “Recordset Structures” dialog box appears.

A stored procedure may - in addition to output parameters - return one or more recordsets, which can be used as data sources in MapForce. Since the structure of these recordsets depends on the procedure’s implementation and cannot be read from the database catalog, you have the following options:

- Retrieve recordsets automatically by calling the procedure with user-defined parameters.
  
  ![Check box and button for defining input parameters and calling procedure]

- Define recordset structures manually by typing in column names and data types.

**Recordsets:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Nullable</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Specify new recordset name here.]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notice, stored procedures that do not return the same recordset structures with each call are not supported.

Calling a stored procedure at design time may have side effects (depending on the procedure implementation). If you do not want to execute the stored procedure at
4. Click **Define input parameters and call procedure**, and then click **OK**. The “Evaluate Stored Procedure” dialog box appears.

5. Click **Execute**, and then click **OK**. The recordset structure (“RS1”) is now visible both on the “Recordset Structures” dialog box and on the mapping.

6. At this stage, you can add a target component where the retrieved data will be written. In this example, data will be written to a CSV file. On the **Insert** menu, click **Text File**, and add a CSV component to the mapping. For more information, see [CSV and Text Files](Altova%20MapForce%202019%20Enterprise%20Edition).
You can now preview the mapping. Click the Output button and observe the mapping result in the Output pane, for example:

![Diagram of the mapping process](image)

### 7.2.9.3 Stored Procedures with Input and Output

This example shows you how to call a procedure that takes input parameters and also retrieves some output from the database. In this scenario, the stored procedure is called similar to a Web service, or a function, and you can map data retrieved by it to any other target component supported by MapForce.

Let us first create the demo stored procedure in the “AdventureWorks” database. To do this, run the script below against the database. You can do this from a query window of Microsoft SQL Server Management Studio, or directly from the DB Query tab of MapForce, see Browsing and Querying Databases. In either case, make sure that your database user account has permission to create stored procedures.

```sql
CREATE PROCEDURE Production.uspSearchProducts
    @SearchString nvarchar(50),
    @MaxPrice money,
    @ComparePrice money OUTPUT
AS
BEGIN
    SET NOCOUNT ON
    SELECT pr.[Name], pr.ListPrice FROM [Production].[Product] pr
    WHERE pr.[Name] like @SearchString AND pr.ListPrice < @MaxPrice

    SET @ComparePrice = @MaxPrice
RETURN @ComparePrice
```

The stored procedure above retrieves a recordset containing product information. It takes as input two parameters: a string with the product name (@SearchString) and the maximum product price (@MaxPrice). In addition to the recordset and the default return parameter, it also retrieves an output parameter (@ComparePrice).

The following steps show you how to create a mapping that consumes data returned by this procedure.

1. Connect to the "AdventureWorks" database from MapForce and add the stored procedure to the mapping, as described in Adding Stored Procedures to the Mapping. Make sure that your database user account has permission to view and execute stored procedures.

2. Click the Show Context Menu button next to the stored procedure and select Insert Call with Parameters. The stored procedure now appears in a separate component on the mapping, where the left side lists the input parameters, and the right side contains the return and the output parameters.

3. Click the Show Context Menu button again, and select Edit Recordset Structures. This is necessary so as to provide to MapForce information about the structure of the recordset returned by the procedure. The "Recordset Structures" dialog box appears.
Calling a stored procedure at design time may have side effects (depending on the procedure implementation). If you do not want to execute the stored procedure at design time, do not click **Execute**, as further described in subsequent steps. Instead, define the expected recordset in the “Recordset Structures” dialog box, by adding recordsets and their associated columns manually. Use the **Add recordset** or **Add column** buttons in the "Recordset Structures" dialog box.

4. Click **Define input parameters and call procedure**. The “Evaluate Stored Procedure” dialog box appears.
5. Fill in the parameter values as shown above, and click **Execute**.

6. Click **OK**. The recordset structure ("RS1") is now visible both on the "Recordset Structures" dialog box and on the mapping.

7. At this stage, you can add a target component where the retrieved data will be written. In this example, data will be written to a CSV file. On the **Insert** menu, click **Text File**, and add a CSV component to the mapping, see also **CSV and Text Files**. Next, draw the mapping connections as illustrated below. Notice that the procedure's input parameters are supplied by means of constants. For more information about constants, see **Add a Constant to the Mapping**.

You can now preview the mapping. Click the **Output** button and observe the mapping result in the
This example shows you how to call a procedure that takes input parameters and updates a database. Calling a procedure this way makes it possible to enable transactions and roll the action back in case of an error, or add a custom SQL statement to be executed before the procedure is called. This scenario implies that the stored procedure acts like a target component in MapForce and you are not interested in the output returned by it. For an example that illustrates how to pass parameters and also map data returned by a stored procedure, see Stored Procedures with Input and Output.

Let us first create the demo stored procedure in the "AdventureWorks" database. To do this, run the script below against the database. You can do this from a query window of Microsoft SQL Server Management Studio, or directly from the DB Query tab of MapForce, see Browsing and Querying Databases. In either case, make sure that your database user account has permission to create stored procedures.

```sql
CREATE PROCEDURE Production.uspAddProductModel
    @ModelName nvarchar(50)
    ,@Inst xml
AS
BEGIN
    INSERT INTO [Production].[ProductModel]
    ([Name]
    ,[Instructions]
    ,[rowguid]
    ,[ModifiedDate])
    VALUES
    (@ModelName
    ,@Inst
    ,NEWID()
    ,GETDATE())
END
```

The stored procedure above takes two parameters (@ModelName, @Inst) as input and inserts the corresponding values into the ProductModel table of the AdventureWorks database, along with some database-generated data.

The following steps show you how to create a mapping that consumes data returned by this
procedure.

1. Connect to the "AdventureWorks" database from MapForce and add the stored procedure to the mapping, as described in Adding Stored Procedures to the Mapping. Make sure that your database user account has permission to view and execute stored procedures.

2. Click the Show Context Menu button next to the stored procedure and select Show Nodes As Target. The stored procedure now appears as target component on the mapping, where the left side lists the input parameters.

3. Click the Show Context Menu button again, and select Procedure Settings. This optional step enables you to execute the stored procedure inside a transaction that can be rolled back. You can also add a custom SQL statement to be executed before the procedure is called.

4. Select the Use Transactions check box.

5. Add the source component that provides data to be inserted into the database. In this example, the source data is supplied by constants; however, any other source component supported by MapForce could act as input. For more information about
constants, see Add a Constant to the Mapping.

Since this mapping updates a database, you do not preview its output directly like with other mappings. Instead, click the Output button to display the pseudo-SQL containing hints about how the database will be modified. If you enabled transactions, these will occur as indicated by the comments.

The pseudo-SQL displayed in the Output pane does not show the actual transaction commands, only hints (as comments). The actual SQL commands are sent to the underlying database API, however.

To run the mapping against the database, do one of the following:

- On the Output menu, click Run SQL-Script.
- Click the Run SQL-Script toolbar button.

Stored procedures and duplicate inputs

If you need to map data from multiple sources on the mapping to the same stored procedure, you can duplicate the stored procedure so that it accepts multiple inputs. To do this, right-click the stored procedure item on the component and select Add duplicate input from the context menu, see also Duplicating Input. When the mapping runs, such duplicate stored procedures will be called once for each duplicated input.

Note that the Add duplicate input command is disabled for the stored procedure parameters, because each parameter is an atomic value (and could also be "nullable").
7.2.9.5  Stored Procedures and Local Relations

Local relations are logical relationships between database fields that you can create in MapForce, saving you the need to change the underlying database, see also Defining Local Relationships. You can define local relations not only for database fields, but also for stored procedures as well, both in source and target components.

In source components, local relations make it easy to read data from related objects, for example, read IDs from a database table and call a stored procedure with each of these IDs to retrieve related information. It is also possible to call a stored procedure with data retrieved from another procedure.

In target components, local relations enable you to define a hierarchical order in which multiple related procedures are to be called. For example, you can first call a stored procedure that creates an ID value, and another one that inserts related information into a table. It is also possible to mix stored procedures and tables in local relations. For example, you can perform the insert directly on a related table instead of calling another procedure, see Using Stored Procedures to Generate Keys.

To create a local relation:

1. Right-click the title bar of a database component and select Add/Remove/Edit Database Objects from the context menu. The “Add/Remove/Edit Database Objects” dialog box opens.
2. Click Add/Edit Relations.
3. Click Add Relation and select the objects between which you want to create the relationship.

In this dialog you can define relations in addition to those defined in the database. Please select for each relation a primary key object and a foreign key object and at least one column pair with matching datatypes.
As illustrated above, a local relation consists of a **primary/unique key** object and a **foreign key** object. Think of it as a parent-child relationship. On the mapping component, the object (table, view, procedure, and so on) where the primary/unique key is will appear as a parent while the object where the foreign key is will appear nested under it. For example, in the database component illustrated below, a local relation was defined between a recordset column (RS1.Department) and a table column (Department.Name). Consequently, the Department table appears as a child of the stored procedure on the mapping. This example is discussed in more detail in Local Relations in Source Components.

![Database Component Illustration]

The following table lists all the possible fields between which you can define local relations. Mixed relationships are possible (for example, mapping the output of a stored procedure to a database column). The fields taking part in the relationship must have the same, or a compatible, data type.

<table>
<thead>
<tr>
<th>Primary/unique key</th>
<th>Foreign key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column of a database table or view</td>
<td>Column of a database table or view</td>
</tr>
<tr>
<td>Output parameter or return value of a stored procedure, see also Stored Procedures</td>
<td>Input parameter of a stored procedure</td>
</tr>
<tr>
<td>Column of a recordset returned by a stored procedure*</td>
<td>Input parameter of a user-defined SELECT statement</td>
</tr>
<tr>
<td>Column of a user-defined SELECT statement, see also SQL SELECT Statements as Virtual Tables</td>
<td></td>
</tr>
</tbody>
</table>

* Applicable if the stored procedure is called either as data source (without parameters) or as a function (with input and output parameters). In order for the recordset to become available for selection, you must execute the stored procedure once, to retrieve the recordset.
7.2.9.6  Local Relations in Source Components

This example shows you how to combine data returned by a stored procedure with data from a table in the same database, with the help of local relations.

If you haven't done so already in a previous example, run the following script to create the demo stored procedure in the "AdventureWorks" database. You can do this from a query window of Microsoft SQL Server Management Studio, or directly from the DB Query pane of MapForce, see Browsing and Querying Databases. In either case, make sure that your database user account has permission to create stored procedures.

```sql
CREATE PROCEDURE HumanResources.uspGetAllEmployees
AS
    SELECT LastName, FirstName, JobTitle, Department
    FROM HumanResources.vEmployeeDepartment
```

The stored procedure above returns employee information from the vEmployeeDepartment view. The following steps show you how to create a mapping that consumes data returned by this procedure.

1. Connect to the "AdventureWorks" database from MapForce, as described in Adding Stored Procedures to the Mapping. Make sure that your database user account has permission to view and execute stored procedures.
2. When prompted to choose database objects, select the Department table and the uspGetAllEmployees stored procedure.

3. Click the Show Context Menu button next to the stored procedure and select Show Nodes as Source.
4. Click the Show Context Menu button again and select Edit Recordset Structures. The "Recordset Structures" dialog box appears.
5. Click Define input parameters and call procedure, and then click OK. The "Evaluate Stored Procedure" dialog box appears.
6. Click **Execute**, and then click **OK**. The recordset structure (“RS1”) is now visible both on the “Recordset Structures” dialog box and on the mapping.

Define the local relationships

Let’s now define a local relationship between the **Department** column of the returned recordset and the **Name** column of the **Department** table.

1. Right-click the title bar of the database component and select **Add/Remove/Edit Database Objects** from the context menu.
2. Click **Add/Edit Relations**, and then click **Add Relation**. Define the relationships as shown below.
3. Click **OK** to close the dialog box. Notice that the **Department** table has now become a child of the **RS1** recordset.

---

**Completing the mapping**

Thanks to the relationship that was just created, it is now possible to map data from the recordset combined with data from the table. For the scope of this example, let's write data to a target XML file, as follows:

1. On the **Insert** menu, click **XML Schema/File** and select the following file:
   `<Documents>\Altova\MapForce2019\MapForceExamples\EmployeesWithDetails.xsd`

2. When prompted to provide a sample XML instance file, click **Skip**.
3. Draw the mapping connections as shown below.

The mapping illustrated above writes data from the database to a target XML file. The source data is a combination of data extracted by the stored procedure with data extracted directly from a table. The mapping uses the `concat` function to produce a string that includes the department name, followed by a dash character, followed by the group name.

To preview the mapping, click the Output button and observe the mapping result in the Output pane, for example:

```
<Employees>
  <Employee Department="Executive-General and Administration">
    <FirstName>Ken</FirstName>
    <LastName>Sánchez</LastName>
    <Title>Chief Executive Officer</Title>
  </Employee>
  <Employee Department="Engineering-Research and Development">
    <FirstName>Terri</FirstName>
    <LastName>Duffy</LastName>
    <Title>Vice President of Engineering</Title>
  </Employee>
</Employees>
```

7.2.9.7 Using Stored Procedures to Generate Keys

This example shows you how to insert some key (ID) generated by a stored procedure into another table, with the help of local relations.

Let us first create the demo stored procedure in the "AdventureWorks" database. To do this, run the script below against the database. You can do this from a query window of Microsoft SQL Server Management Studio, or directly from the DB Query tab of MapForce, see Browsing and Querying Databases. In either case, make sure that your database user account has permission to create stored procedures.
CREATE PROCEDURE Production.uspAddProductModelEx
    @ModelName nvarchar(50)
   ,@Inst xml
   ,@ProductModelID int OUTPUT
AS
BEGIN
    INSERT INTO [Production].[ProductModel]
    ([Name], [Instructions], [rowguid], [ModifiedDate])
    VALUES
    (@ModelName, @Inst, NEWID(), GETDATE())

    SELECT @ProductModelID = SCOPE_IDENTITY()
END

The stored procedure above takes two parameters (@ModelName, @Inst) as input and performs an INSERT operation into the ProductModel table. It then returns the generated @ProductModelID as output parameter. The requirement is to insert the @ProductModelID returned by the stored procedure into the ProductModelIllustration table.

The following steps show you how to create a mapping that satisfies the requirement above.

1. Connect to the "AdventureWorks" database from MapForce, as described in Adding Stored Procedures to the Mapping. Make sure that your database user account has permission to view and execute stored procedures.
2. When prompted to choose database objects, select the ProductModelIllustration table and the uspAddProductModelEx stored procedure.
3. Click the Show Context Menu button next to the stored procedure and select Show Nodes As Target. The stored procedure now appears as target component on the mapping, where the left side lists the input parameters.
4. Optionally, if you want to execute the stored procedure inside a transaction, click the Show Context Menu button again, select Procedure Settings, and then select the Use Transactions check box. Defining the transaction for the stored procedure ensures
that retrieving the key and inserting the record occur during the same transaction.

5. Right-click the title bar of the database component and select **Add/Remove/Edit Database Objects** from the context menu.

6. Click **Add/Edit Relations**, and then click **Add Relation**. Define the relationships as shown below.

![Add/Edit Relations dialog box](image)

7. Click **OK** to close the dialog box. Notice that the `ProductModelIllustration` table now appears as a child of the stored procedure. The stored procedure output parameter (@ProductModelID) is displayed as an indicator that it will be used in the local relation, but it does not have any input or output connectors.

![Database component with relationships](image)

8. In this example, the @Inst parameter is of XML type. Right-click the @Inst parameter on the component and select **Assign XML Schema to Field** from the context menu. Next, select the `Production.ManuNstructionsSchemaCollection` schema from the database. When prompted to select a root element, leave the default value as is, and click **OK**. For more information about mapping data to database XML fields, see **Mapping XML Data to/from Database Fields**.
9. Add the source components that provides data to be inserted into the database. In this example, the source data is supplied by constants; however, any other source component supported by MapForce could act as input. For more information about constants, see Add a Constant to the Mapping.

Since this mapping updates a database, you do not preview its output directly like with other mappings. Instead, click the Output button to display the pseudo-SQL containing hints about how the database will be modified. If you enabled transactions, these will occur as indicated by the comments.
The pseudo-SQL displayed in the Output pane does not show the actual transaction commands, only hints (as comments). The actual SQL commands are sent to the underlying database API, however.

To run the mapping against the database, do one of the following:

- On the Output menu, click Run SQL-Script.
- Click the Run SQL-Script toolbar button.

### 7.2.10 Handling Database Exceptions

When running mappings that modify a database, MapForce (or MapForce Server, or a MapForce-generated program) may potentially encounter various database-related errors. For example, the database account may not have enough privileges to perform a specific database action, or some database constraints might be in place that prevent the mapping from inserting or updating invalid data. The latter may happen, for example, when a table column is mandatory but the mapping does not supply a value for it.

To make it possible to roll back data if database exceptions are encountered, you can configure the mapping to use database transactions. As further described below, you can enable transactions at database component level and at table action level.

The transaction rollback settings you configure in MapForce are preserved when you run the mapping either with MapForce Server or with a MapForce-generated program.

#### Transactions at database component level

To enable transactions at database component level, do the following:

1. Right-click the database component, and select Properties from the context menu (Alternatively, double-click the title bar of the database component). This opens the Database Component Settings dialog box.
2. Select the **Use transactions** check box.

Doing this encloses all changes done by the database component inside a transaction that will be rolled back in case of error. When an error occurs during mapping execution, the outcome depends on the option you select from the drop-down list:

- **Rollback top transaction and stop** — The transaction which encloses all the changes done by the database component is rolled back and execution of the mapping stops.
- **Rollback top transaction and continue** — Same as above, but the mapping continues to run after rollback (for example, in order to process a second target component, if one exists).

Some mappings may contain multiple database components. These, in their turn, may all use the same database connection, or connections to completely different databases. The outcome of such mappings in case of database error depends on the execution engine:

- If the mapping is run with MapForce, only one target component can be executed when the mapping runs. This is the component where the **Preview** button is enabled. If a database error occurs in that component, and if the **Use transactions** check box is enabled, all the changes done by the component will be rolled back.
- If the mapping is run with MapForce Server or a MapForce-generated program, all the target components are executed, sequentially. In this case, when a database error occurs, the rollback will take place for the database component where the error occurred. The mapping will stop or continue executing the next target component depending on the value you selected from the **When an error occurs** drop-down list, as mentioned above.

For reference to all the settings available on the Database Component Settings dialog box, see **Database Component Settings**.

**Transactions at table action level**

Transactions at table action level enclose each individual table action (insert, update, and so on) inside a transaction. With this setup, if some insert or update action fails on a record, it is possible to roll back only the current failed action and continue processing other records. To enable transactions at table action level, select the **Use transactions** check box from the Database Table Actions dialog box (see also **Database Table Actions Settings**).
The available options are as follows:

- **Rollback top transaction and stop** — If the mapping is going to update multiple tables having a parent-child relationship, it is possible to enable transactions not only for the parent table, but also for the child table. Since transactions can be enabled at multiple levels, “top” refers to the transaction at the highest level in the hierarchy. This is the transaction from the database component level, if you enabled transactions at that level. Otherwise, it is the transaction defined in the topmost table action, assuming that the component contains multiple nested table actions. To understand this option, recall that it is possible for a component to update multiple tables hierarchically, as described in Inserting Data into Multiple Linked Tables. For example, a component may have a parent “Insert” action that inserts person records and a nested “Insert” action that inserts one or more addresses for each person record into a separate table. Since you can enable transactions for both insert actions, “top” refers to the highest level in the hierarchy (in this example, the people insert action). Therefore, when an exception is encountered while inserting an address, this rolls back the top transaction (which is at person table
level, in this example) and execution stops. In other words, no person record will be inserted if it wasn't possible to insert the person's address in first place.

- **Rollback top transaction and continue** — Same as above, but the mapping continues to run after rollback (for example, to process another target component, if one exists).
- **Rollback current transaction and stop** — When an exception is encountered, this rolls back only the changes enclosed in the current transaction; changes done previously outside of the current transaction will be committed.
- **Rollback current transaction and continue** — Same as above, but the mapping continues to run after rollback.

### Stored procedure-level transactions

You can also enable database transactions for stored procedures that are called as data target. For an example, see Stored Procedures in Target Components. Transactions are not supported for stored procedures that are called as a data source (without input parameters), or those that are called like a function (with both input and output).

### Rollback

To see if transactions are going to occur at mapping runtime, click the Output tab and observe the pseudo-SQL code generated by MapForce. The image below illustrates the preview of a mapping where transactions are enabled at component level and table action level. As shown below, a main transaction encloses the entire operation (identified by the comments --begin transaction and --commit transaction). Also, multiple smaller transactions occur for each insert operation (these are identified by the SAVEPOINT command).

```
-- begin transaction
PRAGMA foreign_keys = ON;

SAVEPOINT PRODUCTS
INSERT INTO "products" ("name") VALUES ('Product A')
SAVEPOINT PRODUCTS
INSERT INTO "products" ("name") VALUES ('Product B')
SAVEPOINT PRODUCTS
INSERT INTO "products" ("name") VALUES ('Product C')
-- commit transaction
```

When the mapping runs in MapForce Server, or from a MapForce-generated program, the rollback will take place based on the options you have configured while designing the mapping, as described above. For example, if you selected the option **Rollback top transaction and stop**, a failed insert operation will cause all the changes to be rolled back, even if some other inserts may have been successful. However, if you selected **Rollback current transaction and continue**,
only the failed insert will be rolled back to the nearest savepoint, and the mapping will continue to attempt inserting other records.

If you run a transaction-enabled mapping directly in MapForce with the menu command **Output | Run SQL-Script**, a dialog box informs you when the database exception is encountered, for example:

![MapForce: Database Transaction Exception dialog](image)

You can then change the subsequent behavior as follows:

**Rollback this transaction and stop**

- The current transaction is already rolled back at the time the dialog comes up
- An error is shown at the top of the preview window and in the Messages window

```
Transformation-Error in Database SQL-execution:
Error in: INSERT INTO "products" ("name") VALUES (?)
Reason: NOT NULL constraint failed: products.quantity
```

- Any parent transaction is committed
- Mapping execution stops with an error code
Rollback this transaction and continue

- The current transaction is already rolled back at the time the dialog comes up
- The error is treated as a warning, and will be shown in the Messages window when mapping execution finishes.

- Any parent transaction is not affected
- Mapping execution continues with the next record (if this transaction is not on component level) or with the next component (MapForce Server only)

Rollback top and stop

- The current transaction is already rolled back at the time the dialog comes up
- An error is shown at the top of the preview window and in the Messages window
- Any parent transaction is rolled back
- Mapping execution stops with an error code

Note: The Rollback top and stop option is available on the dialog box only when there are nested transactions (savepoints).

Rollback top and continue

- The current transaction is already rolled back at the time the dialog comes up
- The error is treated as a warning, and will be shown in the Messages window when mapping execution finishes.
- Any parent transaction is rolled back
- Mapping execution continues with the next record (if this transaction is not on component level) or with the next component (MapForce Server only)

Note: The Rollback top and continue option is available on the dialog box only when there are nested transactions (savepoints).

7.2.10.1 Example: Transaction Rollback

This example illustrates possible ways to handle database exceptions when running a mapping that modifies a database. The database used in this example stores user records (users table) and addresses (addresses table). Each user can have zero, one, or two addresses (for example, a home address and a work address). Namely, each address in the addresses table contains a user_id field which points to the id field in the users table. The users table is therefore a "parent" table for addresses. The listing below illustrates the SQL creation scripts for both tables (note the syntax is applicable to a SQLite database):

```
CREATE TABLE
```
Source tables

The business requirement is to copy all data from the users and addresses table to some new tables, namely, the new_users and new_addresses tables. These are almost identical to the users and addresses tables. The only difference is that the table new_addresses does not allow null values for the columns is_shipping and is_billing, as highlighted in the code listing below. This is important because it means that exceptions will occur if null values are encountered.

Target tables

The mapping that copies all data from the old to new tables is illustrated below. You can find this mapping at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\DatabaseExceptions.mfd`. 
As illustrated above, this mapping is configured to copy data verbatim from a source database component to a target database component. No other transformations take place, for the sake of simplicity. In this example, both the source and target component belong to the same database, only the source and target tables are different. Double-click the title bar of the target component and notice that all the records are set to be deleted before each insert action. This ensures that the target tables are always empty before each insert.

As stated before, exceptions will occur if the source addresses table contains null values. You can check that the source table contains null values as follows:

1. Click the DB Query tab.
2. Right-click the addresses table, and select Show in SQL Editor | SELECT.
3. Click Execute Query.
As illustrated above, there are various NULL fields in the source addresses table. Considering this, when running this mapping, you have various options to deal with exceptions, for example:

- A) If any exception is encountered, roll back all changes. In other words, do not insert any records if the source data contains invalid records.
- B) If any exception is encountered, skip the records where the exception occurs but keep inserting records that are valid.

**Scenario A: Rollback all changes on exception**

Business requirement: I want to roll back all changes to the database if an exception occurs. To configure the mapping to accomplish this:

1. Right-click the title bar of the target database component and select **Properties** from the context menu. Alternatively, double-click the title bar. This opens the Database Component Settings dialog box.
2. Select the **Use Transactions** check box, and **choose rollback top transaction and stop**.

When the mapping is configured as shown above, any encountered exception will cause the top-level transaction at database component level to be rolled back. To run the mapping with these settings:
1. Click the **Output** tab.
2. On the Output menu, click **Run SQL-Script**. At this stage, the mapping encounters an exception because of a null value in the source addresses table. The Database Transaction Exception dialog box appears.

The result is as follows:

- All the changes are rolled back.
- No records are inserted in the `new_users` table
- No records are inserted in the `new_addresses` table

**Scenario B) Rollback current transaction and continue execution**

Business requirement: I want to skip records which generate an exception but keep inserting valid records. To configure the mapping to accomplish this:

1. **Click the [DEL Alt] button next to the `new_users` table, select the **Use transactions** check box and choose **rollback current transaction and continue**
2. **Click the [Alt] button next to the `new_addresses` table, select the **Use transactions** check box and choose **rollback current transaction and continue**

When you run the mapping with these settings, the Database Transaction Exception dialog box. Select the options as shown below to roll back only the current transaction and keep running the mapping:
The result is as follows:

- All erroneous transactions are rolled back.
- 5 users out of 5 are inserted (because no user record generated any database exception)
- 2 addresses out of 5 are inserted (because 3 addresses contain null values and generated exceptions)

**Scenario C) Rollback top transaction and continue execution**

Business requirement: If an address fails to be inserted, then the parent user record should also not be inserted. To configure the mapping to accomplish this:

1. Double-click the title bar of the database component, and clear the **Use transactions** check box.
2. Click the **DEL** button next to the **new_users** table, select the **Use transactions** check box and choose **rollback current transaction and continue**
3. Click the **Add** button next to the **new_addresses** table, select the **Use transactions** check box and choose **rollback top transaction and continue**

When you run the mapping with these settings, the Database Transaction Exception dialog box. Select the options as shown below, and repeat the choice as many times as prompted:
The result is as follows:

- 2 users are inserted (Sharda Junker, Tobie Hughey)
- 1 address is inserted

Explanation: The user “Sharda Junker” is the only user who does not have an address. No exceptions occurred for this record, so it was inserted. The second user, “Tobie Hughey”, was inserted because it is the only user where no exceptions occurred at address level. All other user records were not inserted because they had at least one address where an exception occurred and the transactions were rolled back.

There are two addresses which don’t have nulls and thus don’t throw exceptions. These are addresses with id=1 and id=4. The insert transaction for the first address, however, was rolled back because the parent user transaction had to be rolled back. Therefore, only the address with id=4 was inserted.

Note that the same result can also be achieved as follows:

1. Click the **DEL Align** button next to the **new_users** table, select the **Use transactions** check box and choose **rollback current transaction and continue**
2. Click the **Align** button next to the **new_addresses** table, and clear the **Use transactions** check box.
Running the mapping with MapForce Server

If you have licensed MapForce Server, you can also run the mapping at the command line, on a Linux, macOS, or Windows machine, as follows:

1. Compile the mapping to a MapForce Server execution file (.mfx) with the menu command File | Compile to MapForce Server Execution File, see also Compiling Mappings to MapForce Server Execution Files.
2. Copy the .mfx file and the ExceptionsDemo.sqlite database to a directory on the server machine. Let's call it the "working directory".
3. Change the command prompt to the working directory and run MapForce Server with the command below:

   mapforceserver run DatabaseExceptions.mfx

Notes:

- **mapforceserver** is the path to the MapForce Server executable as applicable for your operating system. Change the path to the .mfx file as applicable. For example, on a Linux machine, the path is: /opt/Altova/MapForceServer2019/mapforceserver.

In server execution, you can also run mappings as an API call, or as FlowForce Server jobs, either on demand or on recurring basis. For more information, see Automation with MapForce Server.

7.2.11 Database Tracing and Error Logging

When a mapping writes data to a database, you can optionally enable database tracing and error logging. Tracing is useful if you want to track all the changes done by the mapping to the database when the mapping runs. When tracing is on, events such as database insert or update actions or errors are logged in an XML file that you can later analyze or perhaps process further in an automated way. If you prefer the log file to be in a format other than XML, you can optionally map data from it to some other component kind supported by MapForce (for example, a text file, or even another database).

**Note:** In order for a mapping to support tracing and error logging, the mapping language must be set to **Built-in**. Database logging and tracing is not supported in C#, C++, or Java.

Database tracing is configurable, in that you can choose to trace all messages, only errors, or you can disable tracing completely. In addition, you can use tracing at the following levels:

- Database component level. At this level, you effectively switch tracing on or off for that database component. This is useful for mappings that have multiple target database components, and you need tracing only for some of them.
- Table or stored procedure level. On a database component, you can decide whether tracing should take place for each table or stored procedure. When enabled, tracing at this level includes events pertaining to table actions such as "Insert All", "Update If", "Ignore If", "Delete If". In case of stored procedures, events related to the stored procedure call are traced.
- Database field level. By default, all fields are traced, but you can exclude certain fields
from being traced, or you can specify that they should be included in the trace file only on error.

Importantly, the three levels above are hierarchical and tracing settings are propagated from top to bottom. Therefore, if you disable tracing at database component level, it is not possible to enable it at table or stored procedure level. Likewise, if you disable tracing at table level, it is not possible to set it at database field (column) level. The same applies when you narrow down the tracing level. For example, if you limit tracing only to errors at database component level, then it is not possible to use full tracing at table or stored procedure level.

**Tracing at database component level**

To enable tracing at database component level:

1. Right-click the database component, and select **Properties** from the context menu (Alternatively, double-click the title bar of the database component). This opens the Database Component Settings dialog box.

   ![Database Component Settings Dialog](image)

   - **Trace level**: Select the desired trace level (Always, Errors, Disabled). Selecting Errors will create a trace file only if database errors occur when the mapping runs.
   - **Trace file**: Next to Trace file, type the name or path of the XML file where all the tracing details will be written (for example, Log.xml). If you enter a file name instead of a full path, the trace file will be created in the same directory where the mapping runs.

After you enable tracing, the database component changes its structure on the mapping. Namely, it displays, in addition to the database structure, a new node hierarchy of items where the top item in the hierarchy is the trace file. For more information, see **Trace File Structure**.

Enabling tracing at database component level automatically enables it for all tables or stored procedures on that component to which a mapping connection exists. If necessary, you can further tweak tracing at the table or stored procedure level, as described below. Note that tables or stored procedures that are displayed on the component but do not take part in the mapping (that is, no mapping connections exist to them) will not be traced.

**Tracing at table or stored procedure level**

To enable tracing at table or stored procedure level:

1. Make sure that the tracing level at database component level is set to either **Always** or **Error** (see above).

2. Do one of the following:
   - Click the table action button (for example, Add) next to the table where you want to enable tracing, or
   - If this is a stored procedure, click the **Show Context Menu** button, and then select **Procedure Settings** from the context menu.
3. Select the trace level. The **Use component settings** option inherits the same settings that were defined at the component level. The **Limit to errors** option restricts tracing only to error events. **Always disabled** means that no tracing will occur for this table or stored procedure.

**Tracing at database field level**

When you enable tracing at database component level and table or stored procedure level, all the fields (database columns) are included in the tracing report by default. Optionally, you can restrict tracing only to specific database fields or stored procedure parameters. To do this, click the **Fields** button on the Database Table Actions or Database Procedure Settings dialog box, see above.

Selectations made on this dialog box affect the trace file structure. For example, any fields that you opted to hide will no longer appear in the trace file. You might want to do this, for example, in order to reduce the log file size.

To hide or include multiple fields simultaneously, click the respective option in the header.

### 7.2.11.1 Trace File Structure

When tracing is enabled for a database component, a tracing structure becomes available in the lower half of the component, for example:
As illustrated above, the top node in the tracing structure indicates the name of the trace file that will be written when the mapping runs. The rest of the tracing structure is modeled after the structure of the database tables or stored procedures that take part in the mapping. In this example, the database name is "TargetDatabase", so a similar item exists in the trace structure.

The child item of "TargetDatabase" is `users`, which corresponds to the "users" database table. In this example, the database component has only one table, "users". However, in a component with many tables, the tracing structure includes each table present on the component. If you add or remove tables or stored procedures from the database component, then the tracing structure is updated accordingly. For instructions, see Adding, Editing, and Removing Database Objects.

In this example, an `init:users` structure also appears. This happens when you have configured the mapping to perform some initial action before the main insert, update, or some other action. For example, if you opted to delete all existing records from the table before inserting new ones, the `init` structure is present. It collects tracing information about the initial action (such as delete, or a custom SQL query) and any errors that may occur at this stage.

The `trace:values` structure displays all the columns of the database table (in this case, the columns of the "users" table). For stored procedures, this structure displays the parameters of the stored procedure. By default, all columns are set to be traced, but you can configure which...
columns should be traced, as described previously.

The `trace:actions` items indicates which actions are currently set to take place on the database table. In this example, a database update action `trace:up` is set to take place, so the `trace:update` item is available. If you change the component to perform a database insert action `trace:in`, then the trace structure would change to `trace:insert`, and so on for other action kinds. In case of stored procedures, a `trace:execute` action is visible instead.

Each trace action has a `rows-affected` attribute that specifies how many rows were affected by the respective database action.

The `trace:error` item is populated only if a database error occurs during mapping execution. It has two attributes, `code` and `state`. The text of the error and the attribute values are supplied by the database driver and will therefore be different for each database kind.

The `trace:summary` item includes an `errors` attribute which reports the number of encountered errors. The attribute value is 0 if no errors were encountered.

**Previewing trace files**

When tracing is enabled for a target database component, the trace file is also included in the preview when you click the Output tab to preview the mapping result. This is useful if you want to see how the file will approximately look like. Note that the actual generated trace file will look different based on the mapping execution result.

### 7.2.11.2 Example: Logging Database Errors

This example shows you how to configure database tracing and error logging for a mapping which inserts data into a SQLite database. The instructions are similar for all other database types supported by MapForce, as well as for other database action types (such as "update", "ignore", "delete", and so on).

You can find the demo mapping referenced by this example at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\DatabaseExceptions.mfd`. This mapping inserts data from two tables of a database into a new pair of tables of the same database, and is described in more detail in the Example: Transaction Rollback. Note that some of the records in the database are deliberately null so as to generate database exceptions.
In this example, the business requirement is to enable tracing for the target database component in order to determine how many errors were generated, and where exactly did the errors occur. The steps below illustrate how to achieve this goal.

**Step 1: Enable transaction rollback**

This demo mapping is configured to use transaction rollback at database component level. This means that, when the first exception is encountered, the mapping will stop execution and rollback all data to the initial state. Let's first change the transaction rollback options so that the mapping execution continues even if exceptions are encountered for some of the records:

1. Click the **DEL A:in** button next to the **new_users** table, select the **Use transactions** check box and choose **rollback current transaction and continue**.

2. Click the **A:in** button next to the **new_addresses** table, select the **Use transactions** check box and choose **rollback current transaction and continue**.

You could also disable transaction rollback completely; however, doing this would stop mapping execution at the first encountered error and would not rollback changes, so this is not recommended. Tracing and logging in this case would only reflect execution only up to the first encountered exception. For further information about transaction rollback, see [Handling Database Exceptions](#).
Step 2: Configure error logging

To enable tracing, do the following:

1. Right-click the title bar of the database component, and select **Properties** from the context menu (Alternatively, double-click the title bar of the database component). This opens the Database Component Settings dialog box.
2. Select **Errors** as trace level.
3. Next to **Trace file**, type the name of the tracing file (in this example, **Errors.xml**). Do not enter an absolute path if you intend to run the mapping with MapForce Server on other machines or operating systems.

   ![Tracing Settings](image)

   The target database component has now changed its structure and displays an additional compartment with the trace file structure:

   ![Database Component](image)

   So far, you have enabled tracing at the database component level. Consequently, the tracing settings will be inherited by all table actions on this database component. In this example, they are inherited by the "Insert" actions of both **new_users** and **new_addresses** tables. If you click the action button **Align** for each of these tables, the settings illustrate this fact:

   ![Action Settings](image)

   Optionally, click the **Fields** button to configure which fields should be included or hidden in the trace file. In this example, all fields are included.
Step 3: Configure the mapping output

This step shows you how to map the values from the trace file to some other component kind (in this case, a simple output component). Note that this step is optional; an error log file is produced even if you do not map the trace file to some other component. In this example, however, we would like to explicitly configure the mapping to return the total number of encountered errors in the output. To do this, let's take the following steps:

1. Add a simple output component, by selecting the Insert | Insert Output menu command. For more information about simple input components, see Returning String Values from a Mapping.
2. Add a constant, by selecting the Insert | Constant menu command. The constant will provide the static text "Total errors: " to make the mapping output text more clear.
3. Drag the concat function from the Libraries window onto the mapping. For more information about built-in functions, see Add a Built-in Function to the Mapping.

As illustrated below, the concat function concatenates the text of the constant with the errors value supplied by the trace file.

With the configuration above, when the mapping is run with MapForce Server, the output will display the text: "Total errors: {value}", where value represents the number of errors encountered at runtime.

At this stage, running the mapping in MapForce with the Preview button pressed on the "result" component is not possible and will result in an error. A trace file must exist first, because the simple output reads data from the trace file. The next step shows you how to produce the trace file.

Step 4: Running the mapping

To run this mapping with MapForce:

1. Click the Preview button on the database component.
2. Click the Output tab.
3. On the Output menu, click Run SQL-Script. At this stage, the mapping encounters the first exception; this is intentional. The Database Transaction Exception dialog box appears.
4. Select the rollback options as illustrated above, and click OK.

**Note:** If a mapping is executed by MapForce Server, no dialog box appears and the erroneous transactions are rolled back automatically, according to the rollback settings configured in MapForce.

In this example, once the mapping completes execution, all failed transactions are rolled back, and only the successful ones are committed to the database. The encountered errors are written to the `Errors.xml` file, a small fragment of which is illustrated below:
Now that the log file was produced, you can preview the output that returns the total number of errors in MapForce execution, as follows:

1. Back on the mapping, click the **Preview** button on the "result" component.
2. Click the **Output** tab and observe the result.

This part of the mapping reads data from the log file produced previously. If the log file does not exist, an error will be generated.
7.3 CSV and Text Files

MapForce includes support for mapping data to or from text-based file formats such as CSV (comma-separated values) and FLF (Fixed-Length Field) text files. For example, you can create data transformations such as:

- XML schema to/from flat file formats
- Database to/from flat file formats
- UN/EDIFACT and ANSI X12 to/from flat file formats or databases

Note that, in case of CSV, your files can have as delimiter not only commas, but also tabs, semicolons, spaces, or any other custom values.

In addition to CSV and FLF files, mapping to or from text files with more complex or custom structures is possible using MapForce FlexText (this module is available in MapForce Enterprise Edition). FlexText essentially enables you to define the structure of your custom text data (using a so-called "FlexText template"), for the purpose of mapping it to other formats. For more information, see MapForce FlexText.

Mapping data to or from text files is supported in any one of the following languages: Java, C#, C++, or BUILT-IN.

There are two ways that mapped flat file data can be generated:

- By clicking the Output tab which generates a preview using the Built-in execution engine. You can also save the mapping result by selecting the menu option Output | Save output file, or clicking the icon.
- By selecting File | Generate code in | Java, C#, or C++, and then compiling and executing the generated code.

7.3.1 Example: Mapping CSV Files to XML

The goal of this example is to create a mapping which reads data from a simple CSV file and writes it to an XML file. The files used in the example are available in the <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ folder.

1. Select one of the following as transformation language: Java, C#, C++, or BUILT-IN.
2. Add a Text file component to the mapping area (on the Insert menu, click Text File, or click the Insert Text file toolbar button ( ).
3. Select the **Use simple processing** … option, and then click **Continue**.

4. On the Component Settings dialog box, click **Input file** and browse for the *Altova_csv.csv* file. The file contents are now visible in the lower part of the dialog box. Note that only the first 20 rows of the text file are displayed when in preview mode.

5. Click inside the **Field1** header and change the text to *First-name*. Do the same for all the other fields, as follows: Field 2 ➞ *Last-name*, Field 3 ➞ *Tel-extension*, Field 4 ➞ *Email*, Field 5 ➞ *Position*. TIP: Press the **Tab** key to cycle through all the fields: header1, header2 etc.
6. Click OK.
7. When prompted to change the component name, click "Change component name". The CSV component is now visible in the mapping.
9. Click Skip when prompted to supply a sample XML file, and select Company as the root element.
10. Map the corresponding items of both components, making sure to map the Rows item to the Person item in the schema target.

The connector from the Rows item in the CSV component to the Person item in the schema is essential, as it defines which elements will be iterated through. That is, for each row in the CSV file, a new Person element will be created in the XML output file.

11. Click the Output tab to see the result.
The data from the CSV file is now successfully mapped to an XML file.

### 7.3.2 Example: Iterating Through Items

This example illustrates how to create iterations (multiple rows) in a target CSV file. The mapping design file accompanying this example is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\Tut-xml2csv.mfd`.

This mapping has been intentionally created as incomplete. If you attempt to validate the example file using the menu command `File | Validate Mapping`, you will notice that validation warnings occur. Also, if you preview the mapping output, a single row is produced, which may or may not be your intended goal.

Let's assume that your goal is to create multiple rows in the CSV file from a sequence of items in the XML file. You can achieve this by drawing a connection to the `Rows` item of the target CSV file.
For example, to iterate through all offices and have the output appear in the CSV file, it is necessary to connect Office to Rows. By doing this, you are instructing MapForce: for each Office item of the source XML, create a row in the target CSV file.

The Rows item in the CSV component acts as an iterator for the sequence of items connected to it. Therefore, if you connect the Office item, the output creates a row for each office found in the source XML.

```
1  "Microtech, Inc.",Level 1 support,Major Ave 1,Vancouver,558833
2  "Microtech Partners, Inc.",Level 2 support,Perro Bvd 1324,Otto
3
```

In a similar fashion, if you connect Department to the Rows item, a row will be produced for each department found in the source XML.
Finally, mapping Person to the Rows item results in all the Persons being output. In this case, MapForce will iterate through the records as follows: each Person within each Department, within each Office.

7.3.3 Example: Creating Hierarchies from CSV and Fixed-Length Text Files

This example is available at the following path: <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\Tut-headerDetail.mfd. The example uses a CSV file (Orders.csv) which has the following format:

- Field 1: H defines a header record and D a detail record.
- Field 2: A common key for both header and detail records.
- Each Header or Detail record is on a separate line.

The contents of the Orders.csv file are shown below.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;Microtech, Inc.&quot;, Admin, Major Ave 1, Vancouver, 5588339, Clive, Clo</td>
</tr>
<tr>
<td>2</td>
<td>&quot;Microtech, Inc.&quot;, Sales and Marketing, Major Ave 1, Vancouver, 558</td>
</tr>
<tr>
<td>3</td>
<td>&quot;Microtech, Inc.&quot;, Manufacturing, Major Ave 1, Vancouver, 5589099, K</td>
</tr>
<tr>
<td>4</td>
<td>&quot;Microtech, Inc.&quot;, Level 1 support, Major Ave 1, Vancouver, 5588339</td>
</tr>
<tr>
<td>5</td>
<td>&quot;Microtech Partners, Inc.&quot;, Admin, Perro Blvd 1324, Ottawa, 354920,</td>
</tr>
<tr>
<td>6</td>
<td>&quot;Microtech Partners, Inc.&quot;, Sales and Marketing, Perro Blvd 1324, 0</td>
</tr>
<tr>
<td>7</td>
<td>&quot;Microtech Partners, Inc.&quot;, Level 2 support, Perro Blvd 1324, Ottawa</td>
</tr>
</tbody>
</table>
The aim of the mapping is as follows:

- Map the flat file CSV to an hierarchical XML file
- Filter the Header records, designated with an H
- Associate the respective detail records, designated with a D, with each of the header records.

For this to be achieved, the header and detail records must have one common field. In this case the common field, or key, is the second field of the CSV file, i.e. OrderNo. In the CSV file, both the first header record and the following two detail records contain the common value 111.

The Orders.csv file has been inserted twice to make the mapping more intuitive.

The **Tut-headerDetail.xsd** schema file has a hierarchical structure: Order is the root element, with Header as its child element, and Detail being a child element of Header.

The first Orders.csv file supplies the **Header** records (and all mapped fields) to the Header item in
the schema target file. The filter component is used to filter out all records other than those starting with H. The **Rows** item supplies these filtered records to the Header item in the schema file.

The second `Orders.csv` file supplies the **Detail** records (and all mapped fields) by filtering out the Detail records that match the `OrderNo` key of the Header record. This is achieved by:

- Comparing the **OrderNo** field of the Header record with the same field of the Detail records, using the **equal** function (the **priority context** is set on the `a` parameter for enhanced performance).
- Using the **Logical-and** function to only supply those Detail records containing the same `OrderNo` field, as the Header record.

The **Rows** item supplies these filtered records to the Header and Detail items in the schema file, through the on-true parameter of the filter component.

Clicking the Output tab produces the XML file displayed below. Each Header record contains its data, and all associated Detail records that have the same Order No.

```xml
<Header>
  <RecordType>H</RecordType>
  <OrderNo>111</OrderNo>
  <TotalWeight>332.1</TotalWeight>
  <TotalUnitCost>22537.7</TotalUnitCost>
  <Currency/>
  <Shipping-details>Container ship</Shipping-details>
</Header>

<Detail>
  <RecordType>D</RecordType>
  <OrderNo>111</OrderNo>
  <ProductNo>A-1573-227</ProductNo>
  <UnitWeight>10</UnitWeight>
  <UnitNo>3</UnitNo>
  <UnitCost>400</UnitCost>
  <Unit-description>Microwave</Unit-description>
</Detail>

<Detail>
  <RecordType>D</RecordType>
  <OrderNo>111</OrderNo>
  <ProductNo>B-152-427</ProductNo>
  <UnitWeight>7</UnitWeight>
  <UnitNo>6</UnitNo>
  <UnitCost>1200</UnitCost>
  <Unit-description>Miscellaneous</Unit-description>
</Detail>
</Header>
```

Let's now have a look at another example, which uses a slightly different CSV file and is available in the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\` folder as `Head-detail-inline.mfd`. The difference is that:

- No record designator (H, or D) is available
- A common key field, the first field of the CSV file, still exists for both header and detail records (Head-key, Detail-key...). The field is mapped to `OrderNo` in the schema target
Header and all respective Detail fields are all on the same line.

The mapping has been designed as follows:

- The key fields are mapped to the respective OrderNo items in the schema target.
- The Detail item in the schema target file has been duplicated, and is displayed as Detail (2). This allows you to map the second set of detail records to the correct item.
- The result of this mapping is basically the same XML file that was produced in the first example.

### 7.3.4 Setting the CSV Options

After you add a text component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component settings dialog box in one of the following ways:
- Select the component and, on the Component menu, click Properties.
- Double-click the component header.
- Right-click the component header, and then click Properties.

![Component Settings dialog box (in CSV mode)](image)

The available settings are as follows.

<table>
<thead>
<tr>
<th>Component name</th>
<th>The component name is automatically generated when you create the component. However, you can change the name at any time. The component name can contain spaces and full stop characters. It may not contain slashes, backslashes, colons, double quotes, leading or trailing spaces. If you want to change the name of the component, be aware of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- If you intend to deploy the mapping to FlowForce Server, the component name must be unique.</td>
</tr>
</tbody>
</table>
- It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line.

**Input file**
- Specifies the file from which MapForce will read data. This field is meaningful for a source component and is filled when you first create the component and assign to it a text file. The field can remain empty if you are using the text file component as a target for your mapping.

  In a source component, MapForce uses the value of this field to read column names and preview the contents of the instance text file.

  To select a new file, click **Input File**.

**Output file**
- Specifies the file to which MapForce will write data. This field is meaningful for a target component.

  To select a new file, click **Output File**.

**Save all file paths relative to MFD file**
- When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. This setting affects the input and output files used by the text component. See also Using Relative Paths on a Component.

**Input / Output Encoding**
- Allows you specify the following settings of the output instance file:
  - Encoding name
  - Byte order
  - Whether the byte order mark (BOM) character should be included.

  By default, any new components have the encoding defined in the **Default encoding for new components** option. You can access this option from **Tools | Options**, General tab.

**Field delimiter**
- CSV files are comma delimited "," by default. This option enables you to select the **Tab**, **Semicolon**, or **Space** characters as delimiters. You can also enter a custom delimiter in the **Custom** field.

**First row contains field names**
- Select this option to instruct MapForce to treat the values in the first record of the text file as column headers. The column headers then appear as item names on the mapping.

**Treat empty fields as absent**
- When this option is enabled, empty fields in the source file will not produce a corresponding empty item (element or attribute) in the target file.

  For example, the CSV record "General outgassing"
pollutants,,,,” consists of four fields, the last three of which are empty.

Assuming that the output is an XML file, when this option is disabled, the empty fields will be created in the output with an empty value (in this example, the elements Last, Title, and Email):

```
33  <Person>
34    <First>General outgassing pollutants</First>
35    <Last/>
36    <Title/>
37    <Email/>
38  </Person>
```

When this option is enabled, the empty fields will not be created in the output:

```
38  <Person>
39    <First>General outgassing pollutants</First>
40  </Person>
```

**Quote character**

If your input file contains quotes around field values, select the quote character that exists in the source file. The same setting will also be used for output files.

For output files, you can specify additional settings:

- **Add when needed** Adds the selected quote character to fields where the text contains the field delimiter, or line breaks.
- **Add always** Adds the selected quote character to the generated CSV file.

**CSV / Fixed**

Changes the component type to either CSV or FLF (fixed-length field).

**Preview area**

The lower part of the dialog box displays a preview of up to 20 rows of the file selected as input or output.

If necessary, you can create the structure of the file (or change the structure of the existing one), as follows.
Append field  Creates a new field after the last CSV record.
Insert field  Creates a new field immediately before the currently selected CSV record.
Remove field  Deletes the currently selected field.
<<  Moves the currently selected field one position to the left.
>>  Moves the currently selected field one position to the right.

To change the name of a field, click the header (for example, Field1), and type the new value. Note that the field names are not editable when the First row contains field names option is enabled.

<table>
<thead>
<tr>
<th>Field1</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
</tr>
</tbody>
</table>

To change the data type of a field, select the required value from the drop-down list. MapForce checks the data type, so if the input data and the field format to do not agree, then the data is highlighted in red.

<table>
<thead>
<tr>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>decimal</td>
</tr>
<tr>
<td>NaNoroll, Inc.</td>
</tr>
<tr>
<td>NaNoroll, Inc.</td>
</tr>
<tr>
<td>NaNoroll, Inc.</td>
</tr>
</tbody>
</table>

The field types are based on the default XML schema data types. For example, the Date type is in the form YYYY-MM-DD.

### 7.3.5 Example: Mapping Fixed-Length Text Files to Databases

This example illustrates a data mapping operation between a fixed-length text file (FLF) text file and a Microsoft Access database. The files used in the example are available in the `<Documents>`\Altova\MapForce2019\MapForceExamples\Tutorial\ folder. Both the source text file and the target database store a list of employees. In the source file, the records are implicitly delimited by their size, as follows:

<table>
<thead>
<tr>
<th>Field position and name</th>
<th>Size (in characters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field 1 (First name)</td>
<td>8</td>
</tr>
<tr>
<td>Field position and name</td>
<td>Size (in characters)</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Field 2 (Last name)</td>
<td>10</td>
</tr>
<tr>
<td>Field 3 (Phone extension)</td>
<td>3</td>
</tr>
<tr>
<td>Field 4 (Email)</td>
<td>25</td>
</tr>
<tr>
<td>Field 5 (Position)</td>
<td>25</td>
</tr>
</tbody>
</table>

The goals of the mapping is to update the phone extension of each employee in the database to the one existing in the source file, while adding the prefix "100" to each extension. To achieve the goal, take the steps below.

**Step 1: Insert and configure the text component**

1. Select the menu option **Insert | Text file**, or click the insert Text file icon. This opens the "Insert Text Component" dialog box.

   ![Insert Text Component dialog box]

   Select **Use simple processing...** and click **Continue**.

2. Click the **Input file** button and select the file `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\Altova-FLF.txt` file. You will notice that the file is made up of a single string, and contains fill characters of type #.
3. Select **Fixed**.

4. Clear the **Assume record delimiters present** check box.

5. The three rows highlighted in yellow are editable, and enable you to specify i) the field name ii) the data type and iii) the field size. Type **8** as the new field size, and press **Enter**. More data is now visible in the first column, which is now defined as 8 characters wide.
6. Click **Append Field** to add a new field, and set the length of the second field to 10 characters.

7. Use the same method to create three more fields of the following lengths: 3, 25, and 25 characters, and change the field headers to make them easier to map: First, Last, Tel.-Ext, Email, Title. The preview will then look like this:

8. In the Fixed Length Field Settings group, select **Custom**, and type the hash (#) character. This instructs MapForce to treat the # character as fill character.

9. Click **OK**.

10. Click **Change component name**. The Text file component appears in the Mapping window. Data can now be mapped to and from this component.
Step 2: Insert the database component

1. Select the menu command **Insert | Database**, select **Microsoft Access**, and then click **Next**.
2. Select the **altova.mdb** database available in the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\` folder, and click **Connect**.
3. Select the **Person** table and click **OK**.

Step 3: Design the mapping

1. Drag the **core | concat** function from the Libraries window into the mapping.
2. Select the menu command **Insert | Constant**, select Number as type, and enter 100 as value. This constant stores the new telephone extension prefix.
3. Create the mapping as shown below.

4. On the database component, click the **Table Action** button next to **Person**.

5. Next to **Action on input data**, select **Update If**, and ensure that the action for **First** and **Last** fields is set to **equal**. This instructs MapForce to update the Person table only if the first and last name in the source file is equal to the corresponding database field. When this condition is true, the action taken is defined by the mapping. In this case, the telephone extension is prefixed by 100, and copied to the **PhoneExt** field of the Person table.
Step 4: Run the mapping
To generate the SQL statement (for preview in MapForce), click the Output tab. To run the SQL statements against the database, click the Run SQL-script button.

7.3.6 Setting the FLF Options
After you add a text component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component settings dialog box in one of the following ways:

- Select the component and, on the Component menu, click Properties.
- Double-click the component header.
- Right-click the component header, and then click Properties.
The available settings are as follows.

| Component name | The component name is automatically generated when you create the component. However, you can change the name at any time.

The component name can contain spaces and full stop characters. It may not contain slashes, backslashes, colons, double quotes, leading or trailing spaces. If you want to change the name of the component, be aware of the following:

- If you intend to deploy the mapping to FlowForce Server, the component name must be unique.

<table>
<thead>
<tr>
<th>Field</th>
<th>Field</th>
<th>Field</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>str1</td>
<td>str2</td>
<td>str3</td>
<td>str4</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>4</td>
<td>120</td>
</tr>
<tr>
<td>Albert Einstein</td>
<td>1879</td>
<td>Only two things are infinite, the un</td>
<td></td>
</tr>
</tbody>
</table>
It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line.

**Input file**

Specifies the file from which MapForce will read data. This field is meaningful for a source component and is filled when you first create the component and assign to it a text file. The field can remain empty if you are using the text file component as a target for your mapping.

In a source component, MapForce uses the value of this field to read column names and preview the contents of the instance text file.

To select a new file, click **Input File**.

**Output file**

Specifies the file to which MapForce will write data. This field is meaningful for a target component.

To select a new file, click **Output File**.

**Save all file paths relative to MFD file**

When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. This setting affects the input and output files used by the text component. See also [Using Relative Paths on a Component](#).

**Input / Output Encoding**

Allows you specify the following settings of the output instance file:

- Encoding name
- Byte order
- Whether the byte order mark (BOM) character should be included.

By default, any new components have the encoding defined in the **Default encoding for new components** option. You can access this option from **Tools | Options**, General tab.

**Fill Character**

This option allows you to define the characters that are to be used to complete, or fill in, the rest of the (fixed) field when the incoming data is shorter than the respective field definitions. The custom field allows you to define your own fill character in the Custom field.

If the incoming data already contains specific fill characters, and you enter the same fill character in the Custom field, then the incoming data will be stripped of those fill characters!

**Assume record delimiters present**

This option is useful when you want to read data from a source flat file that does not contain record delimiters such as CR/LF, or when you want to produce a target flat FLF file.
### Data Sources and Targets CSV and Text Files

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Treat empty fields as absent</strong></td>
<td>When this option is enabled, empty fields in the source file will not produce a corresponding empty item (element or attribute) in the target file.</td>
</tr>
<tr>
<td></td>
<td>Assuming that the output is an XML file, when this option is disabled, the empty fields will be created in the output with an empty value (in this example, the elements <code>Last</code>, <code>Title</code>, and <code>Email</code>):</td>
</tr>
<tr>
<td><img src="image" alt="XML Output Example" /></td>
<td>When this option is enabled, the empty fields will not be created in the output:</td>
</tr>
<tr>
<td><strong>CSV / Fixed</strong></td>
<td>Changes the component type to either CSV or FLF (fixed-length field).</td>
</tr>
<tr>
<td><strong>Preview area</strong></td>
<td>The lower part of the dialog box displays a preview of up to 20 rows of the file selected as input or output.</td>
</tr>
<tr>
<td></td>
<td>If necessary, you can create the structure of the file (or change the structure of the existing one), as follows.</td>
</tr>
<tr>
<td><strong>Append field</strong></td>
<td>Creates a new field after the last record.</td>
</tr>
<tr>
<td><strong>Insert field</strong></td>
<td>Creates a new field immediately before the currently selected record.</td>
</tr>
<tr>
<td><strong>Remove field</strong></td>
<td>Deletes the currently selected field.</td>
</tr>
<tr>
<td><strong>&lt;</strong></td>
<td>Moves the currently selected field one position to the left.</td>
</tr>
<tr>
<td><strong>&gt;&gt;</strong></td>
<td>Moves the currently selected field one position to the right.</td>
</tr>
<tr>
<td>To change the name of a field, click the header (in this example, <code>Field1</code>), and type the new value.</td>
<td></td>
</tr>
</tbody>
</table>
To change the data type of a field, select the required value from the drop-down list. MapForce checks the data type, so if the input data and the field format to do not agree, then the data is highlighted in red.

<table>
<thead>
<tr>
<th>Field1</th>
<th>string 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vernon##</td>
<td></td>
</tr>
<tr>
<td>Frank###</td>
<td></td>
</tr>
<tr>
<td>Loby####</td>
<td></td>
</tr>
<tr>
<td>Joe#####</td>
<td></td>
</tr>
<tr>
<td>Sus#####</td>
<td></td>
</tr>
<tr>
<td>Fred#####</td>
<td></td>
</tr>
</tbody>
</table>

To set the size of the field in characters, enter the field size in the third row from the top.

---

To better understand this option, open the Altova-FLF.txt file available in the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\` folder. Notice that the file consists of 71-character long records, without any delimiters such as CR/LF. If you would need to read data from this particular file, first you would need to split this file into records. That is, create several fields whose total size sums up to 71 characters (as shown below), and then disable **Assume record delimiters present**. For a step-by-step example, see Example: Mapping Fixed-Length Text Files to Databases.
If you would need to write data from this file to a destination file which uses the same structure, then enabling **Assume record delimiters present** creates a new record after every 71 characters.

The mapping result when "Assume record delimiters present" is enabled:

```
1  Vernon##Callaby##582v.callaby@nanonull.com##Office Manager##
2  Frank##Further##471f.further@nanonull.com##Accounts Receivable##
3  Loby##Matisse##983 L.tatisse@nanonull.com##Accounting Manager##
4  Joe##Firstbread621j.firstbread@nanonull.com##Marketing Manager Europe##
5  Sus##Sanna##753#.sanna@nanonull.com##Art Director##
6  Fred##Lands##952.l.lands@nanonull.com##Program Manager##
7  Michelle##Buckler##1654m.buckler@nanonull.com##Software Engineer##
8  Ted##Little##952x.little@nanonull.com##Software Engineer##
9  Ann##Way##951a.way@nanonull.com##Technical Writer##
10 Liza##Gardner##765l.gardner@nanonull.com##Software Engineer##
11 Paul##Smith##334p.smith@nanonull.com##Software Engineer##
12 Alex##Martin##778a.martin@nanonull.com##IT Manager##
13 George##Hamm##223g.hamm@nanonull.com##Web Developer##
14 Jessica##Baner##241j.baner@nanonull.com##Support Engineer##
15 Lu##King##345l.king@nanonull.com##Support Engineer##
16 Steve##Meier##114s.meier@nanonull.com##Office Manager##
17 Theo##Bone##951s.bone@nanonull.com##Accounts Receivable##
18 Max##Natta##122m.natta@nanonull.com##PA & Marketing Manager US##
19 Valentino##Bass##716v.bass@nanonull.com##IT Manager##
20 Car##Franken##147c.franken@nanonull.com##Support Engineer##
21 Mark##Redgraent##152m.redgreen@nanonull.com##Support Engineer##
22
```
The mapping result when "Assume record delimiters present" is disabled
7.4 MapForce FlexText

Altova web site: [Text and Flat File Mapping](#)

FlexText is a MapForce module which enables you to convert data from non-standard or legacy text files of high complexity to other formats supported by MapForce. While XML files have a schema or a structure from which MapForce can derive the schema, this is not the case of text files, especially when they have a complex and unique structure that does not consistently fit into CSV or fixed-length field patterns. Moreover, sometimes you need to extract only portions of useful data from a legacy text file. FlexText solves these problems by helping you define and test visually, in real-time, the rules, or the template, according to which text data must be split down into mappable items. A FlexText template essentially defines the structural model of your custom text data, according to criteria you specify, for the purpose of mapping it to other formats.

Once you define a FlexText template, you can add it to the MapForce mapping area as a source or target component, and thus map complex and non-standard text data with any other format supported by MapForce. FlexText template files have the .mft (MapForce FlexText) extension. You can reuse the same FlexText template in multiple mappings, to process any number of text files.

7.4.1 Overview

You can start FlexText directly from MapForce, when adding text files as mapping components (use the Insert | Text File menu command). You can also start FlexText as a standalone program, by running the Altova MapForce FlexText executable available in the MapForce...
The screen shot below illustrates a sample FlexText template (InspectionReport.mft) which is available, along with several other FlexText samples, in the <Documents>\Altova\MapForce2019\MapForceExamples directory.

The FlexText graphical Interface

The FlexText interface consists of the following panes: Design pane, Overview pane and Sample Text pane. As you will see next, the position and appearance of these panes can be customized according to your preferences.

**Design pane**

The Design pane is the working area where you define the structure of your text, by means of containers. Containers define the rules for handling text data according to criteria you specify. The first container of any FlexText template is the *Root* container, which represents the entire text file. All other containers essentially define the subsequent text processing logic (typically, splitting text into meaningful units from which you can map data). For example, the *Split Once* container splits a fragment of text into exactly two fragments. By default, the name of any container describes its function (for example, *Repeated Split*); however, you can change it if necessary. You
typically define as many containers as demanded by the structure of the text file with which you are working.

The FlexText Reference section of this documentation covers containers in more detail. However, to begin with, note the following about containers:

- Clicking the icon in the top-right corner of a container opens a pop-up menu from which you can select the container type.

- Each container has a number of options which dictate what happens to text data at that particular place in the structure. These options determine the content of the container, and enable you to refine it before providing it to the MapForce component.

- To show or hide container contents, click the Expand or Collapse icon, respectively.

- To collapse containers as a group, press the Shift key. Two chevrons appear. Clicking the handle collapses the section of the container tree to the right of the one clicked.

- To preview the text created by a particular container directly in the design pane, enable the Node Text in Design view toolbar button.

- When the Auto-collapse unselected node text toolbar button is enabled, the full contents of a container is expanded when you select it. All other containers are collapsed.

Overview pane

The Overview pane gives a bird’s-eye view of the Design pane. To navigate the Design pane, click and drag the red rectangle. To detach the Overview pane and reposition it elsewhere in the interface, click its title bar and drag the pane to the desired location.
Sample Text pane
The Sample Text pane displays the contents of the currently selected container. (Note that this pane is not shown if the Node Text in Design view toolbar button is enabled. Instead, the contents of the currently selected container is shown in the Design pane.)

To help you quickly identify the Tab and Carriage Return/Line Feed characters in the preview text, FlexText displays such characters using the \t and \r symbols, respectively.

7.4.2  FlexText Tutorial

The tutorial will show you how to use the most common, and most powerful, features of FlexText to process a text file and map its output in various ways in MapForce.

The example uses the Flex-tutorial.txt file available in the <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ folder. The .txt file has the following format:

```
111,332.1,22537.7,5,Container ship,Mega,
111,A1579227,10,3,400,Microtone,
111,S152427,7,6,1200,Miscellaneous,
222,978.4,7563.1,69,Air Freight,Mini,
222,Z2AW561,10,5,10000,Gas Chromatograph,

General outgassing pollutants
1100,897,22.1,716235,LOx
1110,9832,22991.3,002,NOx
1120,1213,33.01,008,SOx
```

Aim of the tutorial:
- To separate out the records containing 111, and 222 keys, into separately mappable items.
- To discard the plain text record.
- To create a CSV file of the remaining records.

The tutorial is organized into the following parts:
- **Step 1: Create the FlexText Template**
- **Step 2: Define Split Conditions**
- **Step 3: Define Multiple Conditions per Container**
- **Step 4: Create the Target MapForce Component**
- **Step 5: Use the FlexText Template in MapForce**

7.4.2.1  Step 1: Create the FlexText Template

To create the FlexText template:

2. Select **Insert | Text file**, or click the Insert Text file icon.
3. Click the "**Design graphically new structure** ..." button.

4. Enter a name for your FlexText template, and click Save to continue (e.g. Flex-tutorial.mft). An empty design, along with the "Open" dialog box are displayed.

5. Select the **Flex-tutorial.txt** file in the ...\MapForceExamples\Tutorial folder, and confirm by clicking Open.
The text file contents are now visible.

Clicking the "Node Text in Design view" icon, displays the active container contents, in the Sample Text pane.

Activating "Auto-collapse unselected node text", displays the content in the active container, all other containers which contain content, are collapsed.

6. Click the container icon at the top right, and select Split once from the pop-up menu.
   Two new containers appear next to the Split once container. For more information on the Split once condition, please see: Split once.
The default settings of the Split once container are visible: fixed length, horizontal and split line=1.

The result of these default settings are also visible:

- The top container contains the first line of the text file, highlighted in the Split once container.
- The lower container contains the rest of the text file.

### 7.4.2.2 Step 2: Define Split Conditions

Split conditions allow you to segment text fragments in various ways. To define split conditions:

1. Click the Mode combo box and select "delimited (line based)".
2. Double-click the Separator field and enter "General".
The text fragments in the respective containers have now changed.

Entering "General" and using delimited (line based), allows you to split off that section of text that contains the string "General", into the lower container. The text fragment up to the separator, is placed in the top container.

What we want to do now, is work on the lower container to produce a CSV file containing the records with 1100 and up.

3. Click the lower container and change it to Split once.

Two new containers are created. The default settings can remain as they are, because we now want to split off the first line of this text fragment, and ignore it. The remaining fragment in the lower container will be made into a CSV file.

4. Click the top container and change it to Ignore.
5. Click the lower container icon and change it to **Store as CSV**.

The container now shows the text fragment in a tabular form. The default settings can be retained.

**Configuring the CSV file**

If you want to change the field names, click the field, in the table, and then change the entry in the **Name** field. Columns can also be appended, inserted and deleted in this container, please see “**Store as CSV**” for more information.

We can now continue with defining the remaining text fragment.
7.4.2.3  **Step 3: Define Multiple Conditions per Container**

FlexText allows you to define multiple conditions per text fragment, using the Switch container. An associated container is automatically allocated to each condition that you define.

The current state of the tutorial at this point is that lower text fragment, of the first **Split once** container, has been defined:

- A "Split once" container splits off the first line into an Ignore container.
- The remaining segment is defined/stored as a CSV file.

1. Click the top container icon and change it to **Repeated split**.
The default settings are what we need at this point. The text fragment is split into multiple text blocks of a single line each. The associated container shows a preview of each of the text blocks.

Clicking the Next text block icon allows you to cycle through all the text fragments, of which there are 6.

2. Click the new container and change it to **Switch**.

The initial state of the Switch container is shown above.

- An associated container "default", has been added.
- The content of the first record 1/6, is displayed in the default container.

3. Click the Append condition icon in the "Switch" title bar, to add a new condition.
4. Double click in the field "Content starts with", and enter 111.
This defines the first condition. An associated container (Content starts with 111) has been added above the "default" container.

5. Click the append icon again, and enter "222" in the Content starts with field.

A third container has been added (Content starts with 222).

**Note:** Clicking the "Contents starts with" combo box, allows you to select the "Contains" option. This allows you to specify a "string" which can occur anywhere in the text fragment.

6. Click the Next text block icon several times to see the effect.
Upon reaching record 4 of 6, container 222 opens up, and displays its content.

7. Continue clicking, till you reach record 6 of 6. A single CR / LF character is displayed in the default container.

If a data fragment in the current block satisfies a condition, then the complete data of that block is passed on to the associated container. Data is not split up in any way, it is just routed to the associated container, or to the default container if it does not satisfy any of the defined conditions.

8. Click the first two containers and change them to **Store as value**. Click the last container and change it to **Ignore**.
9. Double-click the "Store" text, and add descriptive text e.g Key 111 and Key 222.
10. Save the FlexText template, e.g. Flex-Tutorial.mft.

7.4.2.4 Step 4: Create the Target MapForce Component

To create the target component for use with the FlexText source:

1. Select **Insert | Text file**, or click the Insert Text file icon.
2. Click the "Use simple processing..." radio button and click **Continue**. This opens the Component Settings dialog box.
3. Click the **Append Field** button to add a new field.
4. Double click the Field1 field name and change it to **Key 111**.
5. Do the same for the other field, and name it **Key 222** and click OK to confirm. A text component with two fields has now been created.
6. Use the same method to create a second text component that consists of five fields.
7.4.2.5  **Step 5: Use the FlexText Template in MapForce**

To use the FlexText template in MapForce:

1. Start, or switch back to MapForce, and select **Insert | Text file**.
2. Click the **Open existing FlexText configuration file...** button and select the previously defined FlexText template (**Flex-tutorial.mft**). The structure of the MapForce component mirrors that of the containers in Design view in FlexText.
3. Map the various items to the previously defined target components, and click the Output tab to preview the results.

Mapping preview of the top text component:

```
1 "111,392.1,22537.7,5,Container ship,Mega,
2 ",
3 "111,Al579227,10,3,400,Microtome,
4 ",
5 "111,B152427,7,6,1200,Miscellaneous,
6 ",
7 ";222,978.4,7563.1,69,Air freight,Mini,
8 ",
9 ";222,ZZAW561,10,5,10000,Gas Chromatograph,,
10 ",
11
```

Mapping preview of the lower text component:

```
1 1100,897,22.1,716235,LOX
2 1110,9832,22991,30,002,NOX
3 1120,1213,33.01,000,SOX
4
```

### 7.4.3 FlexText Component Settings

After you add a FlexText component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component settings dialog box in one of the following ways:
On the **Component** menu, click **Properties** (this menu item becomes enabled when you select a component).

- Double-click the component header.
- Right-click the component header, and then click **Properties**.

The available settings are as follows.

<table>
<thead>
<tr>
<th><strong>Component name</strong></th>
<th>The component name is automatically generated when you create the component. You can however change the name at any time.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If the component name was automatically generated and you select an instance file after that, MapForce will prompt you to optionally update the component name as well.</td>
</tr>
<tr>
<td></td>
<td>The component name can contain spaces and full stop characters. The component name may not contain slashes, backslashes, colons, double quotes, leading or trailing spaces. In general, be aware of the following implications when changing the name of the component:</td>
</tr>
<tr>
<td></td>
<td>- If you intend to deploy the mapping to FlowForce Server, the component name must be unique.</td>
</tr>
<tr>
<td></td>
<td>- It is recommended to use only characters that can be entered at the command line. National characters may have different encodings in Windows and at the command line.</td>
</tr>
</tbody>
</table>
### FlexText Configuration

Specifies the name or path of the FlexText template (.mft) used by the component.

To select a new template, click **Browse**. To edit the template in FlexText, click **Edit**.

### Input Text File

Specifies the text file from which MapForce will read data. This field is meaningful for a source component and is filled when you first create the component and assign to it a text file.

The source text file specified here takes precedence over the one defined in the FlexText template. If this field is empty, the filename defined in the FlexText template is used as source.

### Output Text File (for Code Generation)

Specifies the name or path of the text file to which MapForce will write data. This field is meaningful for a target component when generating code. Entering a full path allows you to specifically define the target directory, for example, `c:\myfiles\sequence.txt`.

### Save all file paths relative to MFD file

When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file.

### 7.4.4 Using FlexText as a Target Component

The main use of FlexText components is to reduce, or split off sections of text files and map the relevant items of the FlexText component to other target components. FlexText can, however, also be used as target component to assemble or reconstitute separate files into a single file, although this can entail a fair amount of trial and error to achieve the desired results.

Using a FlexText component as a **target** reverses the operations previously defined in it. Instead of splitting a file into various subsections, you assemble/reconstitute a file.

In general, the inverse of each operation defined in the FlexText template is carried out (in a bottom-up fashion) when using it as a target:

- "Split" becomes "merge", e.g. mapping to a repeated split delimited by ",," becomes a merge items separated by ",".
- "Store" becomes "load".
- "Switch" becomes "choose the first match".

Note the following when using FlexText as target components:

- As soon as a connection is made between a data source component and one of the input items of a FlexText component, the FlexText component data source is ignored. The data provided by the newly mapped source component now takes precedence.
- If text is mapped to a "Store as..." (Store as CSV and FLF) container, then the separator is retained. However, text might be truncated if a fixed length split occurs in a node above the "Store as..." node.
• Fixed Width Splits truncate the left/top section if Split Base=Head, or the right/bottom section if Split Base=tail, to the predefined length. The truncated section is then as long as the defined length in characters. If the text is too short, then space characters are inserted to pad the section.
• FlexText would normally insert separators (or white space for fixed splits) between the items of a split operation, but this is not the case for 'delimited (line based)' splits. The 'delimited (line based)' operation is not a perfectly reversible operation. The "Delimited" text may occur anywhere in the first line and is included in the text, and therefore an automatic process cannot reliably add it.
  o Delimited (line based), will not add a separator to the first line if it is missing.
  o Delimited (floating), will add a separator between two sections.
• The switch operation cannot be inverted in a meaningful way except for simple cases. The switch scans its branches for the first branch that contains data, and uses/inserts this data. Only the first connection of a switch operation is mapped. To transfer data to the remaining switch containers, filters have to be defined for the remaining connectors and the duplication of the switch parent item is necessary, so that each switch item returns a single item which is then fed to a repeated split item to merge all of them.
• Mapping to a child of a single split container discards all mapping results except for the last item. Only a single result is retained, even if multiple results were generated.

The following analogy to the XML Schema content model gives some idea of FlexText's behavior when used as a target:

• A repeated split is a repeatable element.
• A single split forms part of a 'sequence' content model group.
• A switch forms a 'choice' content model group, each case being a possible child element.
• A store creates an element of simple type.

7.4.5 FlexText Reference

The reference section describes the various features of FlexText and shows how to use them to achieve specific results.

7.4.5.1 Repeated split

Using this option initially creates a single container. The container contains the text defined by the condition set in Repeated Split. There are several versions of the Repeated split option: Fixed length, Delimited Floating, Delimited Line based, and Delimited Starts with... 

When you first select this option, default parameters are automatically set and the resultant fragments appear in the associated container. Note that the Repeated Split container is currently active, and the preview displays all current records/lines, in the Sample Text pane.

Container default settings:

<table>
<thead>
<tr>
<th>Mode</th>
<th>fixed length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orientation</td>
<td>horizontal</td>
</tr>
</tbody>
</table>
Offset 1

Default result:
Each line of text appears as a line/record in the new container, as the Offset is 1. Click the new container to preview its contents. The Sample Text scroll arrows, let you scroll through each of the 11 blocks/fragments produced by these settings.

7.4.5.1.1 Mode - Fixed length

Use the Repeated split (fixed length) mode when you want to split text into multiple horizontal or vertical fragments of fixed length. The settings applicable to the Repeated split (fixed length) mode are described below.

Orientation
Specifies if the text fragment is to be split horizontally or vertically. Choose “Horizontal” to split the fragment into multiple horizontal sections. Choose “Vertical” to split the fragment into multiple vertical columns. The default orientation is “Horizontal”.
Offset

If orientation is set to "Horizontal", this setting specifies the number of lines that each fragment should contain. If orientation is set to "Vertical", this setting specifies the width in characters of each fragment. The default offset is "1". To modify the offset, do one of the following:

- Enter a value into the Offset field
- Drag the tab on the vertical or horizontal ruler.

7.4.5.1.2  Mode - Delimited (floating)

Use the Repeated split with delimited (floating) mode in the following situations:

- To split text where the separator characters that you specify must be stripped out from the resulting fragments
- To split text where the separators are in-line (for example, text that doesn't contain CR/LF characters)

**Note:** A fragment is defined as the text between the first character after the separator, up to the last character before the next instance of the same separator. An exception to this rule are the first and last fragments, as shown in the example below.

The settings applicable to the Repeated split with delimited (floating) mode are described below.

**Regular expression**

This is an optional setting which splits text into fragments whenever there is a regular expression match (see Splitting Text with Regular Expressions). The default value is "no".
Separator

Specifies the character(s) to be used as separator. The default value is "none" (no separator).

For example, using the separator "222," against the text shown below produces three separate fragments.

The first fragment contains all characters from the start of the fragment to the start of the first separator ("222."), that is, from "111" to "Miscellaneous.".

If the separator is not the first set of characters of the first line in the fragment, as in this example, then the first fragment includes all the text up to the first instance of the separator (for example, "222").

If "111" were the separator, then the first fragment would be a zero-length string, as the separator appears at the beginning of the first line of the source fragment.

The second fragment contains the first line containing the separator 222, without the separator.
The third fragment contains the next line containing the separator 222, without the separator itself, up to the end of the text file/fragment.

**7.4.5.1.3 Mode - Delimited (line based)**

Use the **Repeated split with delimited (line based)** mode to split text into multiple fragments, with the following behaviour:

- This mode creates multiple fragments defined by separator characters that you enter in the Separator field.
- The separator characters are included in the fragment.
- A fragment is defined as the entire line containing the separator, up to the next line containing the same separator.
- If the separator does not appear in the first line, then the first fragment contains the line(s) up to the first line containing the separator.

The settings applicable to the **Repeated split with delimited (line based)** mode are listed below.

**Regular expression**

This is an optional setting which splits text into fragments whenever there is a regular expression match (see **Splitting Text with Regular Expressions**). The default value is "no".

**Separator**

Specifies the character(s) to be used as separator. The default value is "none" (no separator).

For example, using the separator "HDR" against the text shown below produces three separate fragments.
The first fragment contains all characters from the start of the file/fragment, including all lines up to the next line containing the same separator.

Note that this option allows you access to any number of lines between two separators. This is useful in files with record types that are optional or not in sequence (such as "DTL" or "TXT" in this example).

The second fragment contains all text from the second occurrence of "HDR" up to the next occurrence of "HDR".

The third fragment contains all text from the third occurrence of "HDR" up to the end.
7.4.5.1.4  **Mode - Delimited (line starts with)**

Use the **Repeated split with delimited (line starts with)** mode to split text into multiple fragments, with the following behaviour:

- This mode creates multiple fragments defined by separator characters that you enter in the Separator field.
- The separator characters are included in the fragment.
- A fragment is defined as the entire line, starting with the separator, up to the next line containing the same separator at the start of the line.
- If the separator does not appear in the first line, then the first fragment contains the line(s) up to the first line containing the separator.

The settings applicable to the **Repeated split with delimited (line starts with)** mode are listed below.

**Regular expression**

This is an optional setting which splits text into fragments whenever there is a regular expression match (see [Splitting Text with Regular Expressions](#)). The default value is "no".

**Separator**

Specifies the character(s) to be used as separator. The default value is "none" (no separator).

For example, using the separator "22" against the text below produces three separate fragments:

```
1 111, 332.1, 22537.7, 5, Container ship, Mega, 2
2 111, A1579227, 10, 3, 400, Microtome, 2
3 111, B152437, 7, 6, 1200, Miscellaneous, 2
4 222, 978.4, 7563.1, 69, Air freight, Mini, 2
5 222, ZA&W561, 10, 5, 10000, Gas Chromatograph, 2
6 222, A157931, 10, 6, 1200, Miscellaneous, 2
7 General outgassing pollutants 2
8 1100, 897, 22.1, 716235, LOX 2
9 1110, 9832, 22991.30, 002, NOX 2
10 1120, 1213, 33.01, 008, SOX 2
```

The first fragment contains all characters from the start of the file/fragment, including all lines up
to the line containing the separator "22".

<table>
<thead>
<tr>
<th>Sample Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111,332.1,22537.7,5,Container ship,Mega,</td>
</tr>
<tr>
<td>2111,A1579227,10,3,400,Microtome,</td>
</tr>
<tr>
<td>3111,B152427,7,6,1200,Miscellaneous,</td>
</tr>
</tbody>
</table>

The second fragment contains all characters/lines from the first occurrence of "22", up to the next occurrence of "22", which in this case is only one line.

<table>
<thead>
<tr>
<th>Sample Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1222,978.4,7563.1,69,Air freight,Mini,</td>
</tr>
</tbody>
</table>

The third fragment contains all characters/lines from the second occurrence of "22", up to the end.

<table>
<thead>
<tr>
<th>Sample Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1222,ZIAW561,10,5,10000, Gas Chromatograph,</td>
</tr>
<tr>
<td>2General outgassing pollutants</td>
</tr>
<tr>
<td>41100,897,22.1,716235,LOX</td>
</tr>
<tr>
<td>51110,9832,22991.30,002,NOX</td>
</tr>
<tr>
<td>61120,1213,33.01,008,SOX</td>
</tr>
</tbody>
</table>

By contrast, here is what would happen if we used the delimited (line based) mode and separator as "22":

<table>
<thead>
<tr>
<th>Sample Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1222,ZIAW561,10,5,10000, Gas Chromatograph,</td>
</tr>
<tr>
<td>2General outgassing pollutants</td>
</tr>
<tr>
<td>41100,897,22.1,716235,LOX</td>
</tr>
<tr>
<td>51110,9832,22991.30,002,NOX</td>
</tr>
<tr>
<td>61120,1213,33.01,008,SOX</td>
</tr>
</tbody>
</table>
There would be six fragments, composed of lines that contained 22 anywhere in that line.

### 7.4.5.2 Split once

Using this option creates two vertically aligned containers. The top container contains the text defined by the condition set in the **Split once** container. The bottom container contains the rest of the text file/fragment. There are several versions of the Split once option: **Fixed** length, Delimited **Floating**, and Delimited **Line Based**.

When you first select this option, default parameters are automatically set, and the resultant fragments appear in both containers. Note that the **Split once** container is currently active, and displays a preview of all current records/lines, in the Sample Text pane.

**Container default settings** are:

- **Mode**: fixed length
- **Orientation**: horizontal
- **Split line**: 1
- **Split base**: head
Default result:
The first line of text appears in the top container. The bottom container contains the rest of the text file/fragment.

7.4.5.2.1 Mode - Fixed length

Use the Split once (fixed length) mode when you want to split text into two horizontal or vertical fragments, at a particular line or column relative to the beginning or end of the text. The settings applicable to the Split once (fixed length) mode are described below.

Orientation
Specifies if the text fragment is to be split horizontally or vertically. Choose “Horizontal” to split the fragment into two horizontal sections. Choose “Vertical” to split the fragment into two vertical sections. The default orientation is “Horizontal”.

Split Line
Specifies the number of lines (or columns) after which the fragment should be divided into two. The default offset is “1”. To modify this setting, do one of the following:

- Enter a value into the Split line field
- Drag the tab on the vertical or horizontal ruler.
Mode - Delimited (floating)

Use the **Split once with delimited (floating)** mode to split text into two fragments, using a custom separator that is anywhere in the text. This is generally useful in files that do not contain CR, or LF characters, and you want to split the fragment into two, at some specific in-line location. Note the following:

- This mode creates two fragments defined by separator characters that you enter in the Separator field.
- The separator characters are not included in the fragment.
- The first fragment is defined as the text between the first character of the file/fragment, up to the last character before the separator.
- The second fragment is defined as the first character after the separator, up to the last character in the file/fragment.
- If the separator appears in the first/last position of the file/fragment, then the first of the two resulting containers remains empty.

The settings applicable to the **Split once with delimited (floating)** mode are listed below.

**Regular expression**

This is an optional setting which splits text into two fragments when there is a regular expression match (see [Splitting Text with Regular Expressions](#)). The default value is "no".

**Separator**

Specifies the character(s) to be used as separator. The default value is "none" (no separator).
7.4.5.2.3  Mode - Delimited (line based)

Use the **Split once with delimited (line based)** mode to split text into two fragments, where the separator is anywhere in one of the lines. The line containing the separator is not split, but is retained whole. This is generally useful in files containing record delimiters (CR/LF), and you want to split the fragment into two separate fragments. Note the following:

- This mode creates two fragments defined by separator characters that you enter in the Separator field.
- The separator characters are included in the fragment.
- The first fragment is defined as all the text up to the line containing the separator.
- The second fragment is defined as the text, and line, including the separator up to the end of the file/fragment.
- If the separator appears in the first/last line, of the file/fragment, then the top container remains empty.

The settings applicable to the **Split once with delimited (line based)** mode are listed below.

**Regular expression**

This is an optional setting which splits text into two fragments when there is a regular expression match (see [Splitting Text with Regular Expressions](#)). The default value is "no".

**Separator**

Specifies the character(s) to be used as separator. The default value is "none" (no separator).

For example, if you use the separator "1200," against the text below, two fragments are created.
The first fragment contains the text up to the line containing the separator.

```
1111,332.1,22537.7,5,Container ship,Mega,,
2111,A1579227,10,3,400, Microtome,,
3111,B152427,7,6,1200, Miscellaneous,,
4222,978.4,7563.1,69,Air freight,Mini,,
5222,ZZAW561,10,5,10000, Gas Chromatograph,,
```

The second fragment contains the entire line containing the separator, and all remaining lines up to the end.

```
1111,332.1,22537.7,5,Container ship,Mega,,
2111,A1579227,10,3,400, Microtome,,
3111,B152427,7,6,1200, Miscellaneous,,
4222,978.4,7563.1,69,Air freight,Mini,,
5222,ZZAW561,10,5,10000, Gas Chromatograph,,
```

**Split base**

<table>
<thead>
<tr>
<th>Split base</th>
<th>Head</th>
<th>Look for the first occurrence of the separator starting from the beginning of text.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tail</td>
<td>Look for the first occurrence of the separator starting from the end of text.</td>
</tr>
</tbody>
</table>

### 7.4.5.2.4 Mode - Delimited (line starts with)

Use the **Split once with delimited (line starts with)** mode if the split should occur at the first line which begins with the specified separator. When you select this mode, two fragments of text are created, as follows:

- The first fragment contains all the text up to the line where the separator is.
- The second fragment contains the remaining text, including the line where the separator is.
- If the separator is at the beginning of the first line, no split occurs (that is, the first resulting fragment remains empty).
The settings applicable to the are listed below.

Regular expression
This is an optional setting which splits text into two fragments when there is a regular expression match (see Splitting Text with Regular Expressions). The default value is "no".

Separator
Specifies the character(s) to be used as separator. The default value is "none" (no separator).

Split base

<table>
<thead>
<tr>
<th>head</th>
<th>Look for the first occurrence of the separator starting from the beginning of text.</th>
</tr>
</thead>
<tbody>
<tr>
<td>tail</td>
<td>Look for the first occurrence of the separator starting from the end of text.</td>
</tr>
</tbody>
</table>

For example, in the image below, the split occurs at the first line which begins with the specified separator ("General outgassing pollutants"). If the Split base option was set to tail, then the split would have occurred at the first line which begins with the specified separator, starting from the end (which in this example happens to be the same line).

The first resulting fragment is as follows:

```
1111,332.1,22537.7,5,Container ship,Mega,¶
2111,A1579227,10,3,400,Microtome,¶
3111,B152427,7,6,1200,Miscellaneous,¶
4222,978.4,7563.1,69,Air freight,Mini,¶
5222,ZZAWS61,10,5,10000,Gas Chromatograph,¶
6¶
7General outgassing pollutants¶
81100,897,22.1,716235,LOX¶
91110,9832,22991.30,002,NOX¶
101120,1213,33.01,008,SOX
```

The second resulting fragment is as follows:

```
1111,332.1,22537.7,5,Container ship,Mega,¶
2111,A1579227,10,3,400,Microtome,¶
3111,B152427,7,6,1200,Miscellaneous,¶
4222,978.4,7563.1,69,Air freight,Mini,¶
5222,ZZAWS61,10,5,10000,Gas Chromatograph,¶
6¶
```
Using the Switch option allows you to define multiple keywords, or conditions, for a single text fragment. Each keyword you define, has its own associated container which receives data only if the specific condition is satisfied, i.e. true. If none of the conditions are satisfied, then the specific fragment is mapped to a "default" container.

Container **default settings** are:

Output For the first triggered condition.

The example below processes a Tomcat log file, where the individual processes are to be separated out, and made mappable. When you first define a Switch container, only the **default** container appears to the right of the Switch container. All data is automatically passed on to it.

The repeated split container, using delimited (line based), separates all INFO sections out of the log file and passes them on to the Switch container.

1. Click the append icon to add a new condition to the Switch container.
2. Double click in the "Content starts with" field, enter "Initializ" and hit Return.
A new container is added. Data will be forwarded to this container if the condition is true. If not, the data is forwarded to the default container.

3. Click the "Content starts with" combo box, and change it to "Content contains". The first condition has now been defined and you can see the result below. The first fragment does not contain "Initializ", and its contents are therefore forwarded to the **default** container.

4. Click the Display next block icon to see the next text fragment.

The Initializing... fragment now appears in its associated container, and the default container is empty. Stepping through the fragments gives you a preview of what the individual containers hold.

5. Click the container icon button, and select **Store as value**.
6. Double click in the "Store" title bar and change the text e.g. Initialize.

7. Click the append icon to add a new condition to the Switch container.
8. Double click in the "Content starts with" field, enter "Starting" and hit Return. You can add as many conditions as you need e.g. Pausing, and Stopping. Give each of the associated containers a name, to make recognition in MapForce easier.

The screenshot above shows all four conditions, and the contents of the "Starting" container at block/fragment no 4. The associated containers have all been renamed to make identification in the MapForce component easier.
Note that conditions can be moved up and down in the condition list, using the respective Move Up/Down buttons, or .

9. Save the template and insert it in MapForce.

Note: If a text fragment in the current fragment satisfies a condition, then the complete data of that fragment is passed on to the associated container. Data is not split up in any way, it is just routed to the associated containers, or to the default container if it does not satisfy any of the defined conditions.

The associated containers produced by Switch, can be used for further processing. You can change such a container to Split once, Repeated split, or anything else if you wish.

**Content starts with:**
Data is only passed to the associated container, if the condition string appears at the start of the text fragment.

**Content contains:**
Data is passed on to the associated container, if the condition string appears anywhere in the text fragment.

**For the first triggered condition:**
Data is passed on when one of the conditions in the condition list is true. Any other conditions that are true are ignored, and no data is passed on to any of the associated containers.

**For all triggered conditions:**
Outputs data for every condition that is true in the condition list. This makes it possible to have multiple occurrences of the same data/fragment in multiple associated containers simultaneously. This might occur if a text fragment contains text that satisfies two conditions simultaneously e.g. "initializing starting sequence" in the example above.
7.4.5.4 Node

Allows you to add a new hierarchical level to the FlexText, and MapForce tree structures. The data that the following node/container contains, is passed on as is.

In the screenshot below, the "All 111 Order No." container is the last container in the top branch.

Click the top-right icon of the container, and select **Node** from the context menu.

A new container has been added to the right of the current one.

**Note:** The automatically appended container was then manually defined as "Store as value".

The screenshot below shows both template structures as they appear when inserted into MapForce.
The left component shows the initial structure before adding the new Node.

The right component shows how the component structure has changed. "All 111..." is now a parent item, and a new child item "Store" has been added below it.

7.4.5.5 Ignore

Allows you to suppress the output of a specific text fragment. What this means, is that the container and any data it may contain, will not be made available as a mappable item in the FlexText component in MapForce.

In the example shown above, the active container has been set to "Ignore". The Sample text that it contains will therefore not appear as a mappable item in MapForce.

The text template when inserted into MapForce, has the structure shown below. There is no mappable item between the two "Split once" items.
Note: Default "ignore" containers also exist. These are the new containers that are automatically appended when selecting “Split once” and “Repeated split” etc.

The contents of these containers are not initially mappable/available to MapForce when the template is inserted. You have to select one of the container options in FlexText: Store as value, Store as CSV etc., to be able to map them.

7.4.5.6 Store as CSV (delimited)

Store CSV allows you to store text fragments as CSV text, and map individual columns to MapForce. Any number of CSV containers/components can be created in FlexText, and each of the CSV containers may have different separators.

The Sample Text pane provides an overview of the current CSV fragment, and also allows you to specify individual field names, and field types. Each column appears as a mappable item in the FlexText component in MapForce.

Container default settings are:

- Record separator: CR LF
- Field separator: ,
- Quote character: "
- Add quote character always: no
- Escape character: (none)
- First row contains field names: no
- Treat empty fields as absent: yes

The following example shows how data in a small text file is split up into two CSV files, and mapped to separate XML files in MapForce.
The **Split once** container shown above, is used to create two containers. The **delimited (line based)** function with the separator 222, is used to achieve this. All records up to the first occurrence of 222, are passed to the CSV container. The first, consisting of all records containing 111, is then defined as a CSV container. The Sample Text pane shows the contents of the currently active container "Split once".

<table>
<thead>
<tr>
<th>Sample Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111,332.1,22537.7,5,Container ship,Mega,1</td>
</tr>
<tr>
<td>2111,A1579227,10,3,400,Microtome,2</td>
</tr>
<tr>
<td>3111,B152427,7,6,1200,Miscellaneous,3</td>
</tr>
<tr>
<td>4222,978.4,7563.1,69,Air freight,Mini,4</td>
</tr>
<tr>
<td>5222,ZZA#551,10,5,10000,Gas Chromatograph,5</td>
</tr>
<tr>
<td>6General outgassing pollutants,6</td>
</tr>
<tr>
<td>71100,897,22.1,716235,LOX,7</td>
</tr>
<tr>
<td>81110,9832,22991.30,002,NCX,8</td>
</tr>
<tr>
<td>91120,1213,33.01,008,SOX,9</td>
</tr>
</tbody>
</table>

The default CSV settings have not been changed. Clicking the CSV container shows its contents in tabular form.
The second container holds the remaining data, and is made into another **Split once** container. This creates two more containers, one of which will be the second CSV. Clicking the Split once container, shows the current contents.
The delimited (line based) function, using 1100 as the separator, is used to split the remaining data into two sections.

- All records up to the first occurrence of 1100, are passed to the first container which is made non-mappable, by defining it as "Ignore".
- The second container is then defined as CSV. The default settings have not been changed. Clicking the CSV container shows the contents in tabular form.
Inserting the FlexText template into MapForce allows you to map the data to any of the supported target files. In this example, each of the CSV items are mapped to two separate XML files.
Note that not all of the items in the CSV sections are mapped to the target files. The first XML file contains all 111 record types.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Header>
    <RecordType>111</RecordType>
    <OrderNo>332</OrderNo>
    <TotalWeight>2237.7</TotalWeight>
  </Header>
  <Header>
    <RecordType>111</RecordType>
    <OrderNo>0</OrderNo>
    <TotalWeight>0</TotalWeight>
  </Header>
  <Header>
    <RecordType>111</RecordType>
    <OrderNo>0</OrderNo>
    <TotalWeight>7</TotalWeight>
  </Header>
</Order>
```

The second XML file contains all records starting with 1100.
Configuring the CSV container/data

```
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Header>
    <RecordType>1100</RecordType>
    <OrderNo>897</OrderNo>
    <TotalWeight>22.1</TotalWeight>
    <TotalUnitCost>716235</TotalUnitCost>
  </Header>
</Order>

<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Header>
    <RecordType>1110</RecordType>
    <OrderNo>9832</OrderNo>
    <TotalWeight>22991.3</TotalWeight>
    <TotalUnitCost>2</TotalUnitCost>
  </Header>
</Order>

<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Header>
    <RecordType>1120</RecordType>
    <OrderNo>1213</OrderNo>
    <TotalWeight>33.01</TotalWeight>
    <TotalUnitCost>8</TotalUnitCost>
  </Header>
</Order>
```

---

### Configuring the CSV container/data

**Split once**
- **Mode**: delimited (line based)
- **Separator**: 1100
- **Split base**: head

**CSV**
- **Record separator**: `<GR><LF>`
- **Field separator**: ,
- **Quote character**: "
- **Add quote character always**: no
- **Escape character**: \
- **First row contains field names**: no
- **Treat empty fields as absent**: yes

### Sample Text

<table>
<thead>
<tr>
<th>Field</th>
<th>Field2</th>
<th>Field3</th>
<th>Field4</th>
<th>Field5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100</td>
<td>897</td>
<td>22.1</td>
<td>716235</td>
<td>LOX</td>
</tr>
<tr>
<td>1110</td>
<td>9832</td>
<td>22991.30</td>
<td>002</td>
<td>NOX</td>
</tr>
<tr>
<td>1120</td>
<td>1213</td>
<td>33.01</td>
<td>008</td>
<td>SOX</td>
</tr>
</tbody>
</table>
Clicking a field in the Sample Text pane highlights it, allowing you to configure it further.

- Click in the **Name** field to edit the default text that is presented.
- Click in the **Type** field to define the field data type: string, boolean, decimal etc.
- Click the append icon to append a new field.
- Click the insert icon to insert a field before the currently active field.
- Click the delete icon to delete the currently active field.

**Note:** The field boundaries can be dragged by the mouse to display the data.

**Add quote character always**

Allows you define if the specified quote character is to be added to all fields of the generated CSV file.

**Treat empty fields as absent**

Allows you to define that empty fields in the source file, will not produce a corresponding empty item (element or attribute) in the target file.

Note that the delimiters for the empty fields in the source file must exist however, e.g. "General outgassing pollutants,,,".

### 7.4.5.7 Store as FLF (fixed length)

![FLF](image)

Store FLF allows you to store text fragments as fixed length text, and map individual columns to MapForce. Any number of FLF containers/components can be created in FlexText, and each of the FLF containers may have different fill characters.

The Sample Text pane provides an overview of the current FLF fragment, and also allows you to specify field names, lengths, and widths. Each column appears as a mappable item in the text component in MapForce.

Container **default settings** are:

- Fill character: (none)
- First row contains field names: no
- Treat empty fields as absent: yes
Configuring the FLF container/data

Having defined a container as "Store FLF", the Sample Text pane appears as shown in the screenshot above. A default field of width 10 is automatically inserted.

- Click the tab icon on the ruler and drag to reposition it. A tip appears showing you the current position.
- Positioning the cursor over the ruler displays a "dot"; clicking places a new tab at the click position.

- Having defined the new position, click the field to select it, and edit the name in the Name field.
To remove a field, click the tab icon and drag it off the ruler. The tab icon changes when this action can be successfully completed.

Note: When you drag a tab on the ruler, all tabs to the right of it will be automatically repositioned. To retain the other tab positions, hold down the Shift key before moving the tab.

Clicking a field in the Sample Text pane highlights it, allowing you to further configure it.

- Click the append icon to append a new field, of length 10.
- Click the insert icon to insert a field before the currently active field, length 10.
- Click the delete icon to delete the currently active field.
- Click in the Name field to edit the default text that is presented.
- Click in the Type field to define the field data type: string, boolean, decimal etc.

Inserting the FlexText template into MapForce allows you to map the data to any of the supported target files. In this example, FLF items are mapped to XML items.

If the option Treat empty fields as absent is yes, then any empty fields in the source file will not produce a corresponding empty item (element or attribute) in the target file. A field is considered
as absent if there is no data between two subsequent fill characters.

7.4.5.8  **Store value**

Allows you to define a container, which makes its data available as a mappable item, in MapForce. If you do not change the container name in FlexText, then the mappable item appears with the name "Store".

Container **default settings** are:

- **Type**: string
- **Trim**: no

The screenshot below shows the "Store" container with its contents visible in the Sample Text pane.

Saving this template and opening it in MapForce, allows you to map the Store item to other items in a target component.

**Note:** The field1 item in the target text file, will contain all 3 fragments supplied by the Store
item, when you click the Output tab to preview the result.

**Type**
Allows you to define the data type of the text fragments.

**Trim side**
Defines the side from which the characters will be trimmed: left, right or both. Selecting left, right, or both activates the "Trim character set" option.

**Trim character set**
Defines the characters you want to trim from this text fragment. You can enter any number of characters here, by double clicking in the field. The characters you enter are removed from the Trim side(s) of the fragment.

### 7.4.6 FlexText and Regular Expressions

In MapForce FlexText, you can use regular expressions as follows:

1. To split text containers whenever a match is found (the matched text acts as separator).
   In this case, regular expressions are implicitly anchored; therefore, the caret (^) and the dollar sign ($) characters are not used. For example, to match any three consecutive digits, use \[0-9\]{3} instead of ^\[0-9\]{3}$.
2. To redirect text from a Switch container if the text contains a regular expression match.

You can use regular expressions in FlexText components in any of the following MapForce target languages:

- Built-in (when previewing the mapping)
- Built-in (when running the MapForce Server execution file)
- Code generation languages (C++, C#, Java). Note that, in these languages, some advanced features of regular expressions may depend on the regular expressions implementation in that specific language.

The regular expression syntax and semantics in FlexText is based on [http://www.w3.org/TR/xmleschema-2/#regexs](http://www.w3.org/TR/xmleschema-2/#regexs), similar to the MapForce core function `tokenize-regexp`. Note the following:

- If the split condition matches two sequences following each other, FlexText creates an empty result in between (this behavior is the same when you are not using regular expressions).
- No regular expression flags ([http://www.w3.org/TR/xquery-operators/#flags](http://www.w3.org/TR/xquery-operators/#flags)) are supported.

This section includes the following topics:

- **Splitting Text with Regular Expressions**
- **Using Regular Expressions in Switch Conditions**
7.4.6.1 Splitting Text with Regular Expressions

When you need to split text into two or more fragments, you can optionally use a regular expression as separator. This is an alternative, more advanced option as compared to separating text by means of single or multiple consecutive characters.

The option to split text by means of regular expressions becomes available in FlexText when the following conditions are true:

- The container is of type Split Once or Repeated Split.
- The Mode option is set to delimited (floating), delimited (line based), or delimited (line starts with).

Sample FlexText template which uses regular expressions to separate text

The options applicable to regular expressions are as follows.

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>Separator pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>To use a regular expression as text separator, switch this option to yes. To use simple text as separator, switch this option to no (this is the default value).</td>
<td>If the Regular expression option is switched to yes, a Separator Pattern text box becomes available where you can enter the regular expression that must act as separator. In the FlexText template illustrated above, the regular expression [A-Z]{1}[0-9]{5} matches exactly one</td>
</tr>
</tbody>
</table>
7.4.6.2 Using Regular Expressions in Switch Conditions

When working with Switch containers (see also Switch), you can optionally create a condition within the Switch container to redirect the text fragment if it contains a regular expression match. In such cases, you can use the the caret (^) and the dollar sign ($) characters to match the beginning or end of the text to be searched, except when C++ is set as target transformation language. (In C++, the caret and dollar sign characters are interpreted as the beginning or end of a line, not as the beginning or end of the whole text).

To use a regular expression in a switch condition:

1. Define the container type as Switch (click on the top-right corner of the container, and then click Switch).
2. Click the Append Condition (条件) button to add a new condition.
3. Set the condition type to Content contains regular expression match and enter the regular expression in the adjacent text box.

Example

Let's assume that you need to map data from the database log file represented below (you can also find this file at the following path: <Documents>\Altova\MapForce2019
Your goals are as follows:

1. Collect in a list all rows where a return value is present. That is, the list must include every row which contains the value "Return value 1" or the value "Return value 0", or any other return value expressed as a digit.

2. Collect in another list all rows where the text contains the value "Action start".

To achieve these goals, you can use a Repeated Split container to split down the text into individual rows. After that, you can use a Switch container to redirect each row as required. Namely, the Switch container will consist of three conditions, as follows:

1. Redirect the current row to output A if it contains the value "Action started". You can find such rows by using a condition of type "Content starts with", and enter "Action started" as value.

2. Redirect the current row to output B if it contains a return value. You can find such rows by using the regular expression `Return value [0-9]\.`. This regular expression will return a match if the row contains the text "Return value", followed by a single digit, followed by a full stop. The backslash (\) before the full stop acts as an escape character, to denote that the full stop must be treated as normal character, not as a metacharacter.

3. Redirect the current row to output C (<default>) if the row satisfies neither of the conditions above.

To create the FlexText template which performs the tasks above:

1. On the Insert menu, click Text File.
2. Click **Design graphically new structure based on a sample file** and save the FlexText .mft template to a directory of your choice.

3. When prompted to open a text file, browse for the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\SampleDatabaseLog.txt` file.

4. Click the top-right corner of the output container and select **Repeated Split**. Because we are using carriage return as split character, choose **delimited (floating)** mode, and `<CR><LF>` as separator. This creates a new output container which consists of 17 blocks (one for each row).

5. Click the top-right corner of the new output container and select **Switch**. Now FlexText will treat the contents of the container as a switch. As shown below, one `<default>` switch condition was created automatically—this condition redirects to a new container any text that does not match other conditions. At this stage, there are no other conditions defined, therefore all text is currently being redirected to the `<default>` output.
6. Click the **Append Condition** button and add the condition of type **Content starts with** with the value "Action start", as shown below. This condition redirects to a new container any text that begins with "Action start".

7. Click the **Append Condition** button and add the condition of type **Content contains regular expression match** with the value `Return value [0-9]`. If you now navigate to block 5 out of 17, you can see that this condition redirects that block to a new container, since the block contains a match for the regular expression.

You have now configured the Switch container so that it redirects text to a different output based on a condition. There are three switch conditions and three possible outputs (one output for each condition). The remaining steps of this tutorial show how you can write each of output to a separate text file.

8. Click the top-right corner of each output container and choose **Store as Value**. Additionally, double-click the title bar of each output and enter a descriptive name: **Output A**, **Output B**, and **Output C**, respectively.

9. Save and close the FlexText template.
The required FlexText template has now been created. You can see how this template works by opening the following tutorial file:

\<Documents\>\Altova\MapForce2019\MapForceExamples\Tutorial\FlexTextSwitchByRegEx.mfd

The FlexTextSwitchByRegEx.mfd mapping illustrated in the screen shot reads text data from the \<Documents\>\Altova\MapForce2019\MapForceExamples\Tutorial\SampleDatabaseLog.txt file and converts it into three separate text files: Output A, Output B, and Output C. Each of the target text files contains only the rows that satisfy one of the three conditions defined previously in the FlexText template. To view the output of a particular target component, click the Preview Component ( ) button in the top-right corner of the component, and then click the Output tab.
EDI

Altova website: EDI mapping

EDI (Electronic Data Interchange) is a family of standards which enable electronic exchange of data between organizations or businesses.

MapForce supports translating data to or from EDI formats in any of the following transformation languages: BUILT-IN, C#, C++, Java (see also Selecting a Transformation Language). The EDI data can be mapped to or from any of the formats supported by MapForce, including databases, flat files, XML documents, and others.

MapForce supports the following flavours of EDI:

<table>
<thead>
<tr>
<th>Standard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASC X12</td>
<td>ASC X12 is an industry standard for document interchange. MapForce supports versions: 3040, 3050, 3060, 3070, 4010, 4020, 4030, 4040, 4041, 4042, 4050, 4051, 4052, 4060, 5010, 5011, 5012, 5020, 5030, 5040, 5050, 6010, 6020, 6030, and 6040. The default ASC X12 version in MapForce is 6040. ASC X12 components have &quot;virtual&quot; nodes into which EDI parser error information/data is written depending on the settings you select in the EDI Validation Settings dialog box (see EDI component validation). An X12 997 Functional Acknowledgement can be generated from any X12 document. For more information about ASC X12, see <a href="http://www.x12.org/">http://www.x12.org/</a>.</td>
</tr>
<tr>
<td>HIPAA X12</td>
<td>HIPAA is based on the X12 EDI 5010 standard, but has its own specialized versions which are natively supported by MapForce 2011 Release 3 or later. The default HIPAA X12 version in MapForce is release A2 of the HIPAA implementation specs (TR3). Previous releases are available for download on the MapForce Components page of the Altova website (<a href="https://www.altova.com/mapforce/download/libraries">https://www.altova.com/mapforce/download/libraries</a>).</td>
</tr>
<tr>
<td>HL7</td>
<td>HL7 is an industry standard for data exchange between medical applications and is an abbreviation of &quot;Health Level Seven&quot;. MapForce supports versions 2.2 to 2.6. The default HL7 version in MapForce is 2.6. A separate installer for the additional HL7 V2.2 - V2.5.1 XML Schemas and configuration files is available on the MapForce Components page of the Altova website (<a href="https://www.altova.com/mapforce/download/libraries">https://www.altova.com/mapforce/download/libraries</a>). The XML-based HL7 version 3.x is supported in MapForce 2019 using XML</td>
</tr>
<tr>
<td>Standard</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>schema components.</td>
</tr>
<tr>
<td></td>
<td>The <strong>MapForceExamples</strong> project contains a sample that maps a HL7 V2.6 to a HL7 V3 XML file (HL7V260_to_HL7V3.mfd).</td>
</tr>
<tr>
<td></td>
<td>For more information about HL7, see <a href="https://www.hl7.org/">https://www.hl7.org/</a>.</td>
</tr>
<tr>
<td>IATA PADIS</td>
<td>PADIS (Passenger and Airport Data Interchange Standards) is an industry standard for the exchange of passenger and airport data using EDI documents. MapForce supports versions: 00.1 to 08.1. The <strong>default PADIS version in MapForce is 08.1.</strong></td>
</tr>
<tr>
<td></td>
<td>For more information about IATA PADIS, see <a href="http://www.iata.org/Pages/default.aspx">http://www.iata.org/Pages/default.aspx</a>.</td>
</tr>
<tr>
<td>NCPDP SCRIPT</td>
<td>SCRIPT is a standard for exchanging electronic information in the US healthcare sector developed by the NCPDP (National Council for Prescription Drug Programs), see <a href="http://www.ncpdp.org">http://www.ncpdp.org</a>. MapForce supports mapping data between files in SCRIPT format and any other format supported by MapForce, including XML, CSV, Microsoft Excel, databases, and others. The supported NCPDP SCRIPT versions are: 8.1, 10.6, 10.10. The <strong>default NCPDP SCRIPT version in MapForce is 10.10.</strong></td>
</tr>
<tr>
<td></td>
<td>For more information about NCPDP SCRIPT, see <a href="http://www.ncpdp.org">http://www.ncpdp.org</a>.</td>
</tr>
<tr>
<td>SAP IDocs</td>
<td>SAP IDocs (intermediate documents) documents are used to exchange business data between SAP R/3 and non-SAP applications. The documents are a form of intermediate data storage which can be exchanged between different systems. For more information about SAP IDocs, see <a href="https://help.sap.com/saphelp_nw70/helpdata/en/0b/2a6095507d11d18ee90000e8366ic2/frameset.htm">https://help.sap.com/saphelp_nw70/helpdata/en/0b/2a6095507d11d18ee90000e8366ic2/frameset.htm</a>.</td>
</tr>
<tr>
<td>TRADACOMS</td>
<td>TRADACOMS (Trading Data Communications) is a UK-specific Electronic Data Interchange standard used in the retail business. MapForce implements the base TRADACOMS specification as laid out in the &quot;TRADACOMS Manual of Standards for Electronic Data Interchange&quot;, published in January 1993 by the Article Numbering Association (ANA) UK, now known as GS1 UK ( <a href="https://www.gs1uk.org">https://www.gs1uk.org</a> ). For other TRADACOMS versions, MapForce can be customized to process new message types, data elements, and code values, by means of configuration files.</td>
</tr>
<tr>
<td>UN/EDIFACT</td>
<td>UN/EDIFACT (the United Nations rules for Electronic Data Interchange for Administration, Commerce and Transport) is a set of standards for electronic document interchange, see <a href="https://www.unice.org/cefact/edifact/welcome.html">https://www.unice.org/cefact/edifact/welcome.html</a>. MapForce supports the messages contained in directories D93.A - D.18B. The <strong>default UN/EDIFACT version in MapForce is D.18B.</strong></td>
</tr>
</tbody>
</table>
A separate installer for all previous versions back to D93.A, including the configuration files required for Interactive EDI mappings, is available on the Altova MapForce Libraries page (https://www.altova.com/components_mapforce.html).

The default version available in MapForce for each EDI flavour is highlighted in bold in the table above. If necessary, you can install previous versions of EDI configuration files from the MapForce Components page of the Altova website (https://www.altova.com/mapforce/download/libraries). After installation, the installed EDI versions become available in MapForce in addition to the default versions.

For a single EDI component that you have added to the mapping area, MapForce can process multiple message types of any single standard EDI release. For example, this enables you to read data from an interchange file which includes more than one message type, or to generate as mapping output an interchange file which contains multiple message types. You can also process data from multiple instance files containing different message types. Note that operations such as the ones described above are only meaningful when they are supported by the underlying specification.

If your organization is using EDI specifications that are customizations of those already supported by MapForce, it is possible to adapt MapForce to your custom EDI specification by means of configuration files. For example, you can add custom messages, segments, data elements, code lists, and so on (or change the existing ones), thus creating a custom EDI collection similar to any of the EDI-related directories that are already supported by MapForce. For more information, see Customizing EDI Structure and Customizing EDI Validation.

### 7.5.1 EDI Terminology

The following list describes some of the common terms used in conjunction with EDI-related standards.

#### Message

A message represents a business document. In ASC X12, messages are called "transaction sets". A message is composed of multiple building blocks known as "segments". MapForce supports one or multiple different message types in a single EDI component (see Adding EDI Messages as Mapping Components).

#### Segment

A segment represents a single "record" in a message. Segments are identified by a two or three character ID at the beginning of the segment. A group of related elements comprise a segment tag (or segment ID, in ASC X12). Segments of a transaction can be defined as mandatory or conditional (optional).
Data Element
A data element is an individual data field within a segment. A data element can be thought of as a field, because it usually contains one type of data (for example, a name or an address). Data elements can be simple or composite elements. In the latter case, they consist of component elements or subelements.

Separator
Data elements are delimited by separator characters, referred to as "separators". Separators are usually ASCII characters which are less likely to occur in text fields (for example, colons, asterisks, pipes, bars, ampersands, etc). The separators vary by EDI flavour. By default, MapForce recognizes the standard separators applicable for each EDI flavour. However, separators can be customized if necessary (see EDI Component Settings).

7.5.2 Adding EDI Components to the Mapping
To map data to or from EDI files, the EDI component must be added to the mapping first. The EDI component represents the structure of the EDI message (or messages) on the mapping. Once the EDI component is added to the mapping, you can draw mapping connections to or from it, similar to any other component types supported by MapForce (see Working with Components). When necessary, you may also configure the EDI component to include multiple messages. This would enable you to read data from multiple EDI files (one message per file), as shown in Processing Multiple Input or Output Files Dynamically, or process different message types within the same file.

To add the EDI component to the mapping:

1. On the Insert menu, click EDI.
2. Click the EDI collection, and select the check box next to the message type that you want to include. You can also include multiple messages, by selecting their corresponding check boxes. For example, in the image below, two messages of the HL7 collection are being added to the mapping.
3. Click OK. You are now prompted to supply a sample EDI file.
4. If you want to read data from the EDI component, browse for the sample EDI file; otherwise, click **Skip**.

The component now appears on the mapping, and it includes the structure of both messages selected previously. If you want to add additional messages, or remove existing ones, click the **+** button that appears next to the first message.

Sample HL7 mapping component which includes two messages

You can now draw mapping connections from this EDI component to a target one, or vice versa. Several MapForce mapping design (.mfd) files that illustrate mapping data to or from various EDI flavours are available in the `<Documents>\Altova\MapForce2019\MapForceExamples` folder:
### Running EDI Mappings

Mappings which contain EDI components are supported when the mapping is set to one of the following transformation language: BUILT-IN, C++, C#, Java. Once an EDI mapping has been designed in MapForce, it can be run in the following ways:

- The result of the data transformation can be previewed directly in MapForce, by clicking the **Output** tab. The output generated by the mapping will be saved to the desired location with the desired name, depending on how you configured both MapForce and the target component.
- The mapping can be run on a server machine (Windows, Linux, macOS) with MapForce Server, through an API call or at the command-line interface. This requires that the mapping be compiled as .mfx file and copied manually to the server machine, see [Compiling Mappings to MapForce Server Execution Files](#).
- The mapping can be deployed to a server machine (Windows, Linux, macOS) through HTTP (or HTTPS) and run there as a scheduled or on demand job, or as a Web service, see [Deploying Mappings to FlowForce Server](#).
- C++, C#, or Java program code can be generated from the mapping. The generated program performs the same result as if the mapping is previewed in MapForce; it can be run standalone or integrated into your existing code, see [Code Generator](#).

To configure only the settings applicable to an EDI component (either source or target), right-click the component header, and select **Properties** from the context menu, see [EDI Component Settings](#). For settings applicable to other component types, see [Changing the Component Settings](#).

To access the code generation settings, go to **Tools | Options**, and then click the **Generation** tab.
7.5.4  EDI Component Settings

After you add an EDI component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component settings dialog box in one of the following ways:

- On the Component menu, click **Properties** (this command is enabled provided that a component is selected on the mapping).
- Double-click the component header.
- Right-click the component header, and then click **Properties**.
The available settings are listed below. Note that some settings are not available if the EDI flavour of the currently selected component does not support them.

<table>
<thead>
<tr>
<th><strong>Component name</strong></th>
<th>The component name is automatically generated when you create the component. However, you can change the name at any time.</th>
</tr>
</thead>
</table>
The component name can contain spaces and full stop characters. It may not contain slashes, backslashes, colons, double quotes, leading or trailing spaces. If you want to change the name of the component, be aware of the following:

- If you intend to deploy the mapping to FlowForce Server, the component name must be unique.
- It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line.

**Input EDI file**
Specifies the EDI source file from which MapForce will read data. This field is meaningful for a source component and is filled when you first create the component and assign to it an EDI instance file.

**Output EDI file**
Specifies the EDI target file to which MapForce will write data. This field is meaningful for a target component.

**Input/Output Encoding**
Allows you specify the following settings of the output instance file:

- Encoding name
- Byte order
- Whether the byte order mark (BOM) character should be included.

By default, any new components have the encoding defined in the Default encoding for new components option. You can access this option from Tools | Options, General tab.

**EDI Settings**
This group of settings enable you to define custom EDI delimiters, separators and terminators (note the settings are available only if supported by the EDI format).

The EDI separators entered in this dialog box always take effect when writing EDI files. When reading in EDI files, the separators only take effect if the input file does not define/contain its own separators (for example, EDIFACT files without the UNA “service string advice” segment).

If an EDI input component/file contains separator definitions (for example, an X12 file with an ISA segment), then the existing separators override any separators defined in the Component Settings dialog box for that file.

You can use non-printable characters as separators by typing "x" followed by the hexadecimal ASCII character code into one of the combo boxes, for example, "x1e" for the RS control character (ASCII record separator, decimal code 30).
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The Subcomponent Separator</strong></td>
<td>Applies to the HL7 standard. The default value of this separator is ampersand (&amp;).</td>
</tr>
</tbody>
</table>
| **Auto-complete missing fields** | This option applies to target EDI components. When this check box is selected, MapForce fills in the values of some data fields automatically. This applies only to those fields where this operation would not contradict the specification of the currently selected EDI format. To disable such behaviour, clear this check box. See also:  
  - ASC X12 Automatic Data Completion Rules  
  - UN/EDIFACT Automatic Data Completion Rules |
| **Begin new line after each segment** | This option applies to target EDI components. When this check box is selected, MapForce adds a CR/LF (carriage return / line feed) character after each EDI segment. The EDI standard ignores these lines if present in a message. |
| **Extended** | Opens a dialog box where you can define additional settings for the current EDI component. The available settings vary by EDI flavour. |
| **Validation** | Opens a dialog box where you can define the validation settings for the current EDI component (see EDI Component Validation). |
| **Enable input processing optimizations based on min/ maxOccurs** | This option allows special handling for sequences that are known to contain exactly one item, such as required attributes or child elements with minOccurs and maxOccurs=“1". In this case the first item of the sequence is extracted, then the item is directly processed as an atomic value (and not as a sequence). If the input data is not valid against the schema, an empty sequence might be encountered in a mapping, which stops the mapping with an error message. To allow the processing of such invalid input, clear this check box. |
| **Save all file paths relative to MFD file** | When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. This setting affects the following files:  
  - The input EDI file (if present)  
  - The output EDI file (if present) |

See also Using Relative Paths on a Component.
7.5.5 EDI Component Validation

MapForce validates all supported EDI source and target documents when the mapping is executed. This happens either when you preview the mapping in MapForce, by clicking the **Output** tab, or when the mapping is run by MapForce Server, or by a C++, C#, or Java program. You can also validate explicitly the EDI output generated by the mapping. To do this, first click the **Output** tab, and then click the **Validate Output** toolbar button (see also Validating the Mapping Output).

MapForce performs validation of EDI documents as specified in this documentation for each individual supported format. The validation process is, however, not a full EDI syntax or semantic validation. For example, there may exist instances where the possible values of a field rely on standards external to EDI and cannot be easily expressed as a finite set of values that would foresee all processing possibilities. For such cases, it is possible to customize the validation rules on a case-by-case basis with the help of configuration files (see Customizing EDI Validation).

At a more simple level, some validation options can be configured from the MapForce graphical user interface. For example, you can choose to ignore specific errors, or stop the mapping execution when specific errors are encountered.

To customize the validation settings, double-click an EDI component and click the **Validation** button in the Component Settings dialog box (see also EDI Component Settings). The default validation settings are shown below.
For each validation event listed above, you can take any of the following actions: **Stop**, **Report & Reject**, **Report & Accept**, **Ignore**.

**Stop** is used to catch fatal errors and stops the execution of the mapping.

**Report & Reject** provides information about the error and rejects the current message. **Report & Accept** provides information about the error and accepts the current message. The actual error information is visible in the Messages window of MapForce. For source X12 and HIPAA components, the validation errors are also logged inside the mapping component, which enables you to generate X12 997 or X12 999 reports (see Validation Results and X12 Acknowledgement).

**Ignore** ignores the validation event. If you select this action, no information is provided about the error, so use it cautiously.

If unexpected segments are encountered while processing an EDI mapping, only MapForce and MapForce Server have the capability to recover and attempt to continue parsing. The generated C++, C#, or Java code cannot recover from unexpected segments and will stop processing. Unexpected segments include the following: incorrect segment order, missing mandatory segment, incorrect segment name, additional segments that are not present in the configuration files.

The EDI validation events for which you can configure custom actions (**Stop**, **Ignore**, etc) are as
follows.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
<th>Error might occur for a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Missing segment</td>
<td>A mandatory segment is missing or the occurrence is less than a specified minimum.</td>
<td>source component</td>
</tr>
<tr>
<td>Unexpected segment</td>
<td>A segment is defined in the specification but not in this message.</td>
<td>source component</td>
</tr>
<tr>
<td>Unrecognized segment ID</td>
<td>A segment was found which is not defined in the specification.</td>
<td>source component</td>
</tr>
<tr>
<td>Missing group</td>
<td>A mandatory group is missing or the occurrence is less than the specified minimum.</td>
<td>source component</td>
</tr>
<tr>
<td>Unexpected end of file</td>
<td>The instance cannot be parsed since some data is missing.</td>
<td>source component</td>
</tr>
<tr>
<td>Missing field or composite</td>
<td>A mandatory field or composite is missing, or the occurrence is less than the specified minimum.</td>
<td>source or target component</td>
</tr>
<tr>
<td>Extra data in segment or composite</td>
<td>The input instance contains additional data that is not expected by the syntax description.</td>
<td>source component</td>
</tr>
<tr>
<td>Extra repeat</td>
<td>The actual number of fields within a segment/composite exceeds the specified maximum number.</td>
<td>source or target component</td>
</tr>
<tr>
<td>Invalid field value</td>
<td>A numeric field contains an invalid character.</td>
<td>source component</td>
</tr>
<tr>
<td>Invalid date</td>
<td>A date field contains either an invalid character or the values for the month or the day are invalid.</td>
<td>source component</td>
</tr>
<tr>
<td>Invalid time</td>
<td>A time field contains either an invalid character or the value for the hours or the minutes are invalid.</td>
<td>source component</td>
</tr>
<tr>
<td>Numeric overflow</td>
<td>A numeric values overflows its defined domain. This error is only supported within the generated code.</td>
<td>source component</td>
</tr>
<tr>
<td>Data element too short</td>
<td>The length of a data element is less than the specified minimum value.</td>
<td>source or target component</td>
</tr>
<tr>
<td>Data element too long</td>
<td>The length of a data element is greater than the specified maximum limit.</td>
<td>source or target component</td>
</tr>
<tr>
<td>Invalid code list value</td>
<td>The value of the data element does not match any of the preconfigured code</td>
<td>source or target component</td>
</tr>
</tbody>
</table>
Data Sources and Targets

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
<th>Error might occur for a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantic error</td>
<td>A semantic error has occurred.</td>
<td>source or target component</td>
</tr>
<tr>
<td>Implementation &quot;Not Used&quot; data element present</td>
<td>An element exists in the input file which is not allowed by the HIPAA configuration file (maxOccurs=&quot;0&quot;).</td>
<td>source component</td>
</tr>
<tr>
<td>Input file was not completely parsed</td>
<td>The input file was not completely parsed.</td>
<td>source component</td>
</tr>
</tbody>
</table>

Validation results and X12 Acknowledgment

When MapForce validates an X12 or HIPAA component, the validation results are placed in "virtual" items at the base of the EDI component (under "ParserErrors_Message" and "ParserErrors_Group", as shown in the image below). This information can be used to generate X12 997 or X12 999 files that report the status of the interchange (see Generating an X12 997 Functional Acknowledgment and Generating an X12 999 Implementation Acknowledgment).

With X12 or HIPAA.X12 components, the validation actions available on the EDI Validation Settings dialog box (discussed above) have the following meaning:

- **Stop** will stop the execution of the mapping without generating a report message.
- **Report & Reject** and **Report & Accept** will provide information in the "ParserErrors_Message".
and "ParserErrors_Group" items of the EDI component which can be mapped further.

Depending on the setting **Reject** or **Accept**, the "Functional Group Acknowledge Code F715" and the "Transaction Set Acknowledgment Code F717" will contain either:
  - the value 'R' for 'Rejected' or
  - the value 'E' for 'Accepted, but errors were noted'. The errors also appear in the Messages window.

**Ignore** ignores the specific error. No information is provided within the "ParserErrors_Message" and "ParserErrors_Group" items.

### 7.5.6 ASC X12

ASC X12 is an industry standard for document interchange. MapForce supports versions: 3040, 3050, 3060, 3070, 4010, 4020, 4030, 4040, 4041, 4042, 4050, 4051, 4052, 4060, 5010, 5011, 5012, 5020, 5030, 5040, 5050, 6010, 6020, 6030, and 6040.

The default ASC X12 version in MapForce is 6040.

ASC X12 components have "virtual" nodes into which EDI parser error information/data is written depending on the settings you select in the EDI Validation Settings dialog box (see **EDI component validation**). An X12 997 Functional Acknowledgement can be generated from any X12 document.

For more information about ASC X12, see [http://www.x12.org/](http://www.x12.org/).

### 7.5.6.1 ASC X12 Validation Rules

When validating ASC X12 documents, MapForce performs the following checks:

- Whether an ISA and an IEA segment exist
- Whether ISA/I01 contains a legal authorization information qualifier.
- Whether ISA/I03 contains a legal security information qualifier.
- Whether the two ISA/I05 segments contain legal interchange ID qualifiers.
- Whether ISA/I08 contains a well-formed date value.
- Whether ISA/I09 contains a well-formed time value.
- Whether ISA/I13 contains a legal Boolean value.
- Whether ISA/I14 contains a legal interchange usage indicator.
- Whether ISA/I12 and IEA/I12 contain the same value.
- Whether IEA/I16 contains the correct number of function groups in the interchange.

Each function group is checked:

- If there is a matching GS and GE pair.
- Whether GS/373 contains a well-formed date value.
- Whether GS/337 contains a well-formed time value.
- Whether GS/28 and GE/28 contain the same value.
- Whether GE/97 contains the correct number of messages in the function group.

Each message is checked:
Data Sources and Targets

• If there is a matching ST and SE pair.
• Whether ST/143 contains the correct message identifier.
• Whether ST/329 and SE/329 contain the same value.
• Whether SE/96 contains the correct number of segments in the message.

7.5.6.2 ASC X12 Automatic Data Completion Rules

When generating an ASC X12 file, MapForce may automatically create certain fields or values, according to the rules below (subsequently referred to as "automatic data completion", or "auto-completion" rules). To disable this behaviour, clear the Auto-complete missing fields check box in EDI Component Settings.

Automatic data completion for EDI/X12 makes sure:

• That an ISA and IEA pair exist on the interchange level.
• That if either GS or GE exist, the other ID also exists.
• That there is at least one ST/SE pair on the message level.
• That ISA/I01 and ISA/I03 exist. If they do not contain data, 00 is inserted.
• That ISA/I02 and ISA/I04 exist. If they do not contain data, ten blanks are inserted.
• That both ISA/I05 segments exist. If they do not contain data, ZZ is inserted.
• That ISA/I08 exists. If it does not contain data, the current date in EDI format is inserted.
• That ISA/I09 exists. If it does not contain data, the current time in EDI format is inserted.
• That ISA/I65 exists. If it does not contain data, the repetition separator is inserted.
• That ISA/I11 exists. If it does not contain data, the interchange control version number from the user-defined settings is inserted (see the setting Extended | Interchange control version-number of the EDI Component Settings).
• That ISA/I12 exists.
• That ISA/I13 exists. If it does not contain data, the request acknowledgment setting is used (see the setting Extended | Request Acknowledgement of the EDI Component Settings).
• That ISA/I14 exists. If it does not contain data, P is inserted.
• That ISA/I15 exists. If it does not contain data, the composite separator from the user-defined settings is inserted (see the setting Composite Separator of the EDI Component Settings).
• That IEA/I16 exists. If it does not contain data, the number of function groups in the interchange is calculated and inserted.
• That IEA/I12 exists. If it does not contain data, the value from ISA/I12 is copied.

Note: Any fields not mentioned here are not inserted or created. The correct values cannot be ascertained automatically. Given a (target) parent element A (in the target EDI component) with child items x, y, and z - where y is mandatory, parent element A will only be created in the output file if the mandatory child element "y" in the target component has been mapped.

The potentially existing function group is checked as follows:

• That GS/373 exists. If it does not contain data, the current date in EDI format is inserted.
• That GS/337 exists. If it does not contain data, the current time in EDI format is inserted.
• That GE/97 exists. If it does not contain data, the number of messages in the function group are calculated and inserted.
• That GE/28 exists. If it does not contain data, the value from GS/28 is copied.
Message checking makes sure:

- That ST/143 exists. If it does not contain data, the name of the message is inserted.
- That SE/96 exists. If it does not contain data, the number of segments in the message is calculated and inserted.
- That ST/329 and SE/329 exist. If SE/329 does not contain data, the value from ST/329 is copied.

7.5.6.3 Generating an X12 997 Functional Acknowledgment

The X12 997 (Functional Acknowledgment) reports the status of the EDI interchange. All errors encountered during processing of the document are reported in it. MapForce can automatically generate an X12 997 component in the main mapping area, and automatically create the necessary connectors.

To generate the EDI 997 Functional Acknowledgment:

1. On the Insert menu, click EDI, and add the X12 component to the mapping. When prompted to select as instance file, browse for the file for which you want to create the X12 997 acknowledgement.
2. Right-click the source EDI component and select Create Mapping to EDI X12 997 (note this command is enabled only for X12 components).

This creates an EDI 997 mapping component and automatically connects the items needed to generate the X12 997 acknowledgment. For an example, see the X12_To_XML_Order.mfd mapping design file available in the <Documents>\Altova\MapForce2019\MapForceExamples\ folder. To see the acknowledgment file generated by this mapping, click the Preview ( ) button of the EDI 997 component, and then click the Output tab.
To save the X12 997 Acknowledgement file:

- Click the Output tab, and then click the Save generated output toolbar button. If you intend to generate program code from the mapping, enter a file name in the "Output EDI File" field of the Component Settings dialog box (see EDI Component Settings).

7.5.6.4 Generating an X12 999 Implementation Acknowledgment

The X12 999 (Implementation Acknowledgment) reports X12 non-compliance or application errors. Because it is a super-set of the 997 Functional Acknowledgement, 999 can be used instead of 997 to accept or reject transaction sets based on either X12 or HIPAA Implementation Guide syntax requirements.

All errors encountered during processing of the document are reported in it (for example, "Required Segment Missing", "Required Data Element Missing", "Code Value Not Used in Implementation", etc). MapForce can automatically generate the X12 999 component on the mapping, and automatically create the necessary mapping connections.

To generate the X12 999 Implementation Acknowledgment:

1. On the Insert menu, click EDI, and add the X12 or HIPAA component to the mapping. When prompted to select as instance file, browse for the file for which you want to create the X12 999 acknowledgement.

2. Right-click the source EDI component and select Create mapping to EDI X12 999 (note this command is enabled only for X12 and HIPAA components).
This creates an EDI 999 mapping component and automatically connects the items needed to generate the X12 999 implementation acknowledgment. To preview the generated acknowledgment file, click the **Output** tab. If the mapping has multiple target components, first click the **Preview** button, and then click the **Output** tab.

To save the 999 Acknowledgement file:

- Click the **Output** tab, and then click the **Save generated output** toolbar button. If you intend to generate program code from the mapping, enter a file name in the "Output EDI File" field of the Component Settings dialog box (see [EDI Component Settings](#)).

### 7.5.7 HIPAA X12

HIPAA X12 is the latest version of the standard for electronic health care records established by the US Department of Health and Human Services for electronic medical data transactions between insurers, providers, and employers, based on EDI X12 version 5010.

MapForce supports the latest release, A2, of the HIPAA implementation specs (TR3). Older releases are downloadable as separate ZIP file from [Altova website](#).
HIPAA components are similar to ordinary ANSI X12 components, and MapForce supports the following transactions:

<table>
<thead>
<tr>
<th>X12 name</th>
<th>Message name</th>
</tr>
</thead>
<tbody>
<tr>
<td>X279A1</td>
<td>&quot;Health Care Eligibility Benefit Inquiry (270)&quot;</td>
</tr>
<tr>
<td>X279A1</td>
<td>&quot;Health Care Eligibility Benefit Response (271)&quot;</td>
</tr>
<tr>
<td>X212</td>
<td>&quot;Health Care Claim Status Request (276)&quot;</td>
</tr>
<tr>
<td>X212</td>
<td>&quot;Health Care Information Status Notification (277)&quot;</td>
</tr>
<tr>
<td>X214</td>
<td>&quot;Health Care Claim Acknowledgment (277)&quot;</td>
</tr>
<tr>
<td>X217</td>
<td>&quot;Health Care Services Review - Request for Review (278)&quot;</td>
</tr>
<tr>
<td>X217</td>
<td>&quot;Health Care Services Review - Response (278)&quot;</td>
</tr>
<tr>
<td>X218</td>
<td>&quot;Payroll Deducted and Other Group Premium Payment for Insurance Products (820)&quot;</td>
</tr>
<tr>
<td>X220A1</td>
<td>&quot;Benefit Enrollment and Maintenance (834)&quot;</td>
</tr>
<tr>
<td>X221A1</td>
<td>&quot;Health Care Claim Payment/Advice (835)&quot;</td>
</tr>
<tr>
<td>X222A1</td>
<td>&quot;Health Care Claim: Professional (837)&quot;</td>
</tr>
<tr>
<td>X224A2</td>
<td>&quot;Health Care Claim: Dental (837)&quot;</td>
</tr>
<tr>
<td>X223A2</td>
<td>&quot;Health Care Claim: Institutional (837)&quot;</td>
</tr>
<tr>
<td>X231A1</td>
<td>&quot;Implementation Acknowledgment For Health Care Insurance (999)&quot;</td>
</tr>
</tbody>
</table>

Differences to standard X12 message handling is that MapForce:

- automatically maintains the hierarchy of HL segments
- supports so called floating structures (in 837 messages)
- auto-completes and validates more fields.

MapForce also supports the automatic generation of HIPAA X12 999 Implementation Acknowledgment, which is similar to 999 Implementation Acknowledgement of the X12 standard (see Generating an X12 999 Implementation Acknowledgment).

Multiple messages per interchange (i.e. EDI component) is also supported for HIPAA components.

### 7.5.7.1 Example: Mapping HIPAA X12 to XML

This example illustrates how to map a file which contains an "837-Q2 Health Care Claim: Dental" transaction to an XML target file. It is accompanied by a mapping available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\HIPAA_837D.mfd`.
The mapping can be created as shown in the steps below.

**Step 1: Add the source HIPAA X12 file to the mapping**
1. On the **Insert** menu, click **EDI**.
2. Select the collection "HIPAA.X12", and the message type "837-Q2 Health Care Claim: Dental".
3. When prompted to supply an instance file, open the source file available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\837-Q2.x12`.
4. When prompted, adjust the settings of the newly added HIPAA X12 component if necessary (for more information, see the [EDI Component Settings](#)).

**Step 2: Add the target schema to the mapping**
1. On the **Insert** menu, click the **XML Schema/File** and open the **837-Q2.xsd** file from the `<Documents>\Altova\MapForce2019\MapForceExamples` directory.
2. When prompted to supply a sample XML file, click **Skip** and select **DentalClaim** as the root of the target document.

**Step 3: Add a function to format the date and draw the mapping connections**
In the source HIPAA file, the DMG segment includes a date, in the format **19490501**. To convert this date to a format suitable for the target XML file **1949-05-01**, we will add a date conversion function, as follows:
1. Drag the `parse-date` function from the Libraries window into the mapping. For more information about functions, see Working with Functions.

2. On the Insert menu, click Constant, and enter `[Y, 4-4] [M, 2-2] [D, 2-2]` as constant value.

3. Draw the following connections:

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>format input item of the <code>parse-date</code> function</td>
<td>Supplies to the function the format according to which the date should be parsed.</td>
</tr>
<tr>
<td>F1251</td>
<td>value input item of the <code>parse-date</code> function</td>
<td>Supplies to the function the actual value to be formatted.</td>
</tr>
<tr>
<td>result</td>
<td>DateOfBirth input item of the XML target</td>
<td>Supplies the value returned by the function to the DateOfBirth XML node.</td>
</tr>
</tbody>
</table>

4. Draw the remaining mapping connections as shown in the example mapping (these are too many to be listed here; essentially, each connection copies values from the source HIPAA file to the target XML file. For basic information about mapping connections, see Working with Connections.)

### 7.5.7.2 Considerations when Mapping HIPAA Transactions

**HL segment hierarchy**

HL segments are generated automatically and do need to be mapped manually. It is however possible to map values if there is such a need.

**Autocompletion**

The configuration files for HIPAA transactions contain many codes and are used for auto-completion of fields whenever possible. Autocompletion is context sensitive, so fields in different places may have different values. To disable auto-completion, clear the Auto-complete missing fields check box in the Component Settings dialog box (see EDI Component Settings).

**Validation**

Validation is similar to auto-completion in that it is also context sensitive. Validation errors are handled in the same way as per X12 validation (see ASC X12 Validation Rules).

**Editing of configurations files**

HIPAA configuration files use a newer schema version than the X12 configuration files (see EDI Configuration Schemas). Therefore, HIPAA configuration files are similar to X12 configuration files, with a few additions:

- Code values at field level
- Conditions
• Completion flags
• "Not Used" Data Elements

Code values
Field instances can have a built-in code list which is used for validation and/or autocompletion. Values can be manually added or removed (see Changing Possible Code Values of Data Elements).

```xml
<Data ref="F365" info="Communication Number Qualifier" nodeName="F365_1">
  <Values>
    <Value Code="EM" />
    <Value Code="FX" />
    <Value Code="TE" />
  </Values>
</Data>
```

Conditions
Conditions work in conjunction with values. As there can be multiple repetitions of a segment, or group (loop), they are identified by a specific condition. A Condition requires a value to be present for it to be fulfilled, otherwise the specific group is not found.

For example, Loop1000 is repeated multiple times, but each instance has a different semantic meaning.

To tell which is which, MapForce uses a condition that specifies that Loop1000A must have a Value Code of "41", in Field F98 of segment NM1. If this is not the case, then Group name Loop1000A is not found.

```xml
<Group name="Loop1000A" info="Submitter Name">
  <Segment name="NM1" info="Submitter Name">
    <Condition path="F98" />
    <Data ref="F98" info="Entity Identifier Code">
      <Values>
        <Value Code="41" />
      </Values>
    </Data>
  </Segment>
</Group>
```

(…)

Condition codes are auto generated in the target component, if they contain single value.

Configuration files are generated by a method that ensures that condition values are guaranteed to be unique across the sequence of repeating segments/loops.

If there is a need to edit conditions, or their values, the uniqueness constraint must be taken into account.

Completion flags
Autocompletion can be adjusted on the configuration file level. A new element "Completion", having three attributes, has been introduced to define this:

• singleConditions – auto-complete single conditions for all fields in the target component
• singleValues – auto-complete single values for all fields in the target component
• HL – generate appropriate fields in the HL segment in the target component (for all HL segments)
(Where "1" means true)

<Message>
<MessageType>837-Q2</MessageType>
<Completion singleConditions="1" singleValues="1" HL="1"/>
<Description>Health Care Claim: Dental</Description>
(…)

"Not Used" Data Elements
HIPAA omits several optional elements that are present in standard X12 transactions. These optional elements are hidden in MapForce components.

The validation engine checks to make sure that these unused fields are not present in source files. Since these fields cannot be mapped to the target component (because they are hidden), they cannot appear in target output files either.

These fields are defined to have the maxOccurs attribute equal to 0, in the configuration files. If necessary, you can manually hide (or unhide) specific fields by using the maxOccurs attribute.

<Data ref="F1037" minOccurs="0" info="Submitter Middle Name or Initial"/>
<Data ref="F1038" minOccurs="0" maxOccurs="0"/>
<Data ref="F1039" minOccurs="0" maxOccurs="0"/>

Scope of validation
Semantic validation is not supported in MapForce. This means that “situational” fields are simply treated as optional.

7.5.8 HL7 Version 2

HL7 is an industry standard for data exchange between medical applications and is an abbreviation of "Health Level Seven". MapForce supports versions 2.2 to 2.6.

The default HL7 version in MapForce is 2.6.

A separate installer for the additional HL7 V2.2 - V2.5.1 XML Schemas and configuration files is available on the MapForce Components page of the Altova website (https://www.altova.com/mapforce/download/libraries).

The XML-based HL7 version 3.x is supported in MapForce 2019 using XML schema components.

The MapForceExamples project contains a sample that maps a HL7 V2.6 to a HL7 V3 XML file (HL7V260_to_HL7V3.mfd).

For more information about HL7, see https://www.hl7.org/.
7.5.8.1 **Example: Mapping HL7 to XML**

This example illustrates how to map a file which contains two HL7 messages ("Query for vaccination record" and "Unsolicited vaccination record update") to two separate XML target files. It is accompanied by a mapping available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\HL7_MultiMessageTypes.mfd.`

The mapping can be created as shown in the steps below.

**Step 1: Add the source HL7 file to the mapping**

1. On the **Insert** menu, click **EDI**.
2. Select the collection "HL7", and then select the check boxes next to the message types "VXQ - Query for vaccination record" and "VXU - Unsolicited vaccination record update".
3. When prompted to supply an instance file, open the source file available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\hl7multi_v02_v04.hl7`.
4. When prompted, adjust the settings of the newly added HL7 component if necessary (for
Step 2: Add the target schema to the mapping

1. On the Insert menu, click the XML Schema/File, and open the VaccinationQueryReport.xsd file from the <Documents>\Altova\MapForce2019\MapForceExamples directory.
2. When prompted to supply a sample XML file, click Skip.

Follow the same steps to add the UnsolicitedVaccinationLog.xsd as a mapping component.

Step 3: Add functions and draw the mapping connections

In the source HL7 file, dates have the format YYYYMMDD. To convert such dates to a format suitable for the target XML file YYYY-MM-DD, we will add a date conversion function, as follows:

1. Drag the parse-date function from the Libraries window into the mapping. For more information about functions, see Working with Functions.
2. On the Insert menu, click Constant, and enter \[Y,4-4\][M,2-2][D,2-2] as constant value.
3. Draw the following connections:

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>format input item of the parse-date function</td>
<td>Supplies to the function the format according to which the date should be parsed.</td>
</tr>
<tr>
<td>RXA-3</td>
<td>value input item of the parse-date function</td>
<td>Supplies to the function the actual value to be formatted.</td>
</tr>
<tr>
<td>result output item of the parse-date function</td>
<td>startdate input item of the XML target</td>
<td>Supplies the value returned by the function to the startdate XML node.</td>
</tr>
</tbody>
</table>

Follow the same steps to a new parse-date function, so as to convert the value of RXA-4 and copy it to the enddate item of the target XML.

The patientname item in the second target XML is created by concatenating two fields of the source HL7 file: FN-1 (Surname) and XPN-2 (Given Name). To achieve this, this drag the concat function from the Libraries window onto the mapping. Also, add a constant that will supply the space character to separate the two names.

Draw the remaining mapping connections as shown in the example mapping (these are too many to be listed here; essentially, each connection copies values from the source HL7 file to the target XML files. For basic information about mapping connections, see Working with Connections.)

To preview the output generated by any of the target XML components, first click its Preview ( ) button, and then click the Output tab.
7.5.9  IATA PADIS

PADIS (Passenger and Airport Data Interchange Standards) are a set of messages using the EDIFACT (ISO 9735) syntax.

- MapForce currently supports only messages using the UNH/UNT message header and trailer segments.
- MapForce supports the collections IATA.00.1 to IATA.08.1.

To add an IATA PADIS file as mapping component:

1. On the **Insert** menu, click **EDI**.
2. Click the "IATA" collection under "IATA PADIS", and select the check box next to the required message type(s).

3. When prompted to supply an instance file, click **Browse**, and open the source IATA file to be used as mapping source. If you are mapping to an IATA PADIS component, click **Skip**.

The IATA component is now added to the mapping, and you can draw mapping connections to or from it. To change the settings of the IATA component, right-click it, and then select **Properties** from the context menu (see also **EDI Component Settings**).

7.5.9.1  Example: Mapping IATA to XML

The **IATA_FlightInformationReport.mfd** file available in the `<Documents>\Altova\MapForce2019\MapForceExamples` folder shows an example mapping of an IATA PADIS file to an XML target file.
This mapping consists of a source IATA component which includes a FLIRES message, a target XML component, and two user-defined functions, `parseDateTime` and `airportcode2Names` (which are reused multiple times).

To view the instance file read or written by the source or target components, as well as other settings, right-click the component, and select **Properties** (see also **EDI Component Settings**).

To view the structure of each user-defined function, double-click it. To go back to the main mapping, click the **Return to Main Mapping** button available in the upper-left corner of the mapping area. For more information, see **User-Defined Functions**.

In this mapping, the user-defined functions have the following purpose:

1) The `parseDateTime` function receives as arguments the date and time from IATA date and time elements and processes them in order to produce an `xs:dateTime` value required in the target XML file. To achieve this goal, it makes use of various functions available in the MapForce core library, such as
2) The *airportcode2Names* function receives as argument the F3223 or F3224 field values (the airport code) and returns the full name of the airport. To achieve this, the function contains a Value-Map component which maps each airport code to an airport name, as follows:

![Value-Map Properties](image)

For more information about Value-Map components, see [Using Value-Maps](#).
On the target side, the XML component uses a technique called "duplication of nodes". Namely, the node Station is being duplicated multiple times on the mapping component. This makes it possible to map values from multiple source items to the same target item (see Duplicating Input).

To preview the output produced by the mapping, click the Output tab.

7.5.10 NCPDP SCRIPT

SCRIPT is a standard for exchanging electronic information in the US healthcare sector developed by the NCPDP (National Council for Prescription Drug Programs), see http://www.ncpdp.org. MapForce supports mapping data between files in SCRIPT format and any other format supported by MapForce, including XML, CSV, Microsoft Excel, databases, and others. The supported NCPDP SCRIPT versions are: 8.1, 10.6, 10.10.

The default NCPDP SCRIPT version in MapForce is 10.10.

Note: To enable support for NCPDP SCRIPT versions other than the default one, download and install the "NCPDP SCRIPT Configuration Files" package from the Altova download center (https://www.altova.com/mapforce/download/libraries).

NCPDP SCRIPT can use either the XML format or the UN/EDIFACT format. This documentation is applicable to the UN/EDIFACT version of NCPDP SCRIPT. If you have the NCPDP XML Schemas, MapForce will support mapping data to or from SCRIPT implicitly, using the standard XML mapping commands and validation (to add XML files to the mapping, use the menu command Insert | XML Schema/File).

Note that MapForce does not support sending or accepting SCRIPT messages as part of data communications between SCRIPT parties. The role of MapForce is to read data in SCRIPT format and map it to other formats, or convert data from other formats into SCRIPT format.

NCPDP SCRIPT files in UN/EDIFACT format can be added to the mapping as follows:

1. On the Insert menu, click EDI.
2. Select the "NCPDP SCRIPT" collection, and then select the check box next to the required message type(s).
3. Click OK. You are now prompted to supply an instance file.

4. Do one of the following:
   - If you want to map data from an NCPDP SCRIPT message to some other format, click **Browse** and select the source SCRIPT file. It is also possible to open files from a URL, see Adding Components from a URL.
   - If you want to map data to an NCPDP SCRIPT message, or supply the source file at a later time, click **Skip**.

At this stage, the structure of the selected NCPDP SCRIPT message(s) is displayed on the mapping (in MapForce terminology, an "EDI Component"). You can now either draw mapping connections to this component (in order to map data to it), or from this component, in order to map data from it. See also:

- Adding EDI Components to the Mapping
- Running EDI Mappings
- EDI Component Settings
- EDI Component Validation

### 7.5.10.1 NCPDP SCRIPT Validation Rules

With respect to NCPDP SCRIPT messages, MapForce performs the following validation checks:

- **UIB/S001/F0001** should be "UNOA"
- **UIB/S001/F0002** should be "0"
- **UIH/S306/F0329** should contain message type "SCRIPT"
- **UIH/S306/F0316** should contain message version number that is defined in selected configuration files
- **UIH/S306/F0318** should contain message release number that is defined in selected configuration files
- **UIH/S306/F0326** should contain message function (or the message type in MapForce terminology)
- **UIT/F0062** should match to corresponding **UIH/F0062** (message header/trailer) or not present (optional field)
- **UIT/F0074** should contain segments count or not be present (optional field)
- **UIZ/F0036** should contain messages count or not be present (optional field).

See also [EDI Component Validation](#).

### 7.5.10.2 NCPDP SCRIPT Auto-Completion Rules

When generating an NCPDP SCRIPT file, MapForce automatically generates certain fields or values, according to the rules below (referred to as "automatic data completion", or "auto-completion" rules). To disable this behaviour, clear the **Auto-complete missing fields** check box in [EDI Component Settings](#).

The following fields will be automatically completed:

#### Interchange Header
- **UIB/S001/F0001** = controlling agency + syntax version number = "UNOA"
- **UIB/S001/F0002** = syntax version number = "0"
- **UIB/S002/F0004** = "Sender"
- **UIB/S003/F0010** = "Recipient"
- **UIB/S300/F0017** = current date
- **UIB/S300/F0114** = current time

#### Message Header
- **UIH/S306/F0329** = "SCRIPT", message type
- **UIH/S306/F0316** = "010" for 10.6, message version number from configuration files.
- **UIH/S306/F0318** = "006" for 10.6, message release number from configuration files.
- **UIH/S306/F0326** = message function (or message type from a MapForce perspective)
- **UIH/F0062** = "0", message reference number

#### Message Trailer
- **UIT/F0062** = **UIH/F0062**
- **UIT/F0074** = number of segments in the message.

#### Interchange Trailer
- **UIZ/F0036** = message count

### 7.5.11 SAP IDoc

SAP IDoc (intermediate documents) documents are used to exchange business data between SAP and non-SAP applications. The documents are a form of intermediate data storage which can be exchanged between different systems.

An IDoc is structured as follows:
Control Record: contains control information about the IDoc: sender, receiver, message type, and IDoc type. The control record format is similar for all IDoc types.

Data Segment: contains the actual data of the segment as well as other metadata: header, segment no. and type as well as the fields containing the data.

Status Records: contain info on the current status of the document, i.e. the currently processed stages, and the stages that still need to be processed. The status format is identical for all types of IDoc.

The version number in the port definition defines the systems you are communicating with. The major differences between the versions are the various name lengths used in the various elements and the use of extensions. SAP R3 version 4.X supports long names (as well as extensions) while the previous versions do not.

Port Version 1: Releases 2.1 and 2.2.
Port Version 2: Releases 3.0, 3.1 and R/2 systems.
Port Version 3: Release 4.x (default value)

MapForce treats IDoc components as fixed-length files of length 30 char for Message type, 30 for IDoc type, and 27 for segment fields.

For an example which illustrates mapping data from SAP IDoc to XML, see <Documents>\Altova\MapForce2019\MapForceExamples\IDoc_Order.mfd.

To add an SAP IDoc document as mapping component:

1. Select the menu option Insert | EDI. This opens the Browse EDI collections dialog box.
2. Click the SAP IDoc entry in the list box.
3. Click Browse and select the IDoc parser report (also known as "IDoc definition file").

The IDoc parser report is created from the SAP system using the transaction WE60 ("Documentation for IDoc types"). Note that the file must be exported from SAP in uncompressed format. For information on how to invoke transaction WE60 and generate the parser report, refer to the SAP IDoc documentation. The MapForce Examples folder includes a sample parser report file (see <Documents>\Altova\MapForce2019\MapForceExamples\ORDERS01-Parseridoc.txt).
4. Click **OK**. MapForce prompts you to optionally select a sample EDI file.

5. If you would like to map data *from* an IDoc file, click **Browse** and select the IDoc (*.idoc) file that supplies the data. A sample IDoc file is available in the MapForce Examples folder (see `<Documents>\Altova\MapForce2019\MapForceExamples\ORDERS.idoc`). Otherwise, click **Skip**.

6. The Component Settings dialog box is displayed. This enables you to review the settings of the IDoc component before adding it to the mapping. You can change these settings at any time later if required (see [Changing the Component Settings](#)).
7. Click OK to close the Component Settings dialog box and add the IDoc component to the mapping area.

### 7.5.12 TRADACOMS

TRADACOMS (Trading Data Communications) is a UK-specific Electronic Data Interchange standard used in the retail business. It was introduced in 1982 as the first EDI standard for UK trade and industry. Although it has many similarities with the UN/EDIFACT standard, TRADACOMS is one of the precursors of EDIFACT and uses different structures within messages.
MapForce implements the base TRADACOMS specification as laid out in the “TRADACOMS Manual of Standards for Electronic Data Interchange”, published in January 1993 by the Article Numbering Association (ANA) UK, now known as GS1 UK (https://www.gs1uk.org). For other TRADACOMS versions, MapForce can be customized to process new message types, data elements, and code values, by means of configuration files.

You can work with the TRADACOMS format in MapForce in the same way as with other supported EDI formats, as follows:

- You can map TRADACOMS files with any other data formats supported by MapForce, in both directions (either as data source or data target).
- Mapping data to or from TRADACOMS format is available in the BUILT-IN language (used when previewing mappings, or in MapForce Server and FlowForce Server execution) and in code generation languages (C++, C#, Java). When you deploy the mapping to a FlowForce Server running on a different machine, the deployed package includes automatically the configurations of selected TRADACOMS message types and all code lists that are used by their data elements. Likewise, in the generated code, MapForce generates classes for the configuration groups, segments, and data elements.
- You can flexibly define which validation events should stop the data conversion, which ones should result in rejected (or accepted) records, and which ones should be ignored.
- You can enable or disable data auto-completion. When "Automatic data completion" (or "auto-completion") is enabled, MapForce fills some of the values automatically when generating TRADACOMS files.
- You can customize the message types, data elements, and code lists by means of configuration files, either globally or locally (see Configuration Files).
- You can set the encoding of parsed or generated files.

The following TRADACOMS file types and message types are available by default in MapForce:

<table>
<thead>
<tr>
<th>File Type</th>
<th>Message Type</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>THE PRODUCT INFORMATION FILE</td>
<td>PROHDR (Product File Header)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>PROINF (Product Details)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>PROTLR (Product File Trailer)</td>
<td>8</td>
</tr>
<tr>
<td>THE PRICE INFORMATION FILE</td>
<td>PRIHDR (Price File Header)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>PRIINF (Price Details)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>PRITLR (Price File Trailer)</td>
<td>8</td>
</tr>
<tr>
<td>THE CUSTOMER INFORMATION FILE</td>
<td>CUSHDR (Customer Information Header)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>CUSINF (Customer Information Details)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>CUSTLR (Customer Information Trailer)</td>
<td>8</td>
</tr>
<tr>
<td>THE ORDER FILE</td>
<td>ORDHDR (Order File Header)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>ORDERS (Order Details)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>ORDTLR (Order File Trailer)</td>
<td>9</td>
</tr>
<tr>
<td>File Type</td>
<td>Message Type</td>
<td>Version</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>THE PICKING INSTRUCTIONS FILE</td>
<td>PICHDR (Picking Instructions File Header)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>PICKER (Picking Instructions File Details)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>PICTLR (Picking Instructions File Trailer)</td>
<td>4</td>
</tr>
<tr>
<td>THE DELIVERY NOTIFICATION FILE</td>
<td>DELHDR (Delivery File Header)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>DELIVR (Delivery Details)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>DELTLR (Delivery File Trailer)</td>
<td>9</td>
</tr>
<tr>
<td>DELIVERY CONFIRMATION FILE</td>
<td>DLCHDR (Delivery Confirmation Header)</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>DLCDET (Delivery Confirmation Details)</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>DLCTLR (Delivery Confirmation Trailer)</td>
<td>5</td>
</tr>
<tr>
<td>THE INVOICE FILE</td>
<td>INVFIL (Invoice File Header)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>INVOIC (Invoice Details)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>VATTLR (VAT Trailer)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>INVTLR (Invoice File Trailer)</td>
<td>9</td>
</tr>
<tr>
<td>THE CREDIT NOTE FILE</td>
<td>CREHDR (Credit Note File Header)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>CREDIT (Credit Note Details)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>VATTLR (File VAT Trailer)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>CRETLR (Credit Note File Trailer)</td>
<td>9</td>
</tr>
<tr>
<td>STATEMENT/REMITTANCE FILE</td>
<td>SRMHDR (Statement/Remittance Details File Header)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>SRMINF (Statement/Remittance Line Details)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>SRMTLR (Statement/Remittance Details File Trailer)</td>
<td>9</td>
</tr>
<tr>
<td>UPLIFT INSTRUCTION FILE</td>
<td>UPLHDR (Uplift File Header)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>UPLIFT (Uplift File Details)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>UPLTLR (Uplift File Trailer)</td>
<td>4</td>
</tr>
<tr>
<td>UPLIFT CONFIRMATION FILE</td>
<td>UCNHDR (Uplift Confirmation Header)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>UCNDET (Uplift Confirmation Details)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>UCNTLR (Uplift Confirmation Trailer)</td>
<td>3</td>
</tr>
<tr>
<td>THE STOCK SNAPSHOT FILE</td>
<td>SNPHDR (Stock Snapshot Header)</td>
<td>3</td>
</tr>
<tr>
<td>File Type</td>
<td>Message Type</td>
<td>Version</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>---------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>SNPSTS (Stock Snapshot Details)</td>
<td>SNPSTS (Stock Snapshot Details)</td>
<td>3</td>
</tr>
<tr>
<td>SNPTLR (Stock Snapshot Trailer)</td>
<td>SNPTLR (Stock Snapshot Trailer)</td>
<td>3</td>
</tr>
<tr>
<td>THE STOCK ADJUSTMENT FILE</td>
<td>SADHHDR (Stock Adjustment Header)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>SADDET (Stock Adjustment Details)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>SADTLR (Stock Adjustment Trailer)</td>
<td>3</td>
</tr>
<tr>
<td>AVAILABILITY REPORT FILE</td>
<td>AVLHDR (Availability File Header)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>AVLDET (Availability Report Details)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>AVLTLR (Availability File Trailer)</td>
<td>4</td>
</tr>
<tr>
<td>GENERAL COMMUNICATIONS FILE</td>
<td>GENHDR (General Communications File Header)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>GENRAL (General Communications Text)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>GENTLR (General Communications Trailer)</td>
<td>3</td>
</tr>
<tr>
<td>COMPLEX ORDER FILE</td>
<td>CORHDR (Complex Order File Header)</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>CORDER (Complex Order Details)</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>CORTLR (Complex Order Trailer)</td>
<td>6</td>
</tr>
<tr>
<td>THE ACKNOWLEDGEMENT OF ORDER FILE</td>
<td>ACKHDR (Acknowledgement File Header)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>ACKMNT (Acknowledgement Details)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>ACKTLR (Acknowledgement Trailer)</td>
<td>4</td>
</tr>
<tr>
<td>PRODUCT PLANNING REPORT FILE</td>
<td>PPRHDR (Product Planning Report Header)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>PPRDET (Product Planning Report Details)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>PPRTL (Product Planning Report Trailer)</td>
<td>2</td>
</tr>
<tr>
<td>THE PAYMENT ORDER FILE</td>
<td>PAYHDR (Payment Order File Header)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>PAYINF (Payment Order Line Details)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>PAYTLR (Payment Order File Trailer)</td>
<td>3</td>
</tr>
<tr>
<td>THE DEBIT ADVICE FILE</td>
<td>DRAHDR (Debit Advice File Header)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>DRAINF (Debit Advice Line Details)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>DRATLR (Debit Advice File Trailer)</td>
<td>3</td>
</tr>
<tr>
<td>THE CREDIT ADVICE FILE</td>
<td>CRAHDR (Credit Advice File Header)</td>
<td>3</td>
</tr>
</tbody>
</table>
### File Type | Message Type | Version
---|---|---
| CRAINF (Credit Advice Line Details) | 3 |
| CRATLR (Credit Advice File Trailer) | 3 |
| THE EXCEPTION CONDITION FILE | EXCHDR (Exception Condition File Header) | 3 |
| | EXCINF (Exception Condition Line Details) | 3 |
| | EXCTL (Exception Condition File Trailer) | 3 |
| LOCATION PLANNING REPORT FILE | LPRHDR (Location Planning Report Header) | 2 |
| | LPRDET (Location Planning Report Details) | 2 |
| | LPRTL (Location Planning Report Trailer) | 2 |
| THE UTILITY BILL FILE | UTLHDR (Utility Bill File Header) | 3 |
| | UTLBIL (Utility Bill File Details) | 3 |
| | UVATLR (Utility Bill VAT Trailer) | 3 |
| | UTLTRL (Utility Bill File Trailer) | 3 |

**Note:** The RSGRSG (Reconciliation Message) version 2 is also supported. This message type may occur in all file types.

### 7.5.12.1 Adding TRADACOMS Files as Mapping Components

**To add a TRADACOMS file as a data mapping source:**

1. Do one of the following:
   - On the **Insert** menu, click **EDI**.
   - Click the **Insert EDI file** toolbar button.
2. On the Browse EDI Collections dialog box, click **TRADACOMS**, and select the message type(s) to be included in the message.
3. Click **OK**. You are now prompted to supply an instance file.

4. If you want to specify the source TRADACOMS file later, click **Skip**. Otherwise, click **Browse** and choose how you would like to open the TRADACOMS file:
   - If you want to open the file from your local drive or network, browse for the file, and then click **Open**.
   - If you want to open the file from a URL, click **Switch to URL** (for more information, see Adding Components from a URL).
   - If you have previously defined the file as a Global Resource, and would like to open it from Global Resources, click **Switch to Global Resources** (see Altova Global Resources).

5. On the Component Settings dialog box, set or change the settings if required (see EDI Component Settings).

To add a TRADACOMS file as a data mapping target:

1. Follow the steps 1 and 2 above.
2. When prompted to supply an instance file, click **Skip**.
7.5.12.2 The TRADACOMS Component in MapForce

In MapForce, the TRADACOMS component replicates the structure of a generic TRADACOMS transmission (with optional batching feature), while retaining all features common to other MapForce EDI-related components.

A sample TRADACOMS component in MapForce

The topmost item of the component, File, displays the name of the TRADACOMS interchange file being processed or generated by the component. "Default" indicates that no file has been assigned yet. The File/String (File/String) button displays advanced features common to all file-based MapForce components. It provides, among other options, the ability to read or process multiple files dynamically (for more information, see Processing Multiple Input or Output Files Dynamically).

The "Envelope" and "Interchange" structures are generic for all MapForce EDI-related components. Where applicable, they provide the ability to process multiple interchanges within the same MapForce component.

The STX and END structures are specific to the TRADACOMS format. They denote the "Start of Transmission" and "End of Transmission" segments, respectively.

The BAT and EOB structures are specific to the TRADACOMS format. They denote the "Start of Batch" and "End of Batch" segments, respectively. Such segments are applicable if you are reading data from a TRADACOMS interchange file with batches, or if you want to generate a file with batches. Since there may exist multiple batches in the same transmission, a Batch sequence is available as parent of the BAT sequence.

The Message_Code structure (Message_INVFIL, in the sample above) corresponds to the TRADACOMS "file". The Select EDI Message Types button (Select EDI Message Types) opens a dialog box where you can change the type of the file (for example, "Invoice File", "Orders File", and so on). By virtue of the existing MapForce EDI functionality, you can also include multiple file types in the same
component. Note, however, that the TRADACOMS specification recommends one file and one type of file in each interchange transmission.

Any TRADACOMS file ("Invoice File", in this example) contains a Header Message, one or more Details Messages, and a Trailer Message. In the component above, these are the INVFIL (Invoice File Header), INVTLR (Invoice File Trailer), the VATTLR (File VAT Trailer) and multiple INVOIC (Invoice Details) messages.

As with any other MapForce component, the input and output connectors (small triangles) displayed on each side of the component provide the ability to map each individual data element or segment to or from other data types or formats supported by MapForce. Likewise, when you move the mouse over any item on the component, you can view additional information about it (such as the minimum and maximum allowed occurrences), provided that tips are enabled from the View | Show Tips menu.

Finally, you can change these and other settings by double-clicking the component head and opening the Component Settings dialog box (see EDI Component Settings). Note that this dialog box is generic for all EDI-related component types and, thus, some of the options might not be applicable to TRADACOMS.

7.5.12.3 **Validation and Automatic Data Completion**

When you run a mapping that reads data from or writes data to a TRADACOMS structure, MapForce performs structural data validation checks according to the TRADACOMS specification, and displays any validation errors in the Messages window.

As shown in the sample above, the validation messages specific to TRADACOMS (information, warnings, and errors) are displayed in addition to the generic validation messages common to any MapForce mapping. For more information about MapForce validation in general, see Validating Mappings.

The following factors determine how MapForce validates the parsed or generated TRADACOMS files:

- The validation constraints defined in the configuration files available in the MapForce installation folder (subfolder MapForceEDI\TRADACOMS). These configuration files
supply, on one hand, the default validation rules of the TRADACOMS specification. On the other hand, they provide the means to adapt the TRADACOMS format to custom requirements. In particular, it is possible to modify the data elements, segments, or code values defined in the configuration files, and thus influence both the outcome of validation and the mapping execution. For more information about the configuration files, see Configuration Files.

- The validation logic built into MapForce. This includes MapForce internal data integrity checks that may not be enforced by means of configuration files.
- Any custom validation settings that you have defined from the MapForce graphical user interface (see EDI component validation). To view or change the current EDI validation settings of any EDI component, including TRADACOMS, double-click the header of the component, and then click Validation on the Component Settings dialog box.

When writing to a TRADACOMS structure, MapForce automatically fills in the contents of those data elements for which the value can be calculated or is predefined. This is referred to as "automatic completion" (or "auto-completion"). To disable this behavior, clear the Auto-complete missing fields check box from the Component Settings dialog box (see EDI Component Settings).

The following TRADACOMS validation rules cause MapForce to either raise validation errors (during file parsing or generation) or to auto-complete missing fields (during file generation):

1. The segments STX (Start of Transmission) and END (End of Transmission) must exist.
2. If STDS-1 has the value 'ANAA' then a Reconciliation Message (RSGRSG) must exist before the end of transmission (END). Otherwise, no Reconciliation Message (RSGRSG) must be present.
3. If STDS-1 has the value 'ANAA' then:
   a. The value of RSGA in the Reconciliation Message must be equal to the value of SNRF in the STX segment.
b. The value of RSGB in the Reconciliation Message must be equal to the value of UNTO-1 in the STX segment.

4. TRDT-1 must contain the date (YYMMDD) and TRDT-2 must contain the time (HHMMSS) of transmission (current date and time).

5. If the Batch Header (BAT) is present then the Batch Trailer (EOB) must also be present, and the number of messages in the batch must be available in the NOLI (Number of Messages in Batch) data element.

6. The MSRF (Message Reference) data element in the Message Header (MHD) must contain the consecutive count of messages within the transmission, starting with 1.

7. The NOSG (Number of Segments in Message) data element in the Message Trailer (MTR) must contain the number of segments, including MHD and MTR.

8. When present, the Reconciliation Message (RSGRSG) must consist of one segment (RSG), except the Message Header and the Message Trailer.

9. The NMST (Number of Messages in Transmission) data element in the END segment must contain the number of messages in interchange (count of MHD segments).

10. In general, when reading a TRADACOMS structure, MapForce expects that the interchange environment is of type "computer to computer" (or, in TRADACOMS terminology, "intelligent terminal to intelligent terminal"). Therefore, a segment such as MHD = 12 + ORDHDR :3 would trigger a validation error, since it contains extra leading and trailing spaces.

11. String data must be in upper case. When generating TRADACOMS output, MapForce converts string data to upper case.

Additionally, as already mentioned, any validation rules defined in the configuration files will also affect TRADACOMS parsing and generation.

7.5.12.4 Preserving Leading Zeros During Conversion

In the TRADACOMS specification, the data type of the TRDT-1 and TRDT-2 (Date and Time of Transmission) data elements is defined as "decimal". This means that, by default, when you map data from the TRDT-1 or TRDT-2 fields, any leading zeros in these fields would be trimmed during conversion, and thus produce undesired results. This may also be the case of any other items which are defined as "decimal", but store values which are meant to be treated as string.

You can instruct MapForce to treat numeric fields as string (and thus preserve the leading zeros) by applying the `core | format-number` MapForce function. In the following example, the value of the TRDT-1 item in the source file is "020312". The normal output of this item would be "20312" (as a result of conversion to decimal), which is not the desired result. Therefore, to keep the leading zero, the `format-number` function has been added to the mapping. (For information about working with functions, see Working with Functions.)
Preserving leading zeros with the format-number function

As illustrated above, the function has the following two input arguments:

1. The value to be formatted (in this case, "020312").
2. The format mask "000000".

To test the output of the function, this example uses a simple output component (see Simple Output). When you click the Output button, the output of the mapping (which is the same as the result of the function) is "020312", as intended.

7.5.12.5 Configuration Files

By default, the MapForce configuration files and the underlying MapForce processing logic reflects the base version of the TRADACOMS standard (931). Therefore, if you use this TRADACOMS version, customization of the TRADACOMS configuration files is not necessary. However, if you want to make changes to the existing configuration for any reason, it is possible to do so. Note that any changes made to the standard TRADACOMS configuration files will take immediate effect and will affect the way MapForce parses or writes TRADACOMS structures.

The MapForce configuration files applicable to TRADACOMS are stored in the following directory: C:\Program Files\Altova\MapForce2019\MapForceEDI\TRADACOMS. Should you need to make changes to the TRADACOMS configuration files, you can start by creating a sibling copy of the TRADACOMS directory, for example "TRADACOMS1", edit the directory permissions not to be "read-only", and then edit the files in it according to your custom requirements. All configuration files adhere to the XML syntax and, so it is recommended that you use an XML editor to modify the files.

<table>
<thead>
<tr>
<th>File</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDI.Collection</td>
<td>The EDI.Collection file contains a list of all messages in the current TRADACOMS configuration directory (for example, C:\Program Files\Altova\MapForce2019\MapForceEDI)</td>
</tr>
</tbody>
</table>
### File | Purpose
--- | ---
"\TRADACOMS). Any messages listed in the collection file become available for selection in the "Browse EDI Collections" dialog box when you add a TRADACOMS component to the mapping (Insert | EDI). Therefore, if you want to suppress any message type on this dialog box, comment out or remove their respective XML element from the EDI.Collection file.

For example, commenting out the following line makes the LPRHDR message type unavailable:

```xml
<!--<Message Type="LPRHDR" File="LPRHDR.Config"
Description="LOCATION PLANNING REPORT FILE"/>-->
```

**Envelope.Config**

Defines the structure of the TRADACOMS component in MapForce (that is, elements that are surrounded by the Envelope sequence). This file is validated against a schema handled by MapForce internally, and it should not be edited.

**Tradacoms.Codelist**

This file defines the TRADACOMS codes and the values that they may contain, and is used for validation of input/output TRADACOMS files. You can add custom code lists to this file, if required.

**Tradacoms.Segment**

This file defines the Segment, Composite and Field names of the TRADACOMS files, and is used when parsing a TRADACOMS file. Changes made to this file are global customizations, and apply to all segments and messages.

**[Message].Config**

Defines the structure of a single TRADACOMS message. Changes made to this file are treated as local (inline) customizations.

For more information, see Customizing EDI Structure and Customizing EDI Validation.

### 7.5.12.6 Example: Converting a TRADACOMS Invoice to XML

This example provides instructions on how to create a mapping design that converts data from a TRADACOMS invoice file to XML format. The mapping design file created in this example is also available at the following path: <Documents>\Altova\MapForce2019\MapForceExamples \TRADACOMS_Invoice.mfd.
The mapping accomplishes the following goals:

1. Convert a source TRADACOMS file (INVFILE.edi) to an XML file which is valid against an existing schema (Order.xsd). Both the source TRADACOMS file and the schema file (Order.xsd) are available in the MapForce Examples folder (<Documents>\Altova\MapForce2019\MapForceExamples).  
2. The date in the TRADACOMS file has decimal format so it must be converted to xs:dateTime format in the target XML file. 
3. For each line item in the purchase order, the tax amount must be calculated as a decimal value, rounded to three decimal places (thousandths). For the scope of this example, we will assume that the tax is calculated according to the following formula:

   \[
   \text{Tax} = \text{round} \left( \text{LEXC} \times \text{VATP} \times 10 \right) \times 0.001
   \]

   Where \text{round} rounds the value to the nearest integer.

To accomplish the goals, take the following steps:

1. Add the source and target components to the mapping area and draw the mapping connections between them.
2. Use date functions to convert the date to the required format.
3. Use math functions to calculate the tax value.
Step 1: Add the components to the mapping area

To add the source TRADACOMS component:

1. On the Insert menu, click EDI.
2. Click the TRADACOMS collection folder, and, under Message Types, select the INVOICE FILE.
3. When prompted to supply a sample file, browse for the INVFIL.edi file available in the MapForce Examples folder (<Documents>\Altova\MapForce2019\MapForceExamples>).
4. When prompted to enter the component settings, leave the default values as is, and click OK.

To add the target XML component:

2. When prompted to supply a sample file, click Skip. (Because this is a target component, the file will be generated, so there is no need to browse for an existing one.)
3. When prompted to select a root element, select "Order".

In this example, the data will be mapped from the source TRADACOMS file to the target XML file as follows:

<table>
<thead>
<tr>
<th>Source (TRADACOMS)</th>
<th>Target (XML)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVN</td>
<td>/Order/Header/Number</td>
</tr>
<tr>
<td>IVDT</td>
<td>/Order/Header/Received</td>
</tr>
</tbody>
</table>
Using the table above as reference, you can now draw mapping connections from all of the above items in the source component to their destination in the target component. For now, you can omit those fields that require calculated values (that is, the ones marked with * in the table); these will be handled in the next steps. For general information about drawing connections in MapForce, see Working with Connections.

Tip: To search for a field by its name inside the MapForce component, press Ctrl + F.

Step 2: Add the date conversion functions
To convert the value of the IVDT item from a decimal value to xs:dateTime type, the mapping uses the format-number function from the core | conversion functions library. For a worked example, see Preserving Leading Zeros During Conversion.

The role of the second function, parse-date, is to convert the YYMMDD string value returned by the format-number to xs:dateTime type. It has two input arguments: (a) the value to be formatted and (b) the format mask. The format mask essentially instructs MapForce to treat the year, month, and date as having a width of exactly two characters each.

To add the functions above to the mapping, drag them from the Libraries window to the mapping area. To supply an argument to a function, insert a constant (the menu command is Insert |
Constant). For general information about working with functions in MapForce, see [Working with Functions](#).

**Step 3: Add the tax calculation functions**

The functions used to calculate tax according to the formula are **multiply** and **round**, available in the **core | math functions** library. The multiply function is extendable (that is, it can take a variable number of arguments). In this example, it takes LEXC, VATP, and the integer 10 as arguments. The first two values are connected from the source TRADACOMS component, while the integer is supplied by a constant. The result of the **multiply** function is then rounded to the nearest integer using the **round** function. Finally, the result of the **round** function is multiplied by 0.001 by means of a second **multiply** function. This result is then connected to the destination item of the XML component (/Order/LineItems/LineItem/Article/Tax).

To preview the output of the mapping design, click the **Output** tab in the lower part of the mapping window.

**Step 4 (optional): Style the mapping output with StyleVision**

Optionally, you can link the target XML component to a StyleVision Power StyleSheet (.sps) file. (For more information about StyleVision, see [https://www.altova.com/stylevision](https://www.altova.com/stylevision).) This would enable you to generate the mapping output as HTML, RTF, PDF, or 2007+ file (provided that StyleVision is installed and the prerequisites required by each format are in place, see [Styling Mapping Output with StyleVision](#)). To take this optional step, double-click the header of the XML component, and, next to "StyleVision Power StyleSheet file", browse for the **Order.sps** file available in the MapForce Examples folder. Back on the mapping area, you can now click the HTML, PDF, RTF, and Word 2007+ tabs to view the mapping output in the corresponding format.

### 7.5.13 UN/EDIFACT

UN/EDIFACT (the United Nations rules for Electronic Data Interchange for Administration, Commerce and Transport) is a set of standards for electronic document interchange, see [https://www.unece.org/cefact/edifact/welcome.html](https://www.unece.org/cefact/edifact/welcome.html). MapForce supports the messages contained in directories D93.A - D.18B.

The **default UN/EDIFACT version in MapForce is D.18B**.

A separate installer for all previous versions back to D93.A, including the configuration files required for Interactive EDI mappings, is available on the Altova MapForce Libraries page ([https://www.altova.com/components_mapforce.html](https://www.altova.com/components_mapforce.html)).

#### 7.5.13.1 Batch Versus Interactive EDIFACT Messages

MapForce supports mapping data to or from both batch and interactive UN/EDIFACT messages. In other words, the UN/EDIFACT component in MapForce provides the necessary structure required to read or write syntactically valid UN/EDIFACT format files (meant for either batch or interactive interchange). Also, some basic validation and auto-completion rules are enforced when reading or writing UN/EDIFACT format files, as further described in this documentation. Note that MapForce does not actually transmit files between parties that take part in the UN/EDIFACT interchange process; the role of MapForce is to generate such files or convert them to other...
There are no special requirements in MapForce to map data to or from batch EDIFACT messages. However, in order to map data to or from Interactive EDIFACT messages, the following requirements apply:

1. MapForce 2018 or later is required.
2. The MapForce configuration files required for supporting Interactive EDI messages must be installed separately. The installation file can be freely downloaded and installed from the Altova MapForce Libraries page (https://www.altova.com/mapforce/download/libraries). Once you do this, the required collections and message types become available on the "Browse EDI Collections" dialog box illustrated below.

To add to the mapping either a batch or an interactive UN/EDIFACT component, select the corresponding collection from the "Browse EDI Collections" dialog box. To open this dialog box, run the Insert | EDI menu command (or click the Insert EDI File toolbar button), see also Adding EDI Components to the Mapping.

Note: On the dialog box above, the suffix "INTERACTIVE" denotes a collection applicable for Interactive EDIFACT structure, for example "EDIFACT.D.2016A.INTERACTIVE". Collections applicable to batch structure include only the UN/EDIFACT directory version in their title, for example "EDIFACT.D.2016A". The same convention applies to the configuration directories (collections) available in the EDI configuration folder of MapForce (C:\Program Files\Altova\MapForce2019\MapForceEDI).

The structure of UN/EDIFACT batch files is different from that of files meant for interactive...
interchange. Therefore, an EDIFACT component in MapForce can be either batch OR interactive (and thus include either batch OR interactive messages), not both simultaneously. Likewise, the validation and auto-completion rules differ for both scenarios, as further described in the following topics:

- **UN/EDIFACT Validation Rules (Batch)**
- **UN/EDIFACT Auto-Completion Rules (Batch)**
- **UN/EDIFACT Validation Rules (Interactive)**
- **UN/EDIFACT Auto-Completion Rules (Interactive)**

### 7.5.13.2 UN/EDIFACT Validation Rules (Batch)

When validating UN/EDIFACT documents, MapForce performs the following checks:

- Whether a UNB and a UNZ segment exist.
- Whether UNB/S004 contains a valid date/time specification.
- Whether UNB/0020 and UNZ/0020 contain the same value.
- Whether UNZ/0036 contains the correct number; which is defined as the number of functional groups, if present, or the number of messages. If there are functional groups, this should be the number of functional groups, otherwise it should be the number of messages contained in the interchange.

Each functional group is checked:

- Whether it contains a matching UNG and UNE pair.
- Whether UNG/S004 contains a valid date/time specification.
- Whether UNE/0060 contains the correct number of messages contained in the functional group.

Each message is checked:

- Whether it contains a matching UNH and UNT pair.
- Whether UNH/S009/0052 contains the same value as UNG/S008/0052 of the enclosing functional group.
- Whether UNH/0062 and UNT/0062 contain the same value.
- Whether UNH/S009/0065 contains the correct message type specifier.
- Whether UNT/0074 contains the correct number of segments contained in the message.

### 7.5.13.3 UN/EDIFACT Auto-Completion Rules (Batch)

When generating UN/EDIFACT files, MapForce may automatically create certain fields or values, according to the rules below (subsequently referred to as "automatic data completion", or "auto-completion" rules). To disable this behaviour, clear the **Auto-complete missing fields** check box in **EDI Component Settings**.

Automatic data completion for UN/EDIFACT makes sure:

- That a UNB and a UNZ segment exist
- If either UNG or UNE exist, that the other ID also exists
- That a UNH and a UNT segment exist
- That UNB/S001 exists. If it does not contain data, the syntax level and syntax version
number from the user-defined settings are used (see the setting Extended | Syntax version number of the EDI Component Settings)

- That UNB/S002 and UNB/S003 exist.
- That UNB/S004 exists. If it does not contain data, the current date/time in EDI format is inserted.
- That UNZ/0036 exists. If it does not contain data, the number of functional groups or messages is calculated and inserted.
- That UNZ/0020 exists. If it does not contain data, the value from UNB/0020 is copied.

**Note:** Any fields not mentioned here are not inserted or created. The correct values cannot be ascertained automatically. Given a (target) parent element A (in the target EDI component) with child items x, y, and z - where y is mandatory, parent element A will only be created in the output file if the mandatory child element "y" in the target component has been mapped.

Functional group checking makes sure:

- That UNG/0038 exists. If it does not contain data, the name of the message is inserted.
- That UNG/S006 and UNG/S007 exist.
- That UNG/S004 exists. If it does not contain data, the current date/time in EDI format is inserted.
- That UNG/0051 exists. If it does not contain data, the first two characters of the controlling agency from the user-defined settings are inserted (see the setting Extended | Controlling agency of the EDI Component Settings)
- That UNE/0060 exists. If it does not contain data, the number of messages in the group is calculated and inserted.
- That UNE/0048 exists. If it does not contain a value, the value from UNG/0048 is copied.

Message checking makes sure:

- That UNH/S009/0065 exists. If it does not contain data, the name of the message is inserted.
- That UNH/S009/0052 and UNH/S009/0054 exist.
- That UNH/S009/0051 exists. If it does not contain data, the first two characters of the controlling agency from the user-defined settings are inserted (see the setting Extended | Controlling agency of the EDI Component Settings)
- That UNT/0074 exists. If it does not contain data, the number of segments in the message is calculated and inserted.
- That UNT/0062 exists. If it does not contain data, the value from UNH/0062 is copied.
- That UNH/0062 exists. If it does not contain data, the value from UNT/0062 is copied. (If only the trailer segment number is mapped, then the corresponding field in the header segment is supplied with the same value.)

### 7.5.13.4 UN/EDIFACT Validation Rules (Interactive)

The following following validation rules apply for MapForce components which contain Interactive EDIFACT messages:

- If UIB is present, then all following UIH/S302 should match UIB/S302
- UIH/S306/F0065 should contain message type (validated by the parser)
- UIH/S306/F0052 should contain message version number that is defined in selected configuration files
Data Sources and Targets

- **UIH/S306/F0054** should contain message release number that is defined in selected configuration files
- **UIT/F0340** should match to corresponding **UIH/F0340** (message header/trailer) or not be present (optional field)
- **UIT/F0074** should contain segments count or not be present (optional field)
- **UIZ/S302** should match **UIB/S302** or not be present (optional composite)
- **UIZ/F0036** should contain messages count or not be present (optional field).

### 7.5.13.5 UN/EDIFACT Auto-Completion Rules (Interactive)

The following auto-completion rules apply for MapForce components which contain Interactive EDIFACT messages:

- **UIB/S001/F0001** will be filled with controlling agency + syntax version number
- **UIB/S001/F0002** will be filled with syntax version number
- **UIB/S002/F0004** will be filled with "Sender"
- **UIB/S003/F0010** will be filled with "Recipient"
- **UIZ/S302/* will be filled with **UIB/S302/*
- **UIZ/F0036** will be filled with the message count
- **UIH/S302/* will be filled with **UIB/S302/* (if **UIB** and **UIB/S302** is present)
- **UIH/S306/F0065** will be filled with the message type
- **UIH/S306/F0052** will be filled with the message version number from configuration files
- **UIH/S306/F0054** will be filled with the message release number from configuration files
- **UIT/F0340** will be filled with **UIH/F0340** (this is an optional field but will be completed anyway)
- **UIT/F0074** will be filled with the number of segments in the message (this is an optional field but will be completed anyway).

### 7.5.13.6 Example: Mapping UN/EDIFACT to XML

This example shows how to map data from UN/EDIFACT messages to an XML schema, in order to produce an XML instance file for further processing. The mapping created in this example is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\EDI_Order.mfd`.

**Step 1: Add the UN/EDIFACT component to the mapping**

1. Create a new mapping and select one of the following transformation languages: Java, C#, C++, or BUILT-IN. In this example, Java is selected as transformation language.
2. On the **Insert** menu, click **EDI**.
3. On the EDI collections dialog box, select the EDIFACT collection, then select the **ORDERS** message type, and click **OK**.
4. When prompted to supply a sample EDI file, click **Browse** and open the `ORDERS.EDI` file from the `<Documents>\Altova\MapForce2019\MapForceExamples` directory. After you open the file, the Component Settings dialog box opens. This enables you to review the settings of the EDI component before adding it to the mapping. You can change these settings at any time later if required (see [EDI Component Settings](#)). Notice that the `ORDERS.EDI` file appears as Input EDI file.

![Component Settings dialog box](image)

5. Click **OK**. The EDI component is now displayed in the mapping area. Double-click the **Message_ORDERS** node to view its children items. To resize the component, click and drag the lower-right corner of the component window.

![Component resized](image)
Step 2: Add the target schema component to the mapping

1. On the Insert menu, click the XML Schema/File, and open the Order.xsd file from the <Documents>\Altova\MapForce2019\MapForceExamples> directory.
2. When prompted to supply a sample XML file, click Skip and select Order as the root of the target document.

At this point, both the source EDI component and the target XML schema are on the mapping area, so we are ready to start drawing the mapping connections.

Step 3: Map the EDI items

The EDI component displays the structure of a message based on the collection (ORDERS) we selected. Typically, not all of the nodes will actually contain data, so you must be sufficiently
familiar with the EDI documents being worked on, to locate the relevant nodes.

In MapForce, you map a source item and a target item by drawing a connection between them. For step-by-step instructions on how to create mapping connections in MapForce, see Working with Connections.

In this example, the following nodes (starting from the Group/Message_ORDERS node) will be mapped (connected) first:

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGM/C106/F1004</td>
<td>Order/Header/Number</td>
</tr>
<tr>
<td>SG2/NAD/C082/F3039</td>
<td>Order/Customer/Number</td>
</tr>
<tr>
<td>SG2/NAD/C080/F3036</td>
<td>Order/Customer/CompanyName</td>
</tr>
<tr>
<td>SG2/NAD/C059/F3042</td>
<td>Order/Customer/Address/Street</td>
</tr>
<tr>
<td>SG2/NAD/F3164</td>
<td>Order/Customer/Address/City</td>
</tr>
<tr>
<td>SG2/NAD/C819/F3229</td>
<td>Order/Customer/Address/State</td>
</tr>
<tr>
<td>SG2/NAD/F3251</td>
<td>Order/Customer/Address/ZIP</td>
</tr>
<tr>
<td>SG2/SG5/CTA/C056/F3412</td>
<td>Order/Customer/ContactName</td>
</tr>
</tbody>
</table>

At this stage, the mapping should look similar to the image below:

Continue the mapping process and map:
Step 4: Format the date

Drag the `to-datetime` function from the `edifact` library into the Mapping area. For instructions on how to work with functions in MapForce, see Working with Functions.

By supplying as arguments to this function the F2380 and F2379 components of the DTM/C507 element, we can create an appropriately formatted `Received` datetime.

We therefore map the following fields:

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG29</td>
<td>Order/LineItems</td>
</tr>
<tr>
<td>SG29/LIN/C212/F7140</td>
<td>Order/LineItems/LineItem/Article/Number</td>
</tr>
<tr>
<td>SG29/IMD/C273/F7008</td>
<td>Order/LineItems/LineItem/Article/Name</td>
</tr>
<tr>
<td>SG29/QTY/C186/F6060</td>
<td>Order/LineItems/LineItem/Article/Amount</td>
</tr>
<tr>
<td>SG29/SG33/PRI/C509/F5118</td>
<td>Order/LineItems/LineItem/Article/SinglePrice</td>
</tr>
</tbody>
</table>

Step 5: Filter the buyer purchase orders

At this point we want to filter the “Buyer” purchase orders. These can be identified by the party function code qualifier of the NAD (Name and address) segment. In this case, the value "BY" indicates a “Buyer” (Party to whom merchandise and/or service is sold).

1. Drag the `equal` function from the `core` library into the Mapping area.
2. Add a filter to the mapping (On the Insert menu, click Filter: Nodes/Rows).
3. Add a constant to the mapping (On the **Insert** menu, click **Constant**). Assign to the constant the value "BY" by entering "BY" into the text field:

Map the following items:

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG2/NAD/F3035</td>
<td>The input of the <strong>equal</strong> function</td>
</tr>
<tr>
<td>The constant &quot;BY&quot;</td>
<td>The input of the <strong>equal</strong> function</td>
</tr>
<tr>
<td>The result of the <strong>equal</strong> function</td>
<td>The <strong>bool</strong> input of the <strong>filter</strong> component</td>
</tr>
<tr>
<td>SG2/NAD</td>
<td>The <strong>node/row</strong> input of the <strong>filter</strong> component</td>
</tr>
<tr>
<td>The <strong>on-true</strong> result of the <strong>filter</strong> component</td>
<td>Order/Customer in the schema</td>
</tr>
</tbody>
</table>

The aim here is to only map data if the NAD node refers to a 'Buyer', as identified by the party function code qualifier 'BY'.

**Step 6: Calculate pricing and tax**

The final step in this task is to calculate the pricing and tax costs.
1. From the core library, drag two **multiply** and one **divide** function into the Mapping area.
2. Insert a Constant component (**Insert | Constant**). Make sure "Number" is selected as type, and enter 100.0 into the text field.
3. Map the following items:

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG29/QTY/C186/F6060</td>
<td>value1 of the first <strong>multiply</strong> function</td>
</tr>
<tr>
<td>SG29/SG33/PRI/C509/F5118</td>
<td>value2 of the first <strong>multiply</strong> function</td>
</tr>
<tr>
<td>The result of the first <strong>multiply</strong> function</td>
<td>Order/LineItems/LineItem/Article/Price</td>
</tr>
<tr>
<td>SG29/SG40/TAX/C243/F5278</td>
<td>value1 of the <strong>divide</strong> function</td>
</tr>
<tr>
<td>The constant &quot;100.0&quot;</td>
<td>value2 of the <strong>divide</strong> function</td>
</tr>
<tr>
<td>The result of the first <strong>multiply</strong> function</td>
<td>value1 of the second <strong>multiply</strong> function</td>
</tr>
<tr>
<td>The result of the <strong>divide</strong> function</td>
<td>value2 of the second <strong>multiply</strong> function</td>
</tr>
<tr>
<td>The result of the second <strong>multiply</strong> function</td>
<td>Order/LineItems/LineItem/Article/Tax</td>
</tr>
</tbody>
</table>

Your mapping should now look like this:
Clicking the output tab performs an "on the fly" transformation and presents you with the XML document result:
Customizing EDI Structure

By default, MapForce represents the structure of EDI files on the mapping according to their formal specifications. In most cases, it shouldn't be necessary to manually customize the default EDI configuration. However, if you need to process various non-standard EDI formats and variations, then customization is required. For example, you may need to slightly adapt the structure or validation rules of certain data elements. This can be achieved by adding a custom EDI collection to MapForce, and then configuring it according to your business needs, which is the subject of this chapter.

You can create EDI variations (custom EDI collections) for the following standards already supported by MapForce: EDIFACT (ISO 9735), ASC X12, HIPAA X12, HL7, NCPDP SCRIPT, or TRADACOMS. Creating a new EDI standard from scratch is not supported.

Creating a Custom EDI Collection

An "EDI Collection" is a directory which contains all configuration files applicable to a specific EDI standard. By default, all built-in EDI collections exist in MapForce at the following path: C:\Program Files\Altova\MapForce2019\MapForceEDI. (If you run 32-bit MapForce on a 64-bit operating system, adjust the directory path to C:\Program Files(x86)....)
It is not recommended to alter the contents of original files and folders in the **MapForceEDI** directory, unless you want to override, at your own responsibility, the default EDI validation and processing in MapForce. The recommended approach is to add new custom directories rather than modifying the existing ones, as shown below. This will also make it easier to migrate your custom configuration to a newer version of MapForce.

**To create a custom EDI collection:**

1. In the `C:\Program Files\Altova\MapForce2019\MapForceEDI` directory, create a sibling copy of a directory that corresponds to the standard which you want to customize (for example "EDIFACT.Custom").

2. Open MapForce.
3. On the **Insert** menu, click **EDI**. At this stage, your custom EDI collection should appear in the "Browse EDI Collections" dialog box in addition to the default ones.
So far, the custom collection that was just created has not been customized yet in any way, so it is not different to the default one. In order to provide you the ability to translate data between your custom EDI format and any other data format supported by MapForce (in both directions), the EDI collection must be customized. This is discussed in the next sections. For now, it is important to remember that a directory must satisfy the following criteria in order to become an EDI collection recognized by MapForce:

1. It must be a subdirectory of \C:\Program Files\Altova\MapForce2019\MapForceEDI\).
2. It must be a copy of any of the following directories: EDIFACT, HIPAA.X12, HL7, NCPDP SCRIPT, TRADACOMS, or X12.

### 7.5.14.2 EDI Configuration Files

An EDI collection (directory) may contain any of the following configuration files:

<table>
<thead>
<tr>
<th>File</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDI.Collection</td>
<td>The <strong>EDI.Collection</strong> file defines all message types present in the custom EDI collection that you are implementing. You can edit this file to add or remove message types that should be available for selection in MapForce (on the &quot;Browse EDI Collections&quot; dialog box). See also Adding or Removing Message Types.</td>
</tr>
<tr>
<td>Envelope.Config</td>
<td>The <strong>Envelope.config</strong> file defines the structure of the EDI mapping component in MapForce (that is, elements that are surrounded by the <strong>Envelope</strong> sequence). Under normal circumstances, this file</td>
</tr>
<tr>
<td>File</td>
<td>Purpose</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>should not be edited manually.</td>
</tr>
<tr>
<td>&lt;Message&gt;.Config</td>
<td>The EDI collection directory must include one or several &lt;Message&gt;.Config files. One such file corresponds to an EDI message type, and contains the group and segment definitions used in that message. The &lt;Message&gt;.Config files are referenced from the EDI.Collection file. You can modify the &lt;Message&gt;.Config files if you want to alter the structure or validation rules applicable to fields of a particular message. Note that changes to these files are considered local customizations (that is, they apply only to that specific message). For examples, see Customizing an EDIFACT message and Customizing an ANSI X12 transaction.</td>
</tr>
<tr>
<td>&lt;Filename&gt;.Segment</td>
<td>This file defines the data elements, segments, composites, and subcomposites of the EDI files, and is used when parsing the EDI file. You can modify the .segment files if you want to alter the structure or validation rules applicable to a field (data element) in all messages where it may occur. Changes made to this file are global customizations.</td>
</tr>
<tr>
<td>&lt;Filename&gt;.Codelist</td>
<td>This file defines EDI code values used by MapForce to validate the EDI input and output files in MapForce. In general, the code values defined here correspond to those defined by the respective EDI standard. Exception to this rule are some edge cases where the standard relies on external code values and does not fully enforce the list of possible values (see also EDI component validation). You can modify .codelist files if you need to change the default validation rules (for example, in order to relax validation, or add new codes that are not already defined in the EDI code list). For more information, see Relaxing Field Validation and Adapting Code Values from External Sources.</td>
</tr>
<tr>
<td>ParserErrors.Config</td>
<td>This file is applicable to the X12 and HIPAA X12 standards only. It defines the structure of &quot;ParserErrors_Message&quot; and &quot;ParserErrors_Group&quot; items on the EDI component in MapForce. These items contain information about parsing errors encountered during file processing and can be used to produce X12 997 (Functional Acknowledgement) and X12 999 (Implementation Acknowledgement) messages.</td>
</tr>
</tbody>
</table>

All of the files above are XML files and can be edited in an XML editor such as XMLSpy. They all refer to some XML schema (see EDI Configuration Schemas), with the exception of the EDI.Collection file. The EDI.Collection file has no explicit schema defined but is validated internally by MapForce.
7.5.14.3  EDI Configuration Schemas

Whenever you make a change to a configuration file, it is important to validate it against its applicable XML schema. The schema is available in the directory C:\Program Files\Altova\MapForce2019\MapForceEDI and it can be one of the following:

<table>
<thead>
<tr>
<th>File</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDIConfig.xsd (version &quot;3&quot;)</td>
<td>This configuration schema is valid starting with MapForce 2011, Release 2. If you have configuration files that are using a schema prior to version 3, they must be upgraded to use this schema (or a newer one). For more information, see Upgrading Older Configuration Files.</td>
</tr>
<tr>
<td>EDIConfig4.xsd</td>
<td>This configuration schema is valid starting with MapForce 2011, Release 3. It contains multiple enhancements to the original EDIConfig.xsd schema, including the ability to define custom recursive hierarchies which might be necessary when you map data to or from X12 or HL7 formats.</td>
</tr>
<tr>
<td>EDIConfig5.xsd</td>
<td>This configuration schema is valid starting with MapForce 2017. It introduces the ability to define a data element that has incomplete validation rules. With this schema, it is possible to define advanced field validation rules at character level (see Configuring Character-Level Validation).</td>
</tr>
<tr>
<td>EDIConfig6.xsd</td>
<td>This configuration schema is valid starting with MapForce 2018. It introduces the ability to map data to or from NCPDP SCRIPT format, see NCPDP SCRIPT.</td>
</tr>
</tbody>
</table>

The EDI configuration schema will also offer information about the structure of any elements that may be present in the configuration files. For example, below is a representation of an EDI Data Element structure as it appears in the EDIConfig4.xsd schema:
7.5.14.4 Upgrading Older Configuration Files

EDI configuration files contain a reference to (and are validated against) one of the XML schemas available in the MapForceEDI directory (see EDI Configuration Schemas). To view the schema and version of a configuration file, open the file in a text editor and check the following line at the beginning:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="../EDICfg5.xsd" Version="5"/>
```

MapForce 2019 recognizes custom EDI configurations that are based on EDICfg.xsd schema version 3 or later, introduced in MapForce 2011, Release 3. Compared to previous versions, this schema adds the ability to map data to or from multiple message types in the same EDI component. If you were using custom EDI configuration files that point to a schema version prior
to version 3, then your configuration files must be upgraded. Otherwise, when you attempt to add the custom EDI collection to the mapping, the following message appears on the "Browse EDI Collections" dialog box:

"The selected EDI collection uses an old configuration file version that supports only a single message type per component."

The following instructions show you how to upgrade the schema of an EDI configuration file from EDIConfig.xsd version 2, to EDIConfig.xsd version 3.

To upgrade the configuration files:

1. Copy Envelope.Config from the original configuration folder (for example, "EDIFACT") to the folder containing your customized EDI collection (for example, "EDIFACT.Custom").
2. Edit EDI.Collection, and change the root element's Version attribute from "2" to "3".
3. Add <Root File="Envelope.Config"/> after the </Meta> tag:

```
<?xml version="1.0" encoding="UTF-8"?>
<Messages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
Version="3">
  <Meta>
    <Version>D</Version>
    <Release>04B</Release>
    <Agency>UN</Agency>
  </Meta>
  <Root File="Envelope.Config"/>
  <Message Type="ORDERS" File="ORDERS.Config" Description="Purchase order message"/>
</Messages>
```

4. Edit ORDERS.Config, and change the root element's Version attribute from "2" to "3".
5. Add <Format standard="EDIFACT"/> (or, depending on the case, X12, or HL7) after the </Meta> tag.
6. Rename <Group name="Message"...> to <Group name="Message_ORDERS"...> (or whatever the custom message type is), and remove the outer group levels ("Envelope", "Interchange", and their segments) as shown below:
If a mapping was open while you were editing the configuration file, it should be reloaded. The connections will be automatically remapped from "Message" to "Message_ORDERS" item.

7.5.14.5 Adding or Removing Message Types

To add custom EDI message types (or transaction types, in X12 terminology), follow the steps below:

1. In the directory where your custom EDI collection is stored, create a `<Message>.Config` file for each message (transaction) type to be added to the collection.
2. Modify the `EDI.Collection` file to make a reference to each available message (transaction) type.

To remove existing message types:
Comment out the required Message elements in the EDI.Collection file.

Adding and configuring the <Message>.Config file

Each <Message>.config file you create must be valid according to its configuration schema (see EDI Configuration Files). To avoid problems, pay special attention to the following elements in the body of each message configuration file:

- Meta
- Format
- Include
- Message

The Meta element indicates the specification version, release, and agency, for example:

```xml
<Meta>
  <Version>D</Version>
  <Release>14B</Release>
  <Agency>UN</Agency>
</Meta>
```

The Format element indicates the name of the base standard, for example:

```xml
<Format standard="EDIFACT"/>
```

The following values are valid:

- Fixed
- EDIFACT
- X12
- HL7
- TRADACOMS (this value requires the EDIConfig4.xsd schema or later, see EDI Configuration Files and Schemas)

The Include element in a <Message>.Config file specifies references to any additional configuration files. For example, the code listing below includes a reference to two .segment files and one .codelist file. A .segment file stores all Segment, Composite, and Data Element definitions at a global level (this is to avoid repeating them in each message where they may occur). Likewise, a .codelist file defines validation rules applicable to each field, at a global level.

```xml
<Include href="Admin.Segment"/>
<Include href="EDSD.Segment"/>
<Include href="UNCL.Codelist"/>
```

Any file referenced by the Include element must exist in the same directory as the <Message>.Config file.
The Message element in each <Message>.Config file defines the structure of the message (that is, the hierarchy of groups, segments, and data elements), as well as any code values used for validation (if applicable). For more information about the structure of a message, see Changing the Message Structure. For examples of altering existing messages, see the following topics:

- Customizing an EDIFACT Message
- Customizing an ANSI X12 Transaction
- Defining Code Values at Message Level (Inline)

Modifying the EDI.Collection file
If you add a custom <Message>.Config file, a Message element must also be added to the EDI.Collection file. Each Message element must have the following attributes:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Specifies the official code of the message type (for example &quot;BALANC&quot;, &quot;999&quot;).</td>
</tr>
<tr>
<td>File</td>
<td>Specifies the file name containing the specification of this particular message type. The actual file must exist in the same directory.</td>
</tr>
<tr>
<td>Description</td>
<td>Provides a description of this message type as it will appear in the MapForce graphical user interface.</td>
</tr>
</tbody>
</table>

For samples of valid EDI.Collection files, check any of the MapForce built-in EDI collections in the C:\Program Files\Altova\MapForce2019\MapForceEDI directory.

7.5.14.6 Changing the Message Structure

In each <Message>.config file, the structure of a Message element is determined by the XML schema of the EDI standard that you are customizing. This schema is called EDIConfig.xsd (or a similar name, see EDI Configuration Schemas). To build or change the structure of the message, you will therefore need to consult both the documentation of your custom EDI standard and the XML schema.

How to add segments, data elements, and composites
To add data elements, segments, composites, and subcomposites, create <Data>, <Segment>, <Composite>, and <SubComposite> elements as follows:

- The .segment file must contain the list of all reusable, globally-defined segments and data elements.
- The .config file corresponding to each message (or transaction) type must contain the actual structure of that message (the hierarchy of segments, composites, and data elements).

This makes it possible to define a data element once (in the .segment file) and reuse it in multiple messages (in .config files).

To refer to a data element, use the ref attribute. For example, the code listing below shows how data element F100 is defined in the X12.Segment file of the HIPAA.X12 collection.

```xml
<Data name="F100" type="string" maxLength="3" minLength="3" info="Currency"/>
```
X12.Segment file

All segments, in all transaction types that need this data element, can now refer to it. The code listing below shows how the "CUR" segment of the "Health Care: Professional" (837P) transaction refers to the F100 data element previously defined:

```
<Include href="X12.Segment" />
...
<Segment name="CUR" minOccurs="0" info="Foreign Currency Information">
  <Data ref="F100" info="Currency Code" />
</Segment>
```

837P.Config file (X12)

A reference can be in the same file, or in an external file. In the code listing above, the .config file includes a reference to an external .segment file. For this reason, an `<Include href="X12.Segment" />` statement is present at the beginning of the 837P.Config file to add the reference to the external X12.Segment file.

For another example that shows how to add a custom segment, see Example: Adding the TA1 Segment.

How to create groups and loops

Creating or changing EDI structures often requires grouping of segments or data elements under some parent node (a group). Even the message on the EDI component is a group, since it groups together all the multiple segments that this message consists of. Groups may be nested (they can have child groups). Also, they may be repeated multiple times (in which case, they become loops). Groups and loops can be created by adding a `<Group>` element. Use the `maxOccurs` attribute to set the number of allowed occurrences of a loop. For example, the following code listing shows a loop that can occur up to 25 times, is optional, and consists of two segments:

```
<Group name="LoopSAC" maxOccurs="25" minOccurs="0">
  <Segment ref="SAC" />
  <Segment ref="CUR" minOccurs="0" />
</Group>
```

850.Config file (X12)

How to specify the type and size of data elements

Use the `type` attribute of a data element or segment to specify its type (string, decimal, etc.). The possible data types (the `type` attribute) are generally any of XML Schema types that are used in the configuration files delivered by default. Other XML Schema simple types can tentatively be used, but cannot be guaranteed. The following simple types are not supported: "anyType", "ENTITIES" and "QName".
Use the `minLength` and `maxLength` attributes of a data element to specify its minimum and maximum length.

**How to specify mandatory and optional data elements**

To specify that a data element is mandatory, do not set the value of the `minOccurs` attribute (this attribute has "1" as default value, which means the field is mandatory by default). To specify that a data element is optional, set the `minOccurs` attribute to "0". For example, the code listing below defines two mandatory data elements (F1331, F1332) and one optional data element (F1325).

```xml
<Composite name="C023" info="Health Care Service Location Information">
  <Data ref="F1331"/>
  <Data ref="F1332"/>
  <Data ref="F1325" minOccurs="0"/>
</Composite>
```

### 7.5.14.7 Handling Multiple Consecutive Data Elements

EDI segments often allow the use of multiple consecutive elements of the same name. For example, this is applicable to the N2 segment of the "850 - PURCHASE ORDER" transaction set of the X12 standard. According to the specification, the N2 segment allows for two consecutive alphanumeric "Name" fields (one mandatory and one optional):

<table>
<thead>
<tr>
<th>REF ELE ID</th>
<th>NAME</th>
<th>RPT ATTRIBUTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 93</td>
<td>Name</td>
<td>M AN 1/60</td>
</tr>
<tr>
<td>02 93</td>
<td>Name</td>
<td>O AN 1/60</td>
</tr>
</tbody>
</table>

When you create or modify an EDI configuration, there are two possibilities to handle such data elements, as shown below.

**Approach 1: Create separate mapping nodes for each consecutive data element**

In this approach, you create multiple occurrences of the data element in the segment file. In this case, a new node will appear on the mapping for each occurrence of the data element. Consider this configuration:

```xml
<Segment name="N2" info="Additional Name Information">
  <Data ref="F93" nodeName="F93_1"/>
  <Data ref="F93" minOccurs="0" nodeName="F93_2"/>
</Segment>
```

On the mapping, this configuration creates two separate F93 nodes, each corresponding to an occurrence of F93:
Using this approach is suitable when every occurrence of the data element is meaningful by itself and, consequently, you want a separate node to appear for it on the mapping.

**Approach 2: Create a single node for all occurrences of the data element**

In this approach, you treat all occurrences of the data element as one single node. To do this, add the `mergedEntries` attribute to the corresponding data element. In the code listing below, the `mergedEntries` attribute of data element F93 is set to "2". This allows for two consecutive occurrences of this field in the EDI instance file.

```xml
<Segment name="N2" info="Additional Name Information">
  <Data ref="F93" mergedEntries="2"/>
</Segment>
```

On the mapping, the node corresponding to the F93 data element will be displayed as a single node:
Mapping data from a data element which has merged entries will create multiple duplicate elements on the target side, if the source EDI file contains multiple (not empty) occurrences of this element. For example, let's consider a source EDI file where the N2 segment contains two consecutive occurrences of F93. The first occurrence is "Michelle Butler" and the second is "Mrs".

N2+Michelle Butler+Mrs+'

Mapping data from such an EDI file would create, on the target side, the following output (notice the duplicate occurrences of <ContactName>):

<Customer>
  <Number>123</Number>
  <ContactName>Michelle Butler</ContactName>
  <ContactName>Mrs</ContactName>
  <CompanyName>Nanonull, Inc.</CompanyName>
</Customer>

Using this approach is suitable when each occurrence of the data element is not meaningful by itself, and you do not want a separate node to appear on the mapping for each occurrence. For example, a data element with multiple occurrences may store multiple line entries that make up an address. For such cases, the "merged entries" approach might be more useful than having individual nodes for each address line.

Some configuration files supplied by default with MapForce use merged entries, and therefore, only one node appears on the mapping for multiple consecutive occurrences of a data element. If you need separate nodes for each occurrence of the data element, this can be achieved by customizing the EDI configuration. For an example, see Splitting Merged Entries into Separate Nodes.
7.5.14.8 Splitting Merged Entries into Separate Nodes

Some data elements in the EDI configuration files supplied by default with MapForce are configured to display only one node on the mapping, even if the corresponding data element may occur multiple times in the source EDI file (see also Handling Multiple Consecutive Data Elements). An example of such data element is F3036 of the UN/EDIFACT standard. According to the specification, F3036 can occur up to five times consecutively in the C080 (“Party name”) composite (see https://www.unece.org/fileadmin/DAM/trade/untdid/d15b/trcd/trcdc080.htm).

In the default UN/EDIFACT collection of MapForce, the data element F3036 is represented through a single node on the mapping. If you draw a mapping connection from this node to a target node, then the connection would create as many items in the target as there are occurrences of F3036 in the source. (This follows the basic rule of MapForce connections, for each item in the source, create one item in the target). However, there may be situations where this is not the desired result, and you need separate nodes for each occurrence of the data element.

The goal of the example is, therefore, to create a custom UN/EDIFACT configuration which would display separate mapping nodes for each occurrence of F3036. This would enable you to map each individual occurrence of F3036 to a separate target node. The images below show the EDIFACT mapping component before and after customization.

The steps required to split the F3036 data element into multiple mapping nodes are as follows:

1. Create a custom EDI collection using the existing UN/EDIFACT collection as starting point (see Creating a Custom EDI Collection). This ensures your changes will not affect in any way the default EDI configuration.
2. Open the EDSD.Segment file of your custom collection in an XML editor. (If you want to customize a different EDI flavour, then edit the corresponding .segment file of that particular EDI flavour. For example, for X12, this would be the X12.Segment file.)
3. Find the composite C080 and remove the mergedEntries="5" attribute of the F3036 data element.
4. Add the `minOccurs="0"` attribute to all fields except for the first one. The reason is that, according to the UN/EDIFACT standard, only the first occurrence of the field is mandatory; the others are conditional.

5. Enter a unique node name for each of the fields (for example, `F3036_1`, `F3036_2`, and so on). At this stage, the configuration should look as follows:

```xml
<Composite name="C080" info="PARTY NAME">
  <Data ref="F3036" nodeName="F3036_1" minOccurs="0"/>
  <Data ref="F3036" nodeName="F3036_2" minOccurs="0"/>
  <Data ref="F3036" nodeName="F3036_3" minOccurs="0"/>
  <Data ref="F3036" nodeName="F3036_4" minOccurs="0"/>
  <Data ref="F3036" nodeName="F3036_5" minOccurs="0"/>
  <Data ref="F3045" minOccurs="0"/>
</Composite>
```

6. Save the edited `EDSD.Segment` file.

You can now view and work with the modified EDI collection in MapForce, as follows:

1. On the **Insert** menu, click **EDI**.
2. When prompted, select the custom collection created previously (in this example, "EDIFACT.Custom").
3. Select any message type that uses the C080 segment (in this example, "ORDERS - Purchase Order Message").

The EDI component now appears modified in MapForce (each occurrence of the `F3036` data element has a separate node on the mapping).
The technique described in this example can also be used to split merged entries for other data elements, regardless of the EDI flavour.

7.5.14.9 Changing Possible Code Values of Data Elements

The possible values of a data element can be changed by means of configuration files. Similar to other configuration information, you can configure the possible code values at each message level (local customization), or globally (across all messages that may use that particular data element).

Like with any other custom configuration, to customize code values, it is recommended to create a custom EDI collection (see Creating a Custom EDI Collection) and apply all customization to it, rather than altering the default MapForce configuration.

The available customization options depend on the schema version used by your custom EDI collection (see EDI Configuration Schemas). To put it simply, a newer schema means that more advanced configuration options are available. In particular, each schema supports custom code values as follows:

- With EDIConfig.xsd, it is possible to change an existing .codelist file in order to add custom code values at a global level.
- With EDIConfig4.xsd, custom code values can be added not only at global level, but also at message level.
- With EDIConfig5.xsd, all of the options above are available, including the ability to define advanced configuration rules at character level (see Configuring Character-Level Validation). Also, data elements can be declared as having incomplete validation (see Relaxing Field Validation).

Global customization

To add custom code values for a data element globally, edit the .codelist file of your custom EDI collection. First, find the data element (field) whose values you need to change, and then add all the required custom values as new <Value> elements, as shown below:
As shown in the code listing above, each `<Value>` element has a `Code` attribute which specifies the actual value code expected for this data element. The `<Title>` and `<Description>` elements provide additional descriptive information about this value code.

The code listing above also has an `Include` statement which instructs MapForce to look for additional code values defined in an external file (`ISO6346.Codelist`) which uses the same schema as the `UNCL.Codelist` file. Use this technique if you need to adapt various external files containing custom code values.

For an example, see [Adapting Code Values from External Sources](#).

**Local customization**

To add custom code values at message level, you will need to edit the .config file corresponding to the message type. For more information, see [Defining Code Values at Message Level (Inline)](#).

### 7.5.14.10 Example: Adding a TA1 Segment

The TA1 Segment is an optional segment and is used to acknowledge the reception of the interchange and the syntactical correctness of the envelope segments within it. The segment can be added to EDI X12 and EDI HIPAA components in MapForce by modifying the configuration files.

**To include the TA1 segment in an EDI component:**

1. Create a custom EDI collection using the HIPAAA.X12 collection as a model (see [Creating a Custom EDI Collection](#)). This ensures that your changes do not affect the original default configuration.
2. Open in an XML editor the **Envelope.config** file of your custom EDI collection.
3. Paste the text `<Segment ref="TA1" minOccurs="0" maxOccurs="unbounded"/>

between the ISA and Group segments, and save the file. The file should now look as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../EDIConfig.xsd" Version="3">
  <Meta>
    <Release>6020</Release>
    <Agency>X12</Agency>
  </Meta>
  <Format standard="X12"/>
  <Include href="X12.Segment"/>
  <Include href="X12.Codelist"/>
  <Include collection="EDI.Collection"/>
  <Group name="Envelope">
    <Group name="Interchange" maxOccurs="unbounded">
      <Segment ref="ISA" minOccurs="0"/>
      <Segment ref="TA1" minOccurs="0" maxOccurs="unbounded"/>
      <Group name="Group" maxOccurs="unbounded">
        ...
      </Group>
      <Segment ref="IEA" minOccurs="0"/>
    </Group>
  </Group>
</Config>
```

The line highlighted above adds a reference to the TA1 segment which is already defined in the **X12.Segment** file. You can now view the structure of the TA1 segment in MapForce as follows:

1. On the **Insert** menu, click **EDI**.
2. On the "Browse EDI Collections" dialog box, select the custom collection created above and a message type (for example, "850 - Purchase Order").

Notice that the customized component now includes the TA1 segment. You can now draw mapping connections to and from this customized component as required.
This example shows you how to customize EDIFACT messages so that MapForce can process non-standard or changed EDIFACT formats. A sample mapping which processes a custom EDIFACT file with a slightly extended CTA (Contact Information) segment of the ORDERS message is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\Orders-Custom-EDI.mfd`. Note that, if you open this mapping before following the customization instructions below, MapForce does not validate it successfully. The reason is that the mapping references a custom EDIFACT configuration which does not exist yet on your computer (it will be created in this example).

The example mapping reads data from the `Orders-Custom.EDI` file, available in the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\` directory. If you open this file with a text editor, notice that it uses a customized CTA segment. Specifically, line 9 contains a Mr entry, and line 11 contains a Mrs entry:
Before you can map data from custom EDI files such as the one above, a custom EDIFACT configuration must be created. In this example, you will customize the default EDIFACT configuration so as to extend the CTA segment with a new field which would accommodate the title of the person. For your convenience, the final result of the customization procedure is available in the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\` directory, as a ZIP archive (EDIFACT.Nanonull.zip). The instructions below assume that you have not already unpacked the ZIP file to the `..\Program Files\MapForceEDI\` directory.

Setting up the customization files

1. Open the MapForce installation directory (`C:\Program Files\Altova\MapForce2019\`). When running 32-bit MapForce on 64-bit operating systems, adjust the path to `C:\Program Files (x86)\Altova\MapForce2019\MapForceEDI`.
2. Under the `..\MapForceEDI\` directory, create a directory called `EDIFACT.Nanonull`.
3. Copy the following files from the `..\MapForceEDI\EDIFACT` folder into the `EDIFACT.Nanonull` folder.

- EDI.Collection
- Envelope.Config
- UNCL.Codelist
- Admin.Segment
- EDSD.Segment (Electronic Data Segment Definition)
- ORDERS.Config

For information about the purpose of each file, see EDI Configuration Files.

Note: Change the attributes of the files to read-write to make them editable.

Configuring the EDI.Collection file

1. Open `EDI.Collection` file in an XML editor (for example, XMLSpy).
2. Remove all "Message" elements, except for the "ORDERS" message. Make sure you retain the `<Messages>` tags, however.

```
<?xml version="1.0" encoding="UTF-8"?>
.Messages Version="3">
<Meta>
```
3. Save the file.

The new **EDI.Collection** file should now be accessible to MapForce. You can test this as follows:

1. Start MapForce.
2. On the **Insert** menu, click **EDI**. Alternatively, click the **Insert EDI** toolbar button. A dialog box such as the one below opens, displaying a new collection named "EDIFACT.Nanonull". When selected, the collection shows only one message type: "ORDERS".

![Browse EDI collections](image)

**Note:** At startup, MapForce scans all sibling subfolders under the ...\MapforceEDI\ directory and looks for a file called "EDI.Collection". Each folder name containing an **EDI.Collection** file appears in the dialog box. The Message types list shows the content of the collection file, which in this example contains only an ORDERS message.

**Global versus inline customization**

The goal of the example is to redefine the CTA (Contact Information) segment. The CTA segment consists of one field (F3139) and one composite (C056). To store the title data (for example, "Mr" or "Mrs"), we will add to the C056 composite a new field, called X1000. There are several ways to perform the customization:

- **Globally**, by customizing the **EDSD.segment** file. All segments, in all messages that use composite C056, will contain/reference the new element.
- **Inline**, by customizing the **ORDERS.Config** file. Only the customized segment (CTA) in the current message will contain the new element.

**Global customization**

To make access to the new X1000 field **global**, changes have to be made only to the **EDSD.Segment** file. All segments, in all messages that use composite C056, will contain/reference the new element.
To redefine the Composite C056 globally:

1. Open the EDSD.Segment file in a text or XML editor, and navigate to Config | Elements | Composite | C056.

```
<Composite name="C056" info="CONTACT DETAILS">
    <Data ref="F3413" minOccurs="0"/>
    <Data ref="F3412" minOccurs="0"/>
</Composite>
```

2. Insert the following line in the C056 segment, under "F3412":

```
<Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
```

The Composite definition should now look as shown below:

```
<Composite name="C056" info="CONTACT DETAILS">
    <Data ref="F3413" minOccurs="0"/>
    <Data ref="F3412" minOccurs="0"/>
    <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
</Composite>
```

In the code listing above, the new X1000 field is defined using the "name" attribute as opposed to other fields of the segment which use the "ref" attribute. The two other fields are defined at the beginning of the file and are only referenced here.

The new X1000 field is now available to all messages that use composite C056. You can preview the new field in MapForce as follows:

1. On the Insert menu, click EDI. Alternatively, click the Insert EDI toolbar button.
2. Click the "EDIFACT.Nanonull" collection, and select the ORDERS message.
3. When prompted to select a source EDI file, click Skip. The ORDERS component is now visible in the mapping window.
4. Click the component header, press Ctrl + F and enter "X1000" as search text. Click Find Next to jump to the next occurrence of the new X1000 element (it should be under Envelope/Interchange/Group/Message_ORDERS/SG2/SG5/CTA/C056).
Inline customization

To make access to the new X1000 field *local (or inline)*, changes have to be made only to the `ORDERS.config` file. In this case, only the redefined CTA segment in the current message will contain/reference the new element. In other words, the CTA segment is redefined locally to contain a redefined Composite C056, with a custom new field X1000.

To redefine the Composite C056 locally:

1. Open the `ORDERS.Config` file in a text or XML editor and navigate to `Group name="SG5"` (or search for `SG5`).

   ```xml
   <Group name="SG5" minOccurs="0" maxOccurs="5" info="CTA - Contact information">
     <Segment ref="CTA"/>
     <Segment ref="COM" minOccurs="0" maxOccurs="5"/>
   </Group>
   ```

2. Replace the line `<Segment ref="CTA"/>` with the following lines:

   ```xml
   <Segment name="CTA" id="CTA_ORDERS_SG5" info="CONTACT INFORMATION">
     <Data ref="F3139" minOccurs="0"/>
     <Composite name="C056" minOccurs="0" info="DEPARTMENT OR EMPLOYEE DETAILS">
       <Data ref="F3413" minOccurs="0"/>
       <Data ref="F3412" minOccurs="0"/>
       <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
     </Composite>
   </Segment>
   ```

The `Group` definition should now look as shown below:
You can preview the new field in MapForce as follows:

1. On the **Insert** menu, click **EDI**. Alternatively, click the **Insert EDI** toolbar button.
2. Click the “EDIFACT.Nanonull” collection, and select the ORDERS message.
3. When prompted to select a source EDI file, click **Skip**. The ORDERS component is now visible in the mapping window.
4. Navigate to **Envelope/Interchange/Group/Message_ORDERS/SG2/SG5/CTA/C056**, to see the new X1000 element.

### Using the customized message in mappings

If you have followed all the instructions above, you can now use the custom EDIFACT configuration in your mappings. An example mapping which uses the custom EDIFACT configuration created above is **Orders-Custom-EDI.mfd**, available in the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial` folder. The example maps the `ORDERS- Custom.EDI` file to the `Order-EDI` schema. The custom field that was added to the EDI structure, X1000, maps to the Salutation item in the target schema.
If you have followed the instructions above, you can open this mapping and run it successfully. However, if you have not followed the customization instructions, you can run this mapping as follows:

1. Create a new directory called "EDIFACT.Nanonull" under "C:\Program Files\Altova\MapForce2019\MapForceEDI\" directory. When running 32-bit MapForce on 64-bit operating systems, adjust the path to C:\Program Files (x86)\Altova\MapForce2019\MapForceEDI.
2. Unzip the EDIFACT.Nanonull.zip file (located in <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial) into the new directory.
3. Open Orders-Custom-EDI.mfd.

To preview the mapping result, click the Output tab.
Similar to other mapping design files in MapForce, EDI mappings can be executed outside MapForce as well. For example, you can deploy them to another server for execution with MapForce Server or FlowForce Server, or generate program code to run the mapping. For the scope of this example, if you choose to run the mapping by generating C++ code, note that a class named "CX1000Type" is generated, which is accessible from the "CC056Type" class.

### 7.5.14.12 Example: Customizing an ANSI X12 transaction

This example illustrates how to customize MapForce so that it can process non-standard or changed X12 formats. The example is accompanied by a sample mapping which is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\Orders-Custom-X12.mfd`. Note that, if you open this mapping before following the customization instructions below, MapForce does not validate it successfully. The reason is that the mapping references a custom X12 collection which does not exist yet on your computer (it will be created in this example).

The example mapping reads data from the `Orders-Custom.X12` file, available in the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\` directory. If you open this file with a text editor, notice that it uses a customized N2 segment. Specifically, line 6 contains an additional `++Mrs` entry:
Before you can map data from custom X12 files such as the one above, a custom X12 configuration must be created. In this example, you will customize the default X12 configuration so as to extend the N2 segment with a new data element which would accommodate the title of the person. For your convenience, the final result of the customization procedure is available in the <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ directory, as a ZIP archive (X12.Nanonull.zip). The instructions below assume that you have not unpacked the ZIP file to the C:\Program Files\Altova\MapForce2019\MapForceEDI directory.

Setting up the customization files

1. Open the MapForce installation directory (C:\Program Files\Altova\MapForce2019\). When running 32-bit MapForce on 64-bit operating systems, adjust the path to C:\Program Files (x86)\Altova\MapForce2019\MapForceEDI.
2. Under the ..\MapForceEDI\ directory, create a directory called X12.Nanonull.
3. Copy the following files from the ..\MapForceEDI\X12 folder into the X12.Nanonull folder.
   - EDI.Collection
   - Envelope.Config
   - 850.Config
   - X12.Segment
   - X12.Codelist

For information about the purpose of each file, see EDI Configuration Files.

Note: Change the attributes of the files to read-write to make them editable.

Configuring the EDI.Collection file

1. Open EDI.Collection file in an XML editor (for example, XMLSpy).
2. Remove all "Message" elements, except for the "Purchase Order" message. Make sure you retain the <Messages> tags, however.

```
<?xml version="1.0" encoding="UTF-8"?>
<Messages Version="3">
  <Meta>
    <Release>6040</Release>
  </Meta>
```

```
1 ISA+00+
2 GS+PO+SenderID+ReceiverID+20060308+182347+1+UN+050112'
3 ST+850+12345'
4 ED+00+SA+ABC123456XYZ12++20040430'
5 N1+1+NaNonull, Inc.+1123'
6 N2+Michelle Butler+Mrs'
7 N3+111 Oakstreet Suite 4876'
8 N4+Verenon+CA+29123'
9 PO+1++1+++7.2''
10 PO3+1+++7.2++1+US''
11 PID+1+++Pizza Pepperoni''
12 TXI+1++9''
13 AMT+14720''
14 PO1++2++13.2''
15 PO3+1+++6.6++2+US''
```
3. Save the file.

The new **EDI.Collection** file should now be accessible to MapForce. You can test this as follows:

1. Start MapForce.

2. On the **Insert** menu, click **EDI**. Alternatively, click the **Insert EDI** toolbar button. A dialog box such as the one below opens, displaying a new collection named "X12.Nanonull". When selected, the collection shows only one message type: "Purchase Order".

![Browse EDI collections](image)

**Note:** At startup, MapForce scans all sibling subfolders under the ...\MapforceEDI\ directory and looks for a file called "EDI.Collection". Each folder name containing an **EDI.Collection** file appears in the dialog box. The Message types list shows the content of the collection file, which in this example contains only a "Purchase Order" message.

**Global versus inline customization**

The goal of this example is to redefine the N2 "Additional Name Information" segment so that it contains a custom X1000 data element which would store a person's title (for example, "Mrs"). Before customization, N2 consists of one data element only, "F93 Name". There are several ways the customization can be achieved:

- **Globally**, by customizing the **X12.Segment** file. "Global" means that all segments, in all transactions that use N2, will contain/reference the new element.
- **Inline**, by customizing the **850.Config** file. "Local" means that only the customized segment (N2) in the current transaction will contain the new element.
Global customization
To make access to the new X1000 field global, changes have to be made only to the X12.Segment file. All segments, in all transactions that use N2, will contain/reference the new element.

To redefine the N2 segment globally:

1. Open the X12.Segment file in an XML editor and navigate to Config | Elements | Segment name="N2".

   ```xml
   <Segment name="N2" info="Additional Name Information">
   <Data ref="F93" mergedEntries="2"/>
   </Segment>
   ```

2. Insert the following line under F93, and save the file.

   ```xml
   <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
   ```

The segment definition should now look as shown below:

```xml
<Segment name="N2" info="Additional Name Information">
   <Data ref="F93" mergedEntries="2"/>
   <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
 </Segment>
```

Notes:
- The new X1000 field is defined using the name attribute as opposed to other fields of the segment which use the ref attribute. The F93 field is defined at the beginning of the X12.Segment file and is only referenced here.
- The mergedEntries attribute of F93 specifies that there can be two consecutive occurrences of this element with the same name. This reflects the definition of this field as it is defined in the ASC X12 specification. For this reason, in the source EDI file that is being customized here, the N1 segment looks as follows:

   ```
   N2+Michelle Butler++Mrs'
   ```

   In the line above, notice the two consecutive "+" separators between "Michelle Butler" and "Mrs". The explanation is that the N2 segment consists of a "F93" data element ("Michelle Butler"), followed by a "+" separator, followed by the second "F93" element (which is empty), followed by a "+" separator, followed by the custom "X1000" data element ("Mrs"). For more information about the mergedEntries attribute, see Handling Multiple Consecutive Data Elements.

The new X1000 field is now available to all transactions that use segment N2. You can preview the new field in MapForce as follows:

1. On the Insert menu, click EDI. Alternatively, click the Insert EDI toolbar button.
2. Click the "X12.Nanonull" collection, and select the "Purchase Order" message.
3. When prompted to select a source EDI file, click Skip. The 850 component is now visible in the mapping window.
4. Click the component header, press Ctrl + F and enter "X1000" as search text. Click Find Next to jump to the next occurrence of the new X1000 element (it should be under Envelope/Interchange/Group/Message_850/Loop_N1/N2).

**Local customization**

If you want to make access to the new X1000 field local, then only the 850.Config file must be edited. In this case, only the segment N2 in the "850 - Purchase Order" transaction will contain reference the new X1000 field.

To redefine the N2 segment locally:

1. Open the 850.Config file in a text or XML editor and navigate to Group name="LoopN1" (or search for LoopN1).

   `<Group name="LoopN1" maxOccurs="10" minOccurs="0">
   <Segment ref="N1"/>
   <Segment ref="N2" minOccurs="0" maxOccurs="2"/>
   ...
   </Group>`

2. Replace the line `<Segment ref="N2" minOccurs="0" maxOccurs="2"/>` with the following lines:

   `<Segment name="N2" info="Additional Name Information">
   <Data ref="F93" mergedEntries="2"/>
   <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
   </Segment>`
The **Group** definition should now look as shown below:

```xml
<Group name="LoopN1" maxOccurs="10" minOccurs="0">
  <Segment ref="N1"/>
  <Segment name="N2" info="Additional Name Information">
    <Data ref="F93" mergedEntries="2"/>
    <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
  </Segment>
  ...
</Group>
```

You can now preview the new field in MapForce, using the same instructions as those described above for global customization.

Using the customized message in mappings

If you have followed all the instructions above, you can now use the custom X12 configuration in your mappings. An example mapping which uses the custom EDIFACT configuration created above is `Orders-Custom-X12.mfd`, available in the `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial` folder. The example maps the `ORDERS-Custom.X12` file to the `Order-EDI.xsd` schema. The custom `X1000` field that was added in the steps above maps to the `Salutation` item in the target schema.
If you have followed the instructions above, you can open this mapping and run it successfully. To preview the mapping result, click the **Output** tab. As expected, the Salutation target item is populated with the value "Mrs" extracted from the custom X1000 data element of the X12 file.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="">
  <Header>
    <Number>A9C123456XYZ</Number>
    <Received>2004-04-30T17:42:00</Received>
  </Header>
  <Customer>
    <Number>123</Number>
    <ContactName>Michelle Butler</ContactName>
    <Salutation>Mrs</Salutation>
    <CompanyName>Nanonull, Inc.</CompanyName>
    <Address>
      <Street>119 Oakstreet Suite 4376</Street>
      <City>Veren0</City>
      <ZIP>29213</ZIP>
      <State>CA</State>
    </Address>
  </Customer>
</Order>
```
7.5.15 Customizing EDI Validation

This section illustrates how validation of EDI messages can be customized in MapForce so as to achieve the following goals:

- Relax validation of fields where validation rules are not completely enforced by the standard and may be extended by means of external standards.

  For example, there are EDI data elements that may accept code values specified in external ISO standards or UN/ECE recommendations, in addition to predefined code values. In this case, when processing such EDI fields, you can configure MapForce to return a warning (not a validation error) when an external code is encountered, and resume the mapping process. For further information, see Relaxing Field Validation.

- Perform full validation of fields such as the ones described above, by adapting custom configuration files to MapForce.

  For example, to validate the Data Element 4405 of the UN/EDIFACT D.16A standard, you may add a custom configuration file which contains the validation rules as defined by the UN/ECE Recommendation 24 (“Trade and Transport Status Codes”. The code values defined in UN/ECE Recommendation 24 are external to EDIFACT standard (and thus are not part of the default set of EDIFACT configuration files supplied with MapForce). For further information, see Adapting Code Values from External Sources.

- Perform validation globally for all messages where a field is used or locally for a particular message.

  It is possible to configure validation to apply globally or locally. Global validation affects all occurrences of the field in all messages. Local (or "inline") validation applies only to a specific message. For further information, see Defining Custom Code Values at Message Level (Inline).

- Define advanced validation rules for fields that require validation not only at field level, but also at character level.

  Certain messages, such as the VERMAS (Verification of Mass) message of UN/EDIFACT D.16A, contain fields that require advanced validation rules (such as Data Element 8155). This field may accept code values specified in ISO 6346. To validate field such as ISO 6346, MapForce can be customized, by means of configuration files, to check if specific characters occur at specific positions within the field. For further information, see Configuring Character-Level Validation. (Note that ISO 6346 is mentioned here only as an example, the MapForce configuration files already include validation rules for ISO 6346.)

7.5.15.1 Relaxing Field Validation

There are cases when a standard does not completely enforce the code values that a field may have, allowing code values defined by external standards or recommendations. Taking EDIFACT D16A as example, several fields allow code values defined in external ISO standards, or in UN/ECE recommendations. For example, Data Element 4405 defines 146 code values, but also
allows use of external UN/ECE Recommendation 24, which specifies 346 code values.

In MapForce, the possible code values for a field are validated against a configuration file having the .codelist extension (hereafter referred to as the "codelist" file). When MapForce processes an EDI file, it normally produces a validation error if a field does not have any of the code values specified in the codelist file for that EDI flavour. (This assumes you have not changed the default validation options so as to silently ignore invalid field values, see EDI component validation.) The codelist files available for each EDI flavour are in the C:\Program Files\Altova\MapForce2019\MapForceEDI\ directory. For example, the UNCL.codelist file available in C:\Program Files\Altova\MapForce2019\MapForceEDI\EDIFACT\ stores possible code values for fields in UN/EDIFACT fields.

The default .codelist file includes all possible code values for a field, according to the respective EDI standard. When code values are not explicitly defined by the standard but enforced by external standards, validation is considered to be incomplete. Namely, when the EDI standard does not fully enforce the list of code values for a particular field (such as Data Element 4405 of UN/EDIFACT D16A), processing the field during the mapping execution would normally raise validation errors. Specifically, MapForce (or the execution engine running the EDI mapping, be it MapForce Server, or a C#, C++, or Java program) reports an error in the format: {Value} is not a legal value for field {Field}. This error means that the field value was not found in the list of existing code list values defined in the .codelist file.

To prevent the mapping process from being stopped when errors such as above are encountered, you can do the following:

1. Change the validation behavior from EDI Validation Settings (for example, select the check box Report & Accept next to "Invalid code list value" on the EDI Validation Settings dialog box. (To configure EDI validation settings, click the Validation button on the EDI Component Settings dialog box.)

2. Relax field validation from configuration files, as shown below.

MapForce provides a mechanism by which a field can be marked as having "incomplete"
validation rules. This ensures that MapForce (or the execution engine running the EDI mapping, be it MapForce Server, or a C#, C++, or Java program) does not produce an error message when the field is processed; instead, a warning is displayed, and the mapping process resumes. In other words, "incomplete" validation means that only the code values defined in the MapForce default .codelist file should be considered for validation, not those defined externally.

To allow for flexibility without adding too much complexity, the default .codelist configuration files supplied with MapForce treat certain fields as having incomplete validation.

By default, MapForce produces warnings (not errors) if any of the following fields contain code values that are defined in external specifications:

- Data Element 4053 of UN/EDIFACT D.16A
- Data Element 4277 of UN/EDIFACT D.16A
- Data Element 4405 of UN/EDIFACT D.16A
- Data Element 8179 of UN/EDIFACT D.16A

The warning is produced so as to make you aware that validation of this field is incomplete. Incomplete validation may result in incorrect data being produced by the mapping. To prevent this from happening, it is recommended to perform full field validation, by means of custom .codelist files to MapForce. Custom .codelist files let you define external code values in a format understood by MapForce and perform full validation of the field. For more information, see Adapting Code Values from External Sources.

Note: Data Element 8155 also relies for validation on an external standard (the ISO 6346 standard); however, for this field MapForce provides full validation by means of a separate .codelist file, see Adapting Code Values from External Sources.

If necessary, you can additionally customize other fields to be treated as having incomplete validation rules. To instruct MapForce that a field has incomplete validation rules, do the following:

1. Locate the .codelist file in the C:\Program Files\Altova\MapForce2019\MapForceEDI\ directory (subdirectories exist for each EDI flavour), and open it with an XML editor such as XMLSpy.
2. Find the <Field> element corresponding to the required data element, and add the attribute Incomplete="1" to it (highlighted below), for example:

```xml
<Field Name="F8179" Length="8" Type="Alphanumeric" Incomplete="1">
  <Title>Transport means description code</Title>
  <Description>Code specifying the means of transport.</Description>
  <Note>1 Use UN/ECE recommendation 28.</Note>
  <Values>
    <Value Code="1">
      <Title>Box tricycle</Title>
      <Description>Tricycle with a basket for delivering goods.</Description>
    </Value>
  </Values>
</Field>
```
Note: You must have write permissions to the C:\Program Files\Altova\MapForce2019 \MapForceEDI directory in order to modify its contents.

If your processing requirements demand that an error should be raised instead of a warning for any fields that were already marked as "incomplete", change the attribute Incomplete to value "0" instead of "1". This way, field validation will be enforced strictly, and errors (not warnings) will occur if the field value is not part of the set of values defined in the .codelist file.

7.5.15.2 Adapting Code Values from External Sources

In the EDI configuration files supplied by default with MapForce, some data elements are configured to have incomplete validation, as explained in Relaxing Field Validation. This is due to the fact that the possible code values for these data elements rely on specifications which are external to the EDI standard itself.

When incomplete (or relaxed) validation of a field is not acceptable or desirable, it is possible to configure MapForce to perform full field validation. This approach requires adding custom .codelist files in addition to the default ones already available in MapForce. The role of the custom .codelist files is to supply any code values that are not defined by default in the EDI standard and consequently are not present in MapForce configuration files. This usually applies to code values that are defined in external standards or recommendations.

Note that the custom .codelist file will require an entry for every possible code value that is to be validated (which may be hundreds of records). An example is the Data Element 4405 of the UN/EDIFACT D16A. In addition to the code values defined in the UN/EDIFACT standard, this field also allows code values that are specified in the UN/ECE Recommendation 24 ("Trade and Transport Status Codes", see https://www.unece.org/fileadmin/DAM/cefact/recommendations/rec24/rec24_ecetrd258e.pdf) which amount to 346 codes. In order to accommodate all these external code values, a custom .codelist file must be created which contains an entry for each of the 346 codes present in the recommendation. Therefore, if that is acceptable by your particular business case, using incomplete validation is a simpler solution (see Relaxing Field Validation). Be aware that incomplete validation involves the risk of allowing invalid data to be produced in the mapping output, so full field validation is recommended instead.

To perform full field validation of fields whose code values are defined in an external standard, take the following steps:

1. Create a copy of the default .codelist file supplied with MapForce. The .codelist files available for each EDI flavour are in the C:\Program Files\Altova\MapForce2019 \MapForceEDI\ directory. For example, if adding a custom .codelist file to validate UN/EDIFACT messages, use as template the UNCL.Codelist file available in C:\Program Files\Altova\MapForce2019\MapForceEDI\EDIFACT\.

Note: You must have write permissions to the C:\Program Files\Altova\MapForce2019 \MapForceEDI\ directory in order to modify its contents.

2. Rename the newly created .codelist file to a short and descriptive filename, since the file will be referred from other MapForce configuration files. For the scope of this example, the custom .codelist file will be called UNECE24.codelist.

3. Using an XML editor such as XMLSpy, delete from the new file all content included
between the `<Codelist>` and `</Codelist>` tags. The file should now look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    Version="5" xsi:noNamespaceSchemaLocation="../EDIConfig5.xsd">
    <Codelist>
        ...
    </Codelist>
</Config>
```

4. Add a new `<Field>` structure under `<Codelist>`, as shown below. In this example, we assume to be adding external code values for Data Element 4405 of the UN/EDIFACT. Otherwise, replace the field attributes as required by your particular case. Remember that you can find the correct name, length and type of each field in the default .codelist file supplied by MapForce (in this example, `UNCL.Codelist`).

```
<?xml version="1.0" encoding="UTF-8"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    Version="5" xsi:noNamespaceSchemaLocation="../EDIConfig5.xsd">
    <Codelist>
        <Field Name="F4405" Length="3" Type="Alphanumeric">
            <Title>Trade and transport status codes</Title>
            <Description>Code specifying a status.</Description>
            <Values>
                ...
            </Values>
        </Field>
    </Codelist>
</Config>
```

5. Under `<Values>`, add a new `<Value>` for each external code, as shown below. In this example, we are adding the first two codes defined in the UN/ECE Recommendation 24 ("Trade and Transport Status Codes", see [https://www.unece.org/fileadmin/DAM/cefact/recommendations/rec24/rec24_ectrd258e.pdf](https://www.unece.org/fileadmin/DAM/cefact/recommendations/rec24/rec24_ectrd258e.pdf)).

```
<?xml version="1.0" encoding="UTF-8"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    Version="5" xsi:noNamespaceSchemaLocation="../EDIConfig5.xsd">
    <Codelist>
        <Field Name="F4405" Length="3" Type="Alphanumeric">
            <Title>Trade and transport status codes</Title>
            <Description>Code specifying a status.</Description>
            <Values>
                <Value Code="1">
                    <Title>Arrival, completed</Title>
                    <Description>The goods/consignment/equipment/means of transport has arrived.</Description>
                </Value>
                ...
            </Values>
        </Field>
    </Codelist>
</Config>
```
At this stage, the custom .codelist file is ready but not yet bound to MapForce. Just be aware that, for the scope of the example, we have only added two external codes ("1" and "2"); however, in a real-life scenario the custom .codelist file is expected to include all code values specified by the external standard.

Finally, before the custom .codelist file can be recognized by MapForce, the following must also be done:

1. Since now the Data Element 4405 does no longer have incomplete validation, MapForce must be instructed to apply full validation. Namely, the Incomplete="1" attribute must be modified in the UNCL.codelist file for this field (the attribute must either be removed, or set to Incomplete="0"). Therefore, modify the UNCL.codelist file as shown below:

   ```xml
   ... 
   <Field Name="F4405" Length="3" Type="Alphanumeric" Incomplete="0">
      <Title>Status description code</Title>
      <Description>Code specifying a status.</Description>
      <Note>1 For transport status, use UN/ECE Recommendation 24.</Note>
   </Field>
   ... 
   ```

2. Modify the UNCL.codelist file to add a reference to the newly created UNECE24.codelist file, as shown below:

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
Version="5" xsi:noNamespaceSchemaLocation="../EDIConfig5.xsd">
   <Include href="UNECE24.codelist"/>
   </Codelist>
   ```

You have now finished adapting MapForce configuration files to validate Field 4405 from code values defined in a custom file. This approach can also be used for other EDI flavours to achieve full field validation based on some external standard.
Although it is possible to define external code lists by editing the existing .codelist file supplied with MapForce (in this example, UNCL.codelist), this is not recommended. A good practice is to always define external code lists in custom files and keep changes to the default MapForce configuration files to minimum; this will make it easier to migrate to new versions of EDI formats in future.

Note that the instructions above cause the field validation to be global, for any segment that may be using Field 4405. In some cases, you may want the custom code lists to apply only to certain messages (in other words, the configuration should not apply globally for all messages, but inline, or locally, for a particular message only). For more information about using the "inline" approach, see Defining Code Values at Message Level (Inline).

7.5.15.3 Defining Code Values at Message Level (Inline)

In a previous topic, Adapting Code Values from External Sources, you have seen how to configure field validation through custom code lists globally (it applies for any message that may be using a particular field). In the event that you need validation to apply only to a specific message, it is possible to add custom code lists inline (locally for a particular message). When you need to do this, follow the steps below:

1. Open in an XML editor the {Message}.config file corresponding to the message of interest. The {Message}.config file can be found in the C:\Program Files\Altova\MapForce2019\MapForceEDI\ directory, in the subdirectory corresponding to the name of the required EDI flavour.
2. Find the <Data> element under the required <Segment>, and add the custom code values as children of <Values>, for example:

```
<Group name="Message_270-B1" info="Health Care Eligibility Benefit Inquiry" minOccurs="unbounded">
    <Segment name="ST" info="Transaction Set Header">
        <Condition path="F1705" value="005010X279A1" />
        <Data ref="F143" info="Transaction Set Identifier Code">
            <Values>
                <Value Code="270" />
            </Values>
        </Data>
        <Data ref="F329" info="Transaction Set Control Number" />
        <Data ref="F1705" info="Implementation Convention Reference" />
    </Segment>

[>>>> custom values can be added here...]

</Values>
</Data>
```

Note the following:

- You must have write permissions to the C:\Program Files\Altova\MapForce2019\MapForceEDI directory in order to modify its contents.
- When a field has both locally-defined values (such as those in the code listing above) and globally-defined code values (in a .codelist file, as shown in Adapting Code Values from...
External Sources), then make sure that local code values are also defined as global code values. Otherwise, a validation error*** will be raised: **(Value) is not a legal value for field (Field)**.

- Inline (local) validation can be used if you require validation at field level (that is, the possible values that a field can take can be expressed as a list). Inline validation cannot be used for those fields that require complex validation rules at character level. When the field needs complex validation rules at character level (such as those defined in ISO 6346 applicable to Data Element 8155 of the UN/EDIFACT standard), add the code values globally, as shown in Adapting Code Values from External Sources. You can then configure validation as shown in Configuring Character-Level Validation.

*** Note that the actual validation outcome takes into account the settings configured for the rule "Invalid code list value" in the EDI Validation Settings dialog box. (To open this dialog box, click the Validation button on the EDI Component Settings dialog box).

### 7.5.15.4 Configuring Character-Level Validation

In most cases, validation of EDI fields means comparing, at mapping runtime, the field value with a list of possible values defined in a MapForce configuration file. For example, if the field value matches one the possible values defined in the configuration file, it is considered valid.

However, certain fields might need more advanced validation rules, meaning that their validation logic cannot be expressed as a simple list of possible values. An example of a field which requires complex validation is Data Element 8155 used in the VERMAS (Verification of Mass) message of UN/EDIFACT D16A. This field accepts not only code values defined in the UN/EDIFACT standard, but also external codes specified by the ISO 6346 standard. The ISO 6346 standard, in its turn, defines validation rules at character-level, namely:

- First character can be one of these: 1, 2, 3, 4, A, B, C, D, E, F, G, H, K, L, M, N, P
- Second character can be one of these: 0, 2, 4, 5, 6, 8, 9, C, D, E, F, L, M, N, P
- Third and fourth character can be: G0, G1, G2, G3, etc. (values are too many to be listed here).

It is possible to adapt validation logic such as the one above to MapForce as well. In this case, MapForce will validate Data Element 8155 not only according to the UN/EDIFACT code values, but also taking into consideration the rules above applicable to the ISO 6346 standard.

From a MapForce configuration perspective, the rules expressed by the three bullets above can be regarded as three separate lists of values. The first list is one character long, the second list is one character long, and the third list is two characters long. All three lists are finite and their values are known. This information makes it possible to validate the value of the data element by means of MapForce configuration.

To see an example of how character-level validation must be configured, open the ISO6346.Codelist file from the C:\Program Files\Altova\MapForce2019\MapForceEDI\EDIFACT directory. This file supplied by default with MapForce is configured to perform validation of Data Element 8155 according to ISO 6346 rules stated above.
ISO6346.Codelist configuration file

**Note:** Some of the values from the sample code listing above were removed, for simplicity.

In the code listing above, notice that, the `<Field>` element is defined three times. This makes it possible to validate the field content at character level. Namely, each occurrence of `<Field>` validates one of the rules stated in the bullets above at a time. To achieve this, each `<Field>` has several attributes (highlighted in the code listing above):

- **Name** - This attribute indicates that the list of values being validated belongs to the same
field. In other words, it binds all three lists to the same field (in this case, "F8155").

- **ListName** - This attribute gives a name to a list of values. Lists of values with the same name are grouped into one code list. Notice that in the example above all three lists have the same `ListName="ISO6346"`. This is needed specifically for character level validation. The actual characters to be validated are specified by the `Length` and `Offset` attributes of each list in the group, as explained below.

- **Length** - This attribute specifies the number of character to be validated (the starting position of characters in the field is defined by the `Offset` attribute). Notice that character length varies for each `<Field>`: the first and the second list of values is one character long, while the third list of values contains two characters.

- **Offset** - This attribute specifies the position at which MapForce (or the external execution engine) must start reading the characters from the data field which is being validated. The first character has offset 1, the second character has offset 2, and so on. The actual number of characters is defined by the `Length` attribute.

If the `Length` attribute is defined, the `Offset` attribute must be defined as well. Also, the value of both `Length` and `Offset` must be greater than zero.

In the code listing above, the following Length-Offset combinations were defined:

- `Length="1", Offset="1"` - this combination will validate one character of the field, starting at offset 1
- `Length="1", Offset="2"` - this combination will validate one character of the field, starting at offset 2
- `Length="2", Offset="3"` - this combination will validate two characters of the field, starting at offset 3

Each Length-Offset combination, combined with the code values defined under `<Values>`, represents a validation condition at character level. When the mapping runs, all character-level conditions must be satisfied (because all have the same `ListName` attribute value) for field validation to be successful. In other words, when validating a message that includes Data Element 8155, MapForce will apply the following logic:

- The field characters are validated against the list of code values defined in the configuration file (in this case, `ISO6346.Codelist`) for a matching Length-Offset combination. For example, the first character of the field must be one of the following: 1, 2, 3, 4, B, C, G, H, L, M, N.
- The field value must satisfy all conditions at character-level (in this case, three conditions
must be satisfied) in order for validation to be successful.

- If any of the conditions is not satisfied, the field value is not valid, and a validation error is
  produced in MapForce (or in the execution engine running the EDI mapping (such as
  MapForce Server, or a C#, C++, or Java program).
- A field may have more than one list of possible code values. In ordered to be considered
  valid, the field value must be present in at least one of the lists.
7.6 JSON

Altova website: JSON mapping

JSON (JavaScript Object Notation) is a very popular data format for storing and transporting data. In MapForce Enterprise Edition, you can map data from any formats supported by MapForce to JSON, and vice versa. You can also serialize JSON data as a string (for example, for writing to a database), or parse a JSON string into a JSON structure, with the help of a Parse/Serialize components, see Parsing and Serializing Strings.

To use JSON files in mappings, the mapping language must be set to one of the following: Built-in, C#, Java. For more information, see Selecting a transformation language.

Similar to other component types supported by MapForce, a schema file is required before you can read or write JSON data. If you don't have a schema file, MapForce can generate one automatically, see Adding JSON Files as Mapping Components. If you supply a JSON schema file not generated by MapForce, then MapForce expects it to be of Draft 04, Draft 06, Draft 07 version, or have no version at all.

The following is an example of how a JSON structure appears in MapForce:

As illustrated above, the structure of nodes in the JSON component reflects the structure of the JSON instance file. The basic JSON types are conventionally represented in MapForce as shown below.

<table>
<thead>
<tr>
<th>Type</th>
<th>MapForce representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>[ ]</td>
</tr>
<tr>
<td>Boolean</td>
<td>01</td>
</tr>
<tr>
<td>Null</td>
<td>{}</td>
</tr>
<tr>
<td>Number or integer</td>
<td>#</td>
</tr>
<tr>
<td>Object</td>
<td>{}</td>
</tr>
</tbody>
</table>
There are also special cases, as follows.

<table>
<thead>
<tr>
<th>Case</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>additionalProperties</code></td>
<td>The node appears on the JSON component under any object whose <code>additionalProperties</code> property is true or not present in the schema. It allows you to map to or from properties not explicitly listed in the schema (see also Example: Mapping from JSON to CSV). This node can also appear for objects having the <code>patternProperties</code> property.</td>
</tr>
<tr>
<td><code>patternProperties</code></td>
<td>JSON schema allows defining subtypes for objects and arrays (anyOf, allOf, oneOf). MapForce displays such subtypes using special structure nodes that do not have a direct representation in the JSON instance file. If your mapping is reading from a JSON file, such subtype nodes provide a value only if the current input value is valid according to the subtype schema. If your mapping is writing to a JSON file, make sure that you choose the correct subtype to fill. Filling multiple subtypes may lead to duplicate object properties and, thus, may result in invalid output JSON files.</td>
</tr>
<tr>
<td><code>Multiple types at the same location</code></td>
<td>JSON schema allows multiple types to occur at the same location. In such cases, the MapForce component displays separate structure nodes for all basic types that can occur at that location.</td>
</tr>
<tr>
<td><code>Type names</code></td>
<td>MapForce displays the <code>title</code> and <code>description</code> properties of types in the JSON schema in the &quot;type&quot; and &quot;annotation&quot; fields, if available. If <code>title</code> is absent, MapForce may also use part of the URI from the <code>$ref</code> property as a type name.</td>
</tr>
<tr>
<td><code>Arrays containing mixed item types</code></td>
<td>If an array has different types of items in the JSON schema (for example, both strings and numbers), MapForce displays an &quot;item&quot; node for each item type. When writing to a JSON file, this enables you to create arrays which contain items of different types.</td>
</tr>
<tr>
<td><code>Arrays defined as tuples</code></td>
<td>If an array has items whose type is assigned by position in the JSON schema, MapForce displays the zero-based index of the item as separate structure nodes (for example <code>item[0]</code>, <code>item[1]</code>, and so on. When writing to a JSON file, this enables you to specify the type of each individual item in the array by its zero-based index.</td>
</tr>
</tbody>
</table>
7.6.1 Adding JSON Files as Mapping Components

Reading or writing JSON instance files with MapForce requires a JSON schema file. You can either generate the JSON schema file with MapForce, or supply it as a custom file, as shown below.

At the time when this documentation is written, there is no formal convention for naming JSON schema files. Therefore, to distinguish JSON schema files from instance ones, you can use the `schema.json` extension for your JSON schema files (for example, `Example.schema.json`), and MapForce will recognize them as such.

To add JSON files as mapping components:

1. On the Insert menu, click JSON Schema/File. A dialog box may prompt you to select the type of the JSON file (schema or instance). This dialog box does not appear if MapForce positively identifies the file as a JSON schema.

2. Select Schema or Instance, as required.

   The size of the JSON instance file must not exceed your system’s available memory.

   If you select Instance, MapForce generates automatically the schema based on the instance file, and prompts you for a location to save the schema.

   If you select Schema, MapForce prompts you to specify a sample JSON file or a global resource (see Global Resources). This is required to preview the transformation and is meaningful only when you want to read data from a JSON file. Click Skip if your mapping is going to write data to JSON.
### 7.6.2 JSON Component Settings

To change the settings of a JSON component, right-click the JSON component header, and then click **Properties** (alternatively, double-click the JSON component header).

![JSON Component Settings dialog box](image)

The available settings are as follows.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Component name</strong></td>
<td>The component name is automatically generated when you create the component. However, you can change the name at any time. The component name can contain spaces and full stop characters. It may not contain slashes, backslashes, colons, double quotes, leading or trailing spaces. If you want to change the name of the component, be aware of the following:</td>
</tr>
<tr>
<td></td>
<td>• If you intend to deploy the mapping to FlowForce Server, the component name must be unique.</td>
</tr>
<tr>
<td></td>
<td>• It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line.</td>
</tr>
</tbody>
</table>
# Data Sources and Targets

## Schema file

Specifies the file name and path of the schema file.

To change the location of the file, click **Browse** and select the new file.

To edit the file in your JSON editor (for example, XMLSpy), click **Edit**.

## Input JSON File

Specifies the file name and path of the input JSON instance.

To change the location of the file, click **Browse** and select the new file.

To edit the file in your JSON editor (for example, XMLSpy), click **Edit**.

## Output JSON File

Specifies the file name and path where the JSON target instance file is placed, when the mapping is executed by MapForce Server.

This is also the default location when you save the output from the **Output** tab.

To change the location of the file, click **Browse** and select the new file.

To edit the file in your JSON editor (for example, XMLSpy), click **Edit**.

## Pretty print output

Reformats your JSON document when the mapping is executed, in order to give it a structured display. Each child node is offset from its parent by a single tab character.

## Encoding

Allows you specify the following settings of the output instance file:

- Encoding name
- Byte order
- Whether the byte order mark (BOM) character should be included.

JSON files are expected to have UTF encoding (see [https://tools.ietf.org/html/rfc7159#section-8.1](https://tools.ietf.org/html/rfc7159#section-8.1)). Other encodings are considered non-standard.

## Use JSON5

Select this check box if the input JSON file from which the component reads data uses JSON5 syntax.

In a target component, selecting the check box instructs MapForce to create a .json5 file where object keys are unquoted if they are valid ECMAScript5 identifiers. See also [JSON5 Support](#).

## Save all file paths relative to MFD file

When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. This setting affects the following files:
7.6.3 JSON5 Support

JSON5 is a proposed extension to JSON that aims to make it easier for people to write JSON files by hand. Importantly, JSON5 is not an official successor to JSON (for more information, see https://json5.org/).

JSON5 files use the .json5 extension; however, at the time when this documentation is written, there is no official MIME type for .json5 files. The .json5 extension is nevertheless recognized by MapForce when you add such files to the mapping. If you deploy the mapping to FlowForce Server (see Deploying Mappings to FlowForce Server), the MIME type of .json5 files will be reported as "application/json5".

Prerequisites

- In order to map data to or from JSON5 files, a valid JSON schema (which should not use JSON5 syntax) must be supplied to MapForce. If you don't have a JSON schema for your file, MapForce will generate it automatically, as shown below. The schema generated automatically by MapForce will be a JSON (not a JSON5) schema. It is also possible to generate a JSON schema for a .json5 file with XMLSpy.

Limitations

- Mapping comments to or from .json5 files is not supported.
- Because MapForce works with strict data types, it is not possible to qualify and map values such as Infinity, -Infinity, NaN, or -NaN.

How to map data to or from .json5 files

1. On the Insert menu, click JSON, and browse for the .json5 instance file (Alternatively, click the toolbar button). If you have a JSON schema file and prefer to supply the .json5 instance file later, browse for a valid JSON (not a JSON5) schema. This works in the same way as with plain JSON files (see Adding JSON Files as Mapping Components).
2. After the JSON component has been added to the mapping, right-click its header, select **Properties**, and then select the **Use JSON5** check box from component properties (see also **JSON Component Settings**). This explicitly instructs MapForce to treat the instance data as such.

![Component Settings](image)

To generate a JSON5 file with MapForce, select the **Use JSON5** check box as shown above, and connect the required input data to the JSON component on the mapping. When the **Use JSON5** check box is selected, MapForce will not surround the object keys by quotes, except where this would make the generated JSON5 instance invalid. The desired file extension (for example, "json5") can also be controlled from the dialog box illustrated above, by entering the appropriate value in the **Output JSON File** field.

### 7.6.4 Example: Convert JSON to CSV

This example shows you how to read data from a JSON file and convert it to a comma-separated text file with MapForce. You will also learn how to map data from additional properties that might be present in JSON instance files like the one illustrated in this example. Additional properties refers to data that is present in the JSON instance file even though it is not defined by the schema. As a rule, you should avoid relying on additional properties and instead define all the properties in your JSON schema. However, in the event that the JSON instance contains additional properties, you can use the technique illustrated in this example.

This example is accompanied by a demo mapping file available at the following path:
If you follow the steps below, you will learn how to create such a mapping file from scratch.

The source JSON file contains people records that must be converted to CSV. Notice the highlighted text: the first and the second person have an additional property that is not defined in the schema, namely birthday.

```json
[
  {
    "name": "Alethia Alonso",
    "email": "altethia@example.com",
    "age": 35,
    "birthday": "4 July"
  },
  {
    "name": "Klaus Mauer",
    "email": "klaus@example.com",
    "age": 57,
    "birthday": "31 August"
  },
  {
    "name": "Natsuo Shinohara",
    "email": "natsuo@example.com",
    "age": 29
  }
]
```

*People.json*

The JSON schema used in the example is represented below. As the `$schema` keyword indicates, the schema validates JSON instances with respect to Draft 04 JSON Schema. It describes a `people` array which consists of multiple `person` objects. The `people` object must contain at least one `person` object to be valid. Each `person` object has a `name`, `age`, and `email` address as properties. Note that `name` and `email` are of type `string`, while `age` is of type `integer`. Also, the `name` and `email` properties are required, while `age` is optional.

```json
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "people",
  "type": "array",
  "items": {
    "title": "person",
    "type": "object",
    "required": [
      "name",
      "email"
    ],
    "properties": {
      "name": {
        "type": "string"
      },
      "email": {
        "type": "string",
        "format": "email"
      },
      "age": {
```
Importantly, the JSON schema above does not have an `additionalProperties` property for the `person` object. When not present in the JSON schema, the `additionalProperties` property has the default value of `true`, which means that the object in the JSON instance can have as many additional properties as required, and still be valid. Therefore, a JSON instance file like `People.json` will validate successfully against this schema.

To achieve the mapping goals of this example, take the steps below.

**Step 1: Add the source JSON file to the mapping**

1. On the **Insert** menu, click **JSON Schema/File** and browse for the following file: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\People.schema.json`. When prompted to specify an instance, select the `People.json` file in the same folder. At this point, the MapForce component looks as follows:

![MapForce component](image)

The structure of the MapForce component resembles that of the JSON file itself, with the exception of `property (additional)` node. This node indicates that the `additionalProperties` property of the `person` object is either missing or set to `true` in the schema. This means that the schema can contain custom additional properties, so MapForce displays the node in case you want to map from any additional properties of the object (the next steps show how to do this).

To change the source JSON file or the JSON schema file at any time, double-click the title bar of the JSON component and tweak the settings, see **JSON Component Settings**.

**Step 2. Add the target CSV to the mapping**

1. On the **Insert** menu, click **Text File**.
2. Select **Use simple processing for standard CSV**, and then click **Continue**.

3. Click **Append Field** several times to add four CSV fields as shown above, and then click **OK**.

For more information about text components, see [CSV and Text Files](#).

**Step 3: Draw the mapping connections**

1. On the mapping, draw the connections between the JSON component and the CSV component as shown below.

For more information, see [Working with Connections](#).

**Step 4: Preview the mapping**

You can now quickly preview the transformation output by clicking on the **Output** tab. The result
is:

<table>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alethia Alonso</td>
<td><a href="mailto:altethia@example.com">altethia@example.com</a></td>
<td>35</td>
</tr>
<tr>
<td>Klaus Mauer</td>
<td><a href="mailto:klaus@example.com">klaus@example.com</a></td>
<td>57</td>
</tr>
<tr>
<td>Natsuo Shinohara</td>
<td><a href="mailto:natsuo@example.com">natsuo@example.com</a></td>
<td>29</td>
</tr>
</tbody>
</table>

As the output shows, for each person object in the JSON file, a new row is created, and object properties are comma-separated, which is the intended behavior.

However, we have not mapped yet the birthday field which exists in the JSON instance file, even if it does not exist in the schema.

**Step 5: Map additional properties**

Recall that the field birthday is an additional property not defined in the schema. To read data from such fields, MapForce provides additional items on the JSON component.

In this example, in order to read data from the birthday field, add the following to the mapping:

1. A constant with the value "birthday", see [Add a Constant to the Mapping](#).
2. A filter component, see [Filters and Conditions](#).
3. The logical function equal, see [Add a Built-in Function to the Mapping](#).

Next, connect the components as illustrated below. Connecting components this way instructs MapForce to look for an additional property with the name birthday, and, if a string value is found, copy it to Field4 of the text component.

In the mapping above, the name item under property (additional) allows you to access the additional property by its name. The subsequent items (string, number, boolean, and so on) correspond to JSON data types. They can be used to access the actual JSON value of that type. In our example, the name of the additional property is "birthday" and the type is "string", which is why these two items are connected to the filter.

If you preview the transformation output at this time, the result is:
As the output shows, the fourth CSV field now includes the value of the only additional property of type "string", which is, in this case, birthday. Also, since the third person does not have a birthday, no value is available in the CSV at the corresponding position.

### 7.6.5 Example: Convert Excel to JSON

This example shows you how to create a JSON document from an Excel file with MapForce. The source Excel workbook consists of two worksheets: Office and Staff. The Office worksheet provides information about the company’s address, while the Staff worksheet lists all the employees.

The business requirements are as follows:

1. Extract data from the Excel file and convert it into a JSON document.
2. The JSON file must specify, for each employee, whether their phone extension can be shared for external contracts. Only phone extensions of employees from the Marketing and Administration departments can be shared externally.
3. The shares of each employee must be indicated in the JSON file as a numeric value, which may include zero. If not known, shares must be indicated as null in the JSON file.

The mapping that accomplishes the goal above is illustrated below. This mapping is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Excel_Company_To_JSON.mfd.`
Excel_Company_To_JSON.mfd

As shown above, the mapping consists of a source Excel component and a target JSON component. Notice that the two worksheets of the Excel file appear as on the source Excel component. The company name is mapped from the Office worksheet, while the rest of the fields are mapped from the Staff worksheet. To view how the mapping is configured to read row data from each worksheet, click the corresponding button. For the Staff worksheet, the settings are illustrated below.
The settings above instruct the mapping to read rows starting from 1 up to the end (dynamically). The first row is a header with column names, so the First row is header... check box is selected. The dialog box also specifies the type of columns A through G, and the name with which each column appears on the mapping. For more information about Excel components, see Microsoft OOXML Excel 2007+.

The target component is a JSON structure. You can add JSON components to the mapping as described in Adding JSON Files as Mapping Components. The structure reflects the JSON schema, employees.schema.json, which is available in the same folder as the mapping. For more information about configuring the JSON component, see JSON Component Settings.

There also several intermediate components on the mapping. The first one is a value-map component. The value-map decides, based on the department, whether the phone number can be shared externally. This corresponds to business requirement #2 above. You can view the settings of the value-map component by double-clicking its title bar:
The configuration above can be summarized as follows: If Department is "Administration" or "Marketing", return true. Otherwise, return false. The true or false value computed this way is then written to the target JSON property. For more information about value maps, see Using Value-Maps.

Finally, the mapping makes use of the exists function whose purpose is to determine, in the context of each row, if a Shares value exists in the source Excel file. This function returns a Boolean true if the value exists, or false otherwise. The resulting true or false value is then supplied as input to an If-Else condition. On false, an empty string provided by a constant is passed on to the Shares as null target item. This empty value will become a null in the resulting JSON file, because this is the data type of the Shares as null item, as indicated by the icon.

If the employee's share value is numeric, it will no longer meet the If-Else condition. Instead, it will be written to the Shares as number target item (which is of numeric type number), through a separate connection.

For more information adding functions to a mapping, see Add a Built-in Function to the Mapping. If-Else conditions are covered in more detail under Filters and Conditions.

Running the mapping
To preview the generated file in MapForce, click the Output tab. The image below illustrates only a fragment of the resulting JSON file.
To save the mapping output to a file, do one of the following:

- On the Output menu, click Save Output File.
- Click the Save generated output toolbar button.

**Automation with MapForce Server**

If you have licensed MapForce Server, you can also run the mapping at the command line, on a Linux, macOS, or Windows machine. Note that running mappings on a server typically requires a few extra steps as outlined in Preparing Mappings for Server Execution. Here are the steps required to run this particular mapping on a server:

1. Right-click the mapping and select Mapping Settings from the context menu. Make sure to clear the Make paths absolute in generated code check box and save the mapping.
2. On the File menu, click Compile to MapForce Server Execution File. This generates an executable .mfx file in the <Documents>\Altova\MapForce2019\MapForceExamples directory.
3. Copy the generated .mfx file to a directory on the server machine (let's call it the "working directory").
4. Copy the Excel file from <Documents>\Altova\MapForce2019\MapForceExamples\Nanonull Inc.xlsx to the working directory on the server machine.
5. Open a terminal and change the current directory to the working directory.
6. Run MapForce Server with the command below. Adjust mapforceserver_exec to be the path to the MapForce Server executable as applicable for your operating system.

```
mapforceserver_exec run Excel_Company_To_JSON.mfx
```

In server execution, you can also run mappings as an API call, or as FlowForce Server jobs, either on demand or on recurring basis. For more information, see Compiling Mappings to MapForce Server Execution Files.
### 7.7 Microsoft OOXML Excel 2007+

MapForce can read data from and write data to Microsoft Excel 2007+ workbooks, in the default *Office Open XML (OOXML)* format. This format was first introduced in Microsoft Office 2007 and, in case of Excel workbooks, is associated with the default `.xlsx` extension. In the MapForce interface, and in this documentation, Microsoft Excel 2007 and later files are generically referred to as "Excel 2007+" files.

Microsoft Office 2013 added support for *Strict Open XML Spreadsheet* format (ISO/IEC 29500 Strict). The *Strict Open XML Spreadsheet* format also has the `.xlsx` extension; however, technically this is a distinct format which adheres to stricter validation rules.

The following table illustrates how reading and writing data from/to Excel 2007+ workbooks is supported across MapForce transformation languages. Notice the differences between *Office Open XML* and *Strict Open XML Spreadsheet* formats.

<table>
<thead>
<tr>
<th>Microsoft Excel Format</th>
<th>MapForce Language</th>
<th>Reading Support</th>
<th>Writing Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Office Open XML</td>
<td>BUILT-IN</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>C#</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Java</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>XSLT2</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Strict Open XML Spreadsheet</td>
<td>BUILT-IN</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>C#</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Java</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>XSLT2</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

If you need to convert any *Office Open XML* files generated by MapForce to *Strict Open XML Spreadsheet* format, open the workbook in Excel 2013, and then save it as *Strict Open XML Spreadsheet*. 

---

© 2019 Altova GmbH

Altova MapForce 2019 Enterprise Edition
Saving to Strict Open XML Spreadsheet format

As mapping components in MapForce, Excel 2007+ files have the following general behavior:

- You can map data from Excel 2007+ to any component supported in MapForce, and vice
versa, including XBRL taxonomy files (see Excel to XBRL example).

- If Microsoft Excel 2007 or later is installed on your computer, you can preview the transformation output immediately in the Output tab of the mapping window, and you can save it to a file. If you don't have Excel 2007 or later, you can still map to or from Excel 2007+ files. In this case, you cannot preview the result in the Output tab, but you can still save it, by clicking Save Output File on the Output menu.

- The generated Excel 2007+ output does not contain any formatting, only data. If you need to format the Excel output generated by MapForce, you can reference the generated cell ranges from another Excel sheet and apply formatting in the referencing (not the referenced) sheet. For further information, see Example: Supplying Data to Preformatted Excel Sheets.

- As an alternative to generating and saving the output manually, you can compile the mapping design to a MapForce Server execution file, or deploy it to a FlowForce Server, and execute it as and when required through FlowForce Server jobs. For further information, see Compiling a MapForce mapping and Deploying a MapForce mapping, respectively.

- You can generate mapping code (see the supported languages above) and execute the mapping from Visual Studio 2008, 2010, 2013, 2015, 2017 or from your custom application. For more information, see Code Generator.

## 7.7.1 Adding Excel 2007+ Files as Mapping Components

To add a Microsoft Excel 2007+ (*.xlsx) component to the mapping area:

1. Do one of the following:
   - Click the Insert Excel 2007+ File toolbar button.

2. Do one of the following:
   - If you want to map data from a Microsoft Excel workbook, click Browse to select the .xlsx file from which you are mapping data. MapForce uses the sample Excel file to read worksheet names and cell ranges from it. If you have defined any Excel files as global resources, you can also select them from the Global Resources dialog box (see Global Resources). Click Skip if you would like to provide a sample file later.
   - If you want to map data to a Microsoft Excel workbook, click Skip. By default, when the mapping transformation runs, MapForce will generate an output Excel file named $xlsm-mapforce.xlsm$ in the mapping folder. If required, you can change the name of the output file from the Excel 2007+ Component Settings (see About the Excel 2007+ Component).
7.7.2  About the Excel 2007+ Component

When you add an Excel 2007+ file to the mapping area without specifying a sample file (see Adding Microsoft Excel Files as Mapping Components), MapForce creates a default component which includes three worksheets (illustrated below). If you provide a sample file, MapForce reads the sample file and creates only the required worksheets.

![Default Excel component](image)

The structure of the Excel 2007+ component in MapForce reflects the structure of data in the Excel workbook, with the difference that in MapForce it is expressed in a tree structure which makes it possible to map each individual cell.

Before you can connect the Excel 2007+ component to any other component type, you will need to instruct MapForce precisely what are the columns and rows to be used in the mapping. Unlike other MapForce components such as XML or JSON, Excel 2007+ files do not have an explicit schema that MapForce can use to infer the structure of your data. Instead, MapForce provides you with settings from where you can define:

- What data precisely (such as worksheets, named ranges and tables, columns, rows) must be selected from your workbook (if you are reading from a workbook);
- To which worksheets, columns, and rows must MapForce write data (if you are writing to a workbook).

You can also configure the component to read from multiple locations within a workbook, or write to multiple locations, in the same mapping operation.

The required component configuration settings are available directly on the component. Use the following table to get started.

<table>
<thead>
<tr>
<th>Component Item</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workbook</td>
<td><img src="image" alt="Workbook Icon" /></td>
<td>Represents an Excel workbook.</td>
</tr>
<tr>
<td>Worksheet</td>
<td><img src="image" alt="Worksheet Icon" /></td>
<td>Represents an Excel worksheet.</td>
</tr>
</tbody>
</table>
### Rows

Represent a range of Excel rows. You can add multiple ranges of rows within a worksheet, see Adding and Removing Row Ranges. This enables you to read from (or write to) multiple ranges of cells in the same mapping operation.

For each defined range, you can specify individual data selection options. For example, one range may begin at row 1 and include all columns of that row, while another range may begin at row 3, and consist of a dynamic number of rows, depending on the amount of data in the source Excel file.

To help you see all range settings at a glance, the component provides visual clues about them, as shown below.

- **Rows \( n \)**: Indicates a range which begins at row \( n \).
- **Row \( n \)**: Indicates a single-line range of row \( n \).
- **Rows prev + \( n \)**: Indicates a range which begins \( n \) rows after the previous range.
- **Rows \( n(h) \)**: Indicates a range which begins at row \( n \), and the first row is designated as a header row.
- **\( n=n \)**: Indicates a range which consists of exactly \( n \) rows.
- **\( n=dyn \)**: Indicates a dynamic range. Dynamic ranges may have an unlimited number of rows.

The button displayed next to each row range lets you specify advanced data selection settings for that range.

### Cells

Represents all the cells (columns) of a particular row. This item appears if the component is configured to show a single cell for all columns (this is the default MapForce behavior).

Alternatively, you can configure a component to display each column separately, in which case it would look as shown in the following sample.
### Component Item | Icon | Description
--- | --- | ---
|  |  | 
|  |  | The button displayed next to each worksheet or row lets you specify settings meaningful in that context. Using this button, you can modify the mapping structure of the Excel component as required, see Excel 2007+ Component Settings.

For example, if you are reading data from an Excel file, you can specify the worksheet, row and column from where MapForce should read data. If you are writing to an Excel file, you can specify the worksheet, row and column to which MapForce should write data.

#### 7.7.3 Adding and Removing Worksheets

You can add or remove worksheets on the Excel 2007+ component, either manually, or by reloading them from an input .xlsx file. To do so, click the button next to a worksheet node.
The options applicable to worksheets are as follows.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Show Worksheets by name</strong></td>
<td>This option must be selected if each worksheet in your workbook has a different layout and therefore must appear as a separate item in MapForce. If all worksheets in your workbook have an identical structure, you can make this option inactive. This way, MapForce will collapse the worksheet items to a single item representing the ordered collection of all worksheets. If a workbook has multiple worksheets and the <strong>Show Worksheets by name</strong> option is inactive, then MapForce treats the workbook like a single worksheet. This allows a mapping to process any number of worksheets at once, but requires that all worksheets have the same structure.</td>
</tr>
<tr>
<td><strong>Insert (↵)</strong></td>
<td>Inserts a new worksheet before the currently selected one.</td>
</tr>
<tr>
<td><strong>Append (↵)↑</strong></td>
<td>Appends a new worksheet. Type a name into the text field to the right of the icon.</td>
</tr>
</tbody>
</table>
### 7.7.4 Adding and Removing Row Ranges

You can instruct MapForce to read (or write) a particular range of cells at a particular location within a worksheet. The following sample component illustrates a range which is available in Sheet2 of the workbook.

![Excel file](image)

By default, any range is set to begin at Row 1 and iterate dynamically for n rows ($n=\text{dyn}$). However, you can change these and other settings if required (see Selecting Ranges of Cells).

You can create as many ranges of rows as required within the same worksheet, and remove the ones you do not need. To add a new range of rows, right-click any Rows node, and then select Add Rows Before or Add Rows After, respectively.
Adding cell ranges

To delete a range, right-click it, and then select **Remove Rows**.

To move a range up or down in the component, right-click it, and then select **Move Up** (or **Move Down**, respectively).

### 7.7.5 Selecting Ranges of Cells

You can define what range of cells must be read by MapForce (when reading from a workbook) or written to (when writing to a workbook) from the "Select Range of Cells" dialog box. To open this dialog box, click the button next to a cell range on the component.
Select Range of Cells dialog box

The available settings are as follows.

<table>
<thead>
<tr>
<th>Load Range from Excel Input File</th>
<th>If you are reading from a workbook</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use this option to select a particular worksheet range, named range or table.</td>
<td>Use this option to select a particular worksheet range, named range or table.</td>
</tr>
</tbody>
</table>
If the **Show Worksheets by name** option is disabled (see [Adding and Removing Worksheets](#)), data from all worksheets is visible in the list.

The **Refresh** button updates the cell ranges from the input Excel file.

Note that only rectangular ranges are currently supported.

**Starting Row**

The **Row** option lets you define the first row of data for the specific range. For example, if you enter "5" as starting row, MapForce will read (or write) beginning with the fifth row of the workbook.

The **Previous range with offset** option is meaningful if there is another range in the same worksheet. It instructs MapForce to move the current range \(N\) rows down from the previously defined range. The minimum offset value is 1.

**Row Count**

If you are reading from a workbook  
**Count** defines the exact number of rows from which you want to read data, starting from the position defined in the **Starting Row** (see previous option). This value is automatically populated if you selected an Excel named range or table. The option **Dynamic** instructs MapForce to read all rows found in the source data beginning with **Starting Row**. Use this option only if your range is the last defined range of the worksheet, otherwise any subsequent range will not select data from the source Excel file.

If you are writing to a workbook  
**Count** defines the exact number of rows to which data should be written, starting from the position defined in the **Starting Row** (see previous option). Note that if your input instance contains more rows than allowed by **Count**, MapForce writes only the number of rows defined by **Count**, and ignores the rest of data without any warning.

The option **Dynamic** instructs MapForce to write all rows found in the input instance, beginning with **Starting Row**.

If you defined a header row using the **First row is header**
with column names option, Count does not take the header row into account (see the Excel_Company_to_XML.mfd sample).

Show a single Cells item for all columns

Collapses all cell items into a single mappable Cells item as shown below.

If you are reading from a workbook
Use this option if you want to read all the cells of a particular row. For examples, see the ExcelColumnsToRecords.mfd and ExcelWith2Dimensions.mfd samples available in the MapForceExamples project, OOXML Excel 2007+ folder.

If you are writing to a workbook
Use this option to write data to one or multiple cells in the same row. For an example, see the Altova_Hierarchical_Excel.mfd sample.

Show separate items for columns

This option enables you to access individual columns of the given row range.

If you selected a worksheet range, named range or table, the column names are automatically populated. Otherwise, you
can select specific column names by typing their corresponding alphabetic letter in the from and to text boxes.

If the ranges in the input Excel file have changed, click Reload to update the component with the changes.

To instruct MapForce to consider the first row of a range as the column header for that range, select the First row is header with column names option. When you activate or deactivate this option, and Row Count has been set, MapForce prompts you to optionally adjust the Row Count value. This prevents the Row Count from being one row too large, or too small.

Note that the Row Count setting does not take the header row into account.

7.7.6 Inserting Columns Between Existing Ones

When you map data to an Excel spreadsheet, it is possible to easily add new columns in between any of the existing columns, without redoing the existing mapping connections. To do this, right-click a cell in the Excel component, and select Add Cell Before from the context menu.

Note: The Add Cell Before command is available if the cell belongs to a row range (that is, it must be child of a "Row" or "Rows" item of an Excel component). Also, the option Show separate items for columns must be enabled in the "Select Range of Cells" dialog box (see Selecting Ranges of Cells).

For example, let's suppose you have created a mapping which writes data from an XML file to an Excel spreadsheet (such as Sales_to_Excel.mfd from the folder <Documents>\Altova\MapForce2019\MapForceExamples). This mapping outputs an Excel spreadsheet which consists of three columns (Month, West, and East), as illustrated below.
Assuming that you need to add a new column immediately before "West" (for example, "North"), this is possible without redoing any of the existing connections, as follows:

1. Right-click the cell B on the mapping, and select **Add Cell Before** from the context menu.

2. Optionally enter a column name (for example, "North").
3. Click OK.

The new column ("North", in this example) has now been added to the mapping, and it is now possible to draw a mapping connection to this column. All the existing connections are not affected by the change.

Existing columns can also be removed in a similar fashion: right-click a cell on the mapping, and select Remove Cell from the context menu.

7.7.7 Excel 2007+ Component Settings

After you add an “Excel 2007+” component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component Settings dialog box in one of the following ways:

- On the Component menu, click Properties (this menu item becomes enabled when you
select a component).

- Double-click the component header.
- Right-click the component header, and then click **Properties**.

![Component Settings dialog box](image)

The available settings are as follows.

| **Component name** | The component name is automatically generated when you create the component. However, you can change the name at any time. The component name can contain spaces and full stop characters. It may not contain slashes, backslashes, colons, double quotes, leading or trailing spaces. If you want to change the name of the component, be aware of the following:

- If you intend to deploy the mapping to FlowForce Server, the component name must be unique.
- It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line. |
| **Input Excel 2007+ file** | Specifies the Microsoft Excel file from which MapForce will read data. This field is meaningful in a source component, when MapForce uses it to read the Excel worksheet names and columns. To change the location of the file, click **Browse** and select the new file. To edit the file in Microsoft Excel, click **Edit**. |
### Output Excel 2007+ file

Specifies the name or path of the Microsoft Excel file to which MapForce will write data. This field is meaningful in a target component.

To change the location of the file, click **Browse** and select the new file. To edit the file in Microsoft Excel, click **Edit**.

If you do not specify an output file, MapForce generates the output to a file named `xlsx-mapforce.xlsx`. By default, this file is generated in the same folder as the `.mfd` file, unless you configured a different path.

Any existing data in the output Excel file will be overwritten when the mapping transformation runs.

### Show error for type mismatch

Enables error messages in the Messages window, when mismatches occur between a data type declared in the component and data in the input `.xlsx` file.

For example, let's assume that this setting is enabled, and you declared a column as numeric in the `Select range of cells` dialog box. If the Excel file contains text data in this column, an error message is shown in the Messages window after you connect this column to some output and preview the output.

### Save all file paths relative to MFD file

When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. See also Using Relative Paths on a Component.

### 7.7.8 Example: Mapping Excel 2007+ to XML

The mapping file used in the following example is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\Excel-mapping.mfd`. When you open the sample file, you will notice that it contains three distinct mapping transformations. The top two transformations are discussed in this section.

The aim of the first mapping is to do the following:

- Select from the source Excel workbook only people whose phone extension (column C of the workbook) starts with a “1”.
- Add the prefix “10” to the original number, and write it to a target XML file, along with the First and Last names of the respective persons.
Excel-mapping.mfd (sample 1)

The mapping is configured as follows:

- **Altova.xlsx** is the source Excel 2007+ workbook. Columns A and B supply the First and Last names respectively. Column C supplies the phone extension number.
- Both worksheets of the workbook are shown as one node in the component (in other words, the *Show worksheets by name* option is disabled). This is indicated by the "Worksheets" node under the Workbook item.
- The *starts-with* function checks if the phone extension (col. C) starts with a "1", and if the result is true then those records are forwarded by the filter component.
- The *concat* filter adds the prefix "10" to each of the telephone extensions and writes it to the PhoneExt item.
- MFCompany.xsd is the target component and contains the filtered person details when data is output.

The result of the mapping is that four persons have been mapped to the XML file with their details.
The second mapping is identical with the first one, except that worksheets have been individually enabled using the **Show worksheets by name** option. This mapping is configured as follows:

- The **Admin** and **Development** worksheets are both visible under the Workbook item.
- Connectors have only been defined from the **Admin** worksheet to the target component.

The result of the mapping is that only two persons have been mapped to the XML file with their details.

```
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns:xsi:schemaLocation="http://my-company.com/namespace"
</Person>
</Person>
</Person>
</Person>
</Person>
</Person>
</Person>
</Person>
</Person>
</Person>
</Company>
```

Excel-mapping.mfd (Sample 2)
Example: Mapping Database Data to Excel 2007+

The mapping file used in this example is available as Excel-mapping.mfd in the <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ folder. The third mapping of the three is discussed here.

The aim of the mapping is as follows:

- Extract from the "altova" database only persons whose department primary key is equal to 4 (that is, those who are in the IT department).
- Write the extracted records to a default Excel 2007+ component.

The mapping is configured as follows:

1. The database "altova" was added to the mapping area from the <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ folder, using the Insert | Database menu command, and following the wizard for a Microsoft Access database.
2. The default Excel 2007+ component was added using the **Insert | Excel 2007+ file** menu command, and then skipping the option to supply a sample file.

3. The first worksheet (Sheet1) was renamed by clicking the button adjacent to it and then entering “Altova” as worksheet name.

4. The **Rows 1, n=dyn** range was configured by clicking the button adjacent to it. The cell range options were defined as follows:

![Select Range of Cells](image)

5. Other options were defined as follows:

   - The value of the **PrimaryKey** is compared to the value "4", supplied by the **Constant** component, using the **equal** function.
o The filter component passes on the First and Last fields if the Boolean condition is true (that is, if the department primary key is "4").
o The on-true item is connected to the Rows 1, n=dyn item in the Excel file.

The result of the mapping is that four persons of the IT department are shown in the Excel workbook.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alex</td>
<td>Martin</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>George</td>
<td>Hammer</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Jessica</td>
<td>Bander</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Lui</td>
<td>King</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 7.7.10 Example: Supplying Data to Preformatted Excel Sheets

Excel sheets generated by MapForce do not contain any formatting, only data. If you need to format Excel data generated by MapForce, this is possible outside of MapForce, through standard Excel functionality, as follows:

1. Generate your Excel sheet with MapForce and save it to the disk (let's call it `xlsx-mapforce.xlsx`).
2. Create a new Excel sheet with Excel (let's call it `Sales-presentation.xlsx`).
3. From `Sales-presentation.xlsx`, create external references (links) to cell ranges from `xlsx-mapforce.xlsx`. This is a one-time operation; for detailed instructions, see [https://support.office.com/en-us/article/create-an-external-reference-link-to-a-cell-range-in-another-workbook-c98d1803-dd75-4668-ac6a-d7cca2a9b95f](https://support.office.com/en-us/article/create-an-external-reference-link-to-a-cell-range-in-another-workbook-c98d1803-dd75-4668-ac6a-d7cca2a9b95f).
4. Apply all the required formatting (fonts, colors, and so on) in `Sales-presentation.xlsx`.

From now on, you can generate data as many times as necessary into `xlsx-mapforce.xlsx`. To update the `Sales-presentation.xlsx` with the most recent data generated by MapForce, click the Refresh All button in the Data tab. This will keep all existing formatting intact.

In the example below, you will first generate some Excel data with MapForce and save it to a file. You will then be able to see this data nicely formatted, in a second Excel sheet (which is preconfigured to have external references to the first one). To proceed, open the following sample mapping: `<Documents>\Altova\MapForce2019\MapForceExamples\Sales_to_Excel.mfd`. 
Next, click the **Output** tab to generate the mapping result.

Next, click **Save generated output** and save it as "xlsx-mapforce.xlsx", to the folder `<Documents>\Altova\MapForce2019\MapForceExamples`.

You can now view this data formatted in a second sheet that is already configured to read data from `xlsx-mapforce.xlsx` file. Open the `Sales-presentation.xlsx` file in Excel (from the same folder as above). A warning appears that external links have been disabled and they need to be enabled to link to the source data. The current cursor position, B6, shows that the source data range is `Sales'!$B$3:$C$14` of the `xlsx-mapforce.xlsx` file.
Click the Enable button (or Enable Content, in Excel 2016). The external links are now enabled and the underlying source data is shown as a table and a chart.
You can also manually refresh the links between the two sheets, as follows:

- On the **Data** tab, click **Refresh all**.
7.8 XBRL

XBRL (Extensible Business Reporting Language) is an open international standard that enables exchange of business information. XBRL is used by governments, financial institutions, and businesses, primarily for financial and regulatory reporting. XBRL is managed by a global not for profit consortium, XBRL International (https://www.xbrl.org).

A company or organization may have different business needs in conjunction with XBRL (for example, XBRL validation, creation of XBRL data, creation of new XBRL taxonomies, export of XBRL data to other formats). Various Altova products make it possible to accomplish XBRL-related tasks as follows:

- **XMLSpy** edits or creates new taxonomies and generates XBRL reports based on XSLT transformation files (XSLT transformation files can be created in StyleVision).
- **StyleVision** creates taxonomy stylesheets/templates, allowing you to generate XBRL reports.
- **MapForce** maps data to or from XBRL instance files, and enables you to use XBRL taxonomies when designing the mapping structure. You can map XBRL data to or from other formats supported by MapForce (such as Microsoft Excel 2007 and later, databases, or CSV files). For example, you can filter specific data from existing XBRL instance documents, or create XBRL instances from data extracted from a database.
- **RaptorXML+XBRL Server** is a cross-platform XBRL processor which validates XBRL instances and taxonomies.
- **Altova Work in Process (WIP) XBRL add-in for Excel** enables businesses who submit XBRL US Work In Process reports to view and prepare XBRL report data using Microsoft Excel.

In order to map data to or from XBRL instance files in MapForce, the underlying XBRL taxonomy is required. The taxonomy consists of a file archive (usually .zip) which is provided by the organization or entity to which XBRL data is submitted. The following are examples of taxonomies that can be used for data mapping with MapForce:

- US-GAAP (Generally Accepted Accounting Principles), published by the Financial Accounting Standards Board (http://www.fasb.org)
- COREP / FINREP (Common Reporting and Financial Reporting), published by the European Banking Authority (http://www.eba.europa.eu)

As long as you have the taxonomy files from the relevant taxonomy provider, you can create data mappings without additional prerequisites. However, note that, by default, only the most recent versions of the US-GAAP taxonomy are included with MapForce and thus optimized for validation. Other taxonomies will likely take a longer time to load and validate, unless they are installed locally. To install additional taxonomies (or older taxonomy versions) locally, download and run the XBRL Taxonomies installer available on the Altova components download page (https://www.altova.com/components/download).

MapForce supports the following XBRL-related standards:

- XBRL 2.1
- XBRL Dimensions 1.0
Data Sources and Targets

- XBRL Table Linkbase 1.0 (see Working with XBRL Tables)
- XBRL Taxonomy Packages 1.0 (see XBRL Taxonomy Packages)
- Inline XBRL 1.0 and 1.1, including Transformation Registry 3 (only for reading data, see also Reading Data from Inline XBRL)

Conventions
This documentation makes references to XBRL terminology as defined by the following specifications.

<table>
<thead>
<tr>
<th>Specification</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inline XBRL</td>
<td><a href="https://specifications.xbrl.org/spec-group-index-inline-xbrl.html">https://specifications.xbrl.org/spec-group-index-inline-xbrl.html</a></td>
</tr>
</tbody>
</table>

In this documentation, references to the specifications above are as indicated by the Specification column. Additionally, the § character is used to denote a particular section number within the specification. For example, a reference to Section 5.4 of the XBRL Table Linkbase 1.0 specification looks as follows:

A table (Table Linkbase Specification 1.0, §5.1) is represented by the icon in the XBRL component.

7.8.1 Adding XBRL Files as Mapping Components

Before you can map data to or from an XBRL document, the XBRL document must be added to the mapping area as a mapping component. There are two ways of adding XBRL documents as mapping components:

- As a flat XML document
- As a hierarchically structured XBRL document

The first option inserts the XBRL file as a flat XML file without any business logic concepts or specific XBRL hierarchical structure. Such a component is useful only for very simple mappings where no dimension handling or automatic XBRL context generation is required.

The second option inserts the XBRL taxonomy and displays its contents in a hierarchical fashion suitable for XBRL. In this scenario, MapForce resolves the Discoverable Taxonomy Set (DTS) references and builds automatically the derived XBRL structure. This means that the XBRL mapping component will include XBRL-specific items (which may be actual nodes in the XBRL file or "abstract" nodes derived from the DTS). For example, the following hierarchical structure is
derived from the xlink:from and xlink:to presentation arcs in the presentation linkbase. (The term "linkbase" here has the meaning as found in the XLink specification (see https://www.w3.org/TR/xlink11/). Specifically, it represents a collection of links (or a link database). XLink linkbases are extended, among other uses, to XBRL, and provide additional information about concepts defined in a taxonomy.)

This example is only a basic illustration of how MapForce resolves the taxonomy in order to display data in a conceptual manner better suited for mapping operations than plain XML. In addition to the presentation linkbase, MapForce can also render structures derived from other linkbases, including the table linkbase (see Working with XBRL Tables).

For information about items that can appear on an XBRL component, including their conventional graphical representation, see About XBRL Component Items.

To open XBRL documents as flat XML:

- Select the XBRL instance file, and then click Open. Note that the instance file must reference a schema; otherwise, MapForce will prompt you to generate it.

To open XBRL documents with automatic context handling:

2. Select the taxonomy file (*.xsd) or an XBRL instance file (*.xbrl or *.xml). If you select a taxonomy, a further dialog will prompt you to select a valid instance file.
3. Click Browse... if you intend to use this XBRL component as a source instance and select the XBRL instance file.

When you open an XBRL taxonomy, a dialog box opens prompting you to select the structure views to shown on the XBRL component. If you are not sure what structure view to select, leave the default option as is. By default, MapForce selects automatically one of the options by analyzing data from the loaded Discoverable Taxonomy Set. You can change the structure views later if necessary (see also Selecting Structure Views).

7.8.2 Reading Data from Inline XBRL

Some regulators that work with XBRL may provide (or accept for submission) information in inline XBRL format, as an alternative (or in addition) to plain XBRL. Inline XBRL, or iXBRL, refers to machine-readable XBRL content embedded into HTML documents. This makes it possible to display XBRL data in a human-readable format in a Web browser. For the list of specifications related to iXBRL, see https://specifications.xbrl.org/spec-group-index-inline-xbrl.html.
In MapForce, files that contain inline XBRL (such as .htm, .html, or .xhtml files) can be added to the mapping as a data source. That is, MapForce can read data from Inline XBRL instance files for the purpose of mapping it to any supported target format. Note that writing data to inline XBRL instances is not supported.

Prerequisites

- The mapping transformation language must be set to BUILT-IN (see Selecting a Transformation Language); other languages are not supported. With the BUILT-IN transformation language, mappings can be previewed either directly in MapForce, or they can be compiled to server execution files and run by MapForce Server (see Compiling Mappings to MapForce Server Execution Files).
- The taxonomy of the Inline XBRL file might be either a URL address or a reference to a local file. In the latter case, if the mapping is run by MapForce Server, note the following additional requirements:
  - The taxonomy file must be in the same directory as the .mfx (mapping execution) file.
  - The Inline XBRL file must point to the taxonomy file using a relative path, for example `<link:schemaRef xlink:type="simple" xlink:href="taxonomy.xsd" />`.
  - The Component Settings dialog box (see XBRL Component Settings) must also refer to the taxonomy file using a relative path (for example, "taxonomy.xsd").
- The source Inline XBRL file must contain a single XBRL instance; multiple instances are not supported.

Adding Inline XBRL instance files to the mapping

2. Do one of the following:
   a. If you have the XBRL taxonomy, browse for the XBRL taxonomy (typically, a file with .xsd extension). This must be a valid taxonomy for the Inline XBRL instance file. You can also specify a URL as taxonomy; to do this, click the Switch to URL button on the Browse dialog box (see also Adding Components from a URL).
   b. If you don't have the XBRL taxonomy file or URL, browse for the Inline XBRL instance file instead. In this case, MapForce will attempt to determine and load the corresponding XBRL taxonomy automatically.
3. When prompted to supply an instance, browse for the HTML file that contains inline XBRL (typically, a file with .html, .htm or .xhtml extension).
4. When prompted to select a structured view, leave the default option as is. You can change the structure view(s) at any time later (see Selecting Structure Views).
5. Click OK.

At this stage, the structure of the Inline XBRL content becomes available in the component. In case there are validation errors, they are displayed in the Messages window. Otherwise, you can draw connections from the inline XBRL component to a target structure such as XML, text, database, and so on, in a similar fashion as with standard XBRL components.

The "Extract Inline XBRL" check box

When the Extract Inline XBRL check box on the XBRL Component Settings dialog box is selected, the .html or .xhtml file specified as input will be treated as Inline XBRL. Consequently, MapForce will extract the Inline XBRL content from the input file and validate it when you close the dialog box. This check box is typically selected automatically when you add an Inline XBRL instance file to the mapping. To check whether this option is enabled, double-click the XBRL component header on the mapping (alternatively, right-click the XBRL component header, and
7.8.3 XBRL Taxonomy Packages

An XBRL Taxonomy Package is a ZIP archive that contains an offline copy of an XBRL taxonomy. The advantages of an XBRL taxonomy package (if one is available from the taxonomy provider) is that XBRL resources such as schemas are resolved from local (offline) taxonomy files rather than being retrieved from the Internet. The structure and format of XBRL taxonomy packages is laid out in the Taxonomy Packages 1.0 Recommendation (https://www.xbrl.org/Specification/taxonomy-package/REC-2016-04-19/taxonomy-package-REC-2016-04-19.html).

If your XBRL taxonomy provider offers for download XBRL taxonomy packages conforming to the Recommendation above, you can add such packages to MapForce. This affects XBRL validation and processing (in the sense that certain URLs such as XBRL schema references will be redirected to local files included in the taxonomy package). Taxonomy packages also enable you to view all entry points of the XBRL taxonomy directly from MapForce, and improve the performance of XBRL validation.

**Note:** The following Altova desktop applications support adding XBRL Taxonomy Packages: XMLSpy, MapForce, and StyleVision. The XBRL taxonomy package list is common to all these applications. If you edit the list in one application, then the modified list will be displayed (and XBRL validation and processing will be affected) in other applications as well.

**How it works**

Each XBRL taxonomy package contains an internal catalog file (`catalog.xml`). This catalog file defines the URIs that should be redirected (remapped) to local resources. Catalog files conform to a restricted subset of the XML Catalog specification (https://www.oasis-open.org/committees/download.php/14809/xml-catalogs.html). A similar mechanism also exists in MapForce to remap URLs to local resources (see Catalog Files).

After you add one or more XBRL taxonomy packages (see instructions below), they influence, with immediate effect, how MapForce resolves resources for XBRL validation. For example, if you add an XBRL instance to the mapping, its schema will be loaded from the local taxonomy package instead of being retrieved from the Internet (assuming that the taxonomy package is active and its `catalog.xml` file remaps the schema URI to a local resource). Consequently, this
Data Sources and Targets

will affect all subsequent XBRL validation and processing.

Caution: Package catalogs might redirect to incompatible resources
A resource pointed to by the catalog file of an active package will be used for all MapForce operations that require that resource. An example of such a resource would be XML Schema, which is used for XML validation as well as XBRL validation. If the offline resource located by the package's catalog file is incompatible with your existing environment, then errors might result. In this case, deactivate the taxonomy package and contact the creators of the package with the error information.

Adding a taxonomy package

1. Download the XBRL taxonomy package (.zip archive) from the provider and save it to a local directory that is not likely to change frequently. Do not unzip the downloaded archive.


2. On the Tools menu, click Options | XBRL | Taxonomy Packages.

3. Click Edit.

4. Click Add Taxonomy and browse for the .zip taxonomy package. To select multiple packages, hold the Shift key pressed.

The list of added XBRL taxonomy packages is displayed in the dialog box, as a tree of two levels. The first level indicates the taxonomy; the second level shows the packages of that taxonomy. When a taxonomy package is selected in the XBRL Taxonomy Packages dialog, its details (including its offline location) are displayed in the dialog's lower pane.
The check box to the left of a taxonomy entry indicates whether that taxonomy is active. By default, a newly added taxonomy is active. To deactivate a taxonomy, uncheck its check box. Deactivation is useful if you encounter problems with the package or wish to switch between two versions of a taxonomy.

If you wish to add an additional package to an existing taxonomy, select the taxonomy entry, and then click **Add Packages**. The added package will be displayed at the second level of that taxonomy. To remove a package, select it and click **Remove** (note this does not delete the actual taxonomy files from the disk).

### 7.8.4 About XBRL Component Items

The following table gives an overview of the component items (nodes) which are typically part of an XBRL component in MapForce. Note that some of the nodes are only available when you selected the relevant structure view (see Selecting Structure Views). For example, the nodes specific to the table linkbase are not visible if you choose to display XBRL data only from the presentation and definition linkbases.

<table>
<thead>
<tr>
<th>Component Item</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract item</td>
<td><img src="icon.png" alt="Icon" /></td>
<td>Abstract items are used to organize related facts. They are either defined within the presentation linkbase or within domain member networks of definition linkbases. They do</td>
</tr>
<tr>
<td>Component Item</td>
<td>Icon</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>not actually exist in the XBRL instance file; they allow grouping of facts in an intuitive way.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Breakdown</strong></td>
<td>[Icon]</td>
<td>As defined in the Table Linkbase Specification 1.0, §5.4.</td>
</tr>
<tr>
<td><strong>Context</strong></td>
<td>[Icon]</td>
<td>A context node is a container for all related business facts. Context nodes inside hypercubes manage their context elements and dimensions automatically. The xbrli:context node which is a child of the XBRL root element is used for manual dimension handling.</td>
</tr>
<tr>
<td><strong>Dimension</strong></td>
<td>[Icon]</td>
<td>Dimensions (XBRL Dimensions Specification 1.0, §2.5) are used to structure contextual information for business facts. Dimensions are defined in the taxonomy within a hypercube.</td>
</tr>
<tr>
<td><strong>Explicit member</strong></td>
<td>[Icon]</td>
<td>An explicit member is a member of a dimension which is defined by an enumeration of QName values (see also Showing Dimensions in a Component).</td>
</tr>
<tr>
<td><strong>Explicit member value</strong></td>
<td>[Icon]</td>
<td>The value of an explicit member.</td>
</tr>
<tr>
<td><strong>Fact</strong></td>
<td>[Icon]</td>
<td>Facts (XBRL Specification 2.1, §1.4) are the values of the XBRL items. They can be of the following types:</td>
</tr>
<tr>
<td>- Monetary items</td>
<td>[Icon]</td>
<td></td>
</tr>
<tr>
<td>- String items</td>
<td>[Icon]</td>
<td></td>
</tr>
<tr>
<td>- Numeric items</td>
<td>[Icon]</td>
<td></td>
</tr>
<tr>
<td>- General items</td>
<td>[Icon]</td>
<td></td>
</tr>
<tr>
<td>- Shares items</td>
<td>[Icon]</td>
<td></td>
</tr>
<tr>
<td><strong>Footnote</strong></td>
<td>[Icon]</td>
<td>Footnotes allow you to assign additional text information to facts.</td>
</tr>
<tr>
<td><strong>Hypercube</strong></td>
<td>[Icon]</td>
<td>Hypercubes (XBRL Dimensions Specification 1.0, §2.2) use information from the definition linkbase and the presentation linkbase to hierarchically structure dimensions, contexts and related XBRL concepts. See also Working with XBRL Hypercubes.</td>
</tr>
<tr>
<td><strong>Root element</strong></td>
<td>[Icon]</td>
<td>The xbrli:xbrl element is instance root element of every XBRL component.</td>
</tr>
<tr>
<td><strong>Rule node</strong></td>
<td>[Icon]</td>
<td>As defined in the Table Linkbase Specification 1.0, §6.6.</td>
</tr>
<tr>
<td><strong>Structural node</strong></td>
<td>[Icon]</td>
<td>As defined in the Table Linkbase Specification 1.0, §5.5.</td>
</tr>
<tr>
<td><strong>Table</strong></td>
<td>[Icon]</td>
<td>As defined in the Table Linkbase Specification 1.0, §5.1. MapForce displays the table structures in a hierarchy (see Working with XBRL Tables).</td>
</tr>
<tr>
<td>Component Item</td>
<td>Icon</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Table set</td>
<td><img src="image" alt="Table Icon" /></td>
<td>As defined in the Table Linkbase Specification 1.0, §5.2.</td>
</tr>
<tr>
<td>Tuple</td>
<td><img src="image" alt="Tuple Icon" /></td>
<td>Tuples (XBRL Specification 2.1, §1.4) are complex elements containing facts or other tuples as members.</td>
</tr>
<tr>
<td>Typed member</td>
<td><img src="image" alt="Typed Icon" /></td>
<td>Typed members are members that are defined by an XML schema element (see <a href="https://www.xbrl.org/specification/dimensions/rec-2012-01-25/dimensions-rec-2006-09-18+corrected-errata-2012-01-25-clean.html#sec-typed-dimensions">https://www.xbrl.org/specification/dimensions/rec-2012-01-25/dimensions-rec-2006-09-18+corrected-errata-2012-01-25-clean.html#sec-typed-dimensions</a>).</td>
</tr>
<tr>
<td>Unit</td>
<td><img src="image" alt="Unit Icon" /></td>
<td>The unit element (&lt;)xbrl:unit contains units to which XBRL items refer. It is mandatory to define a value for the (<a href="">xbrl:unit\</a> element when mapping data. For example, for dollars, (&lt;)UnitID is (\text{usd}) and (&lt;)Measure is (\text{iso4217:USD}). See also Working with XBRL Defaults.</td>
</tr>
<tr>
<td>View</td>
<td><img src="image" alt="View Icon" /></td>
<td>Views represent extended link roles from the definition and presentation linkbase of an XBRL taxonomy.</td>
</tr>
</tbody>
</table>

Note that the icon \(<\) is generic and is used with a variety of node types, including the XBRL root element.

Additionally, the following graphical user interface elements can appear on XBRL components if required by the context.

<table>
<thead>
<tr>
<th>GUI element</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional information</td>
<td><img src="image" alt="Additional Info Icon" /></td>
<td>This icon accompanies items that have an associated message (such as errors). Click on the icon to display additional information in the Messages pane.</td>
</tr>
<tr>
<td>Error</td>
<td><img src="image" alt="Error Icon" /></td>
<td>This icon indicates an error message.</td>
</tr>
<tr>
<td>Show Context menu</td>
<td><img src="image" alt="Menu Icon" /></td>
<td>This icon accompanies items that have additional context menu options available for selection.</td>
</tr>
<tr>
<td>Unknown</td>
<td><img src="image" alt="Unknown Icon" /></td>
<td>This icon accompanies items that could not be resolved by MapForce due to invalid namespace references, or when the referenced item does not exist in the taxonomy.</td>
</tr>
</tbody>
</table>

### 7.8.5 Selecting Structure Views

MapForce provides various ways to display the XBRL structure inside a component, by means of so-called "structure views". You can select the XBRL structure views either when you add an XBRL document to the mapping area (see Adding XBRL Files as Mapping Components), or at any time later.
To select the structure views to be shown on the XBRL component:

1. On the root element of the XBRL component, click the **Show Context Menu** button, and then **Select Structure Views**.

2. Select one or more structure views, and then click **OK**.

The available structure views are as follows:

<table>
<thead>
<tr>
<th>Structure View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tables from table linkbase</strong></td>
<td>Shows the tables defined in the table linkbase of the taxonomy. This option is disabled if the taxonomy does not contain table definitions.</td>
</tr>
<tr>
<td></td>
<td>This view is conditional. To proceed, select either this check box, or the <strong>Views from presentation and definition linkbases</strong> check box.</td>
</tr>
<tr>
<td><strong>Views from presentation and definition linkbases</strong></td>
<td>Shows data from the presentation and definition linkbases. The presentation linkbase includes hierarchies and the definition linkbase includes extended link roles with hypercubes and dimensions.</td>
</tr>
<tr>
<td></td>
<td>This view is conditional. To proceed, select either this check box, or the <strong>Table from table linkbase</strong> check box.</td>
</tr>
<tr>
<td><strong>All concepts (with context management)</strong></td>
<td>Shows the additional node &quot;All concepts&quot; in the hierarchical structure. This node contains the hypercube &quot;Dimensionless&quot; which enables mappings of all concepts of the taxonomy regardless whether they are reported within hypercubes by means of the two default dimensions identifier and period.</td>
</tr>
</tbody>
</table>
handling is provided by the context node which contains all XBRL concepts of the taxonomy as children. Abstract items are not shown.
This view is optional to display.

**All concepts (raw)**
Shows the additional node "All concepts (raw)" which provides access to all facts of the XBRL instance without any support for automatic context handling or dimension handling. It models the raw XML structure of the concepts in the instance file. The contextRef attribute has to be mapped manually when mapping these items.
This view is optional to display.

**Notes**
- When reading data from an XBRL component (that is, if the XBRL component is a source component), you can choose any combination of structure views from the context menu. However, when writing data to XBRL, if the Tables from table linkbase view is selected, it is not possible, for technical reasons, to write data to any other view except this one.
- The Tables from table linkbase view requires that "Built-in" is set as transformation language (see also Selecting a transformation language).

### 7.8.6 XBRL Component Settings

Once you added an XBRL document to the mapping area (see Adding XBRL Files as Mapping Components), you can configure the settings applicable to the component from the Component Settings dialog box.

**To open the Component Settings dialog box:**
- Double-click the header of the XBRL component, or right-click it and select Properties.
Data Sources and Targets

The Component Settings dialog box includes the following settings.

<table>
<thead>
<tr>
<th>Component name</th>
<th>The component name is automatically generated when you create the component. However, you can change the name at any time. The component name can contain spaces and full stop</th>
</tr>
</thead>
</table>
characters. It may not contain slashes, backslashes, colons, double quotes, leading or trailing spaces. If you want to change the name of the component, be aware of the following:

- If you intend to deploy the mapping to FlowForce Server, the component name must be unique.
- It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line.

### Taxonomy

Specifies the file name and path of the main taxonomy file.

To change the location of the file, click **Browse** and select the new file.

To edit the file in your XBRL editor (for example, XMLSpy), click **Edit**.

### Input XBRL File

Specifies the file name of the input XBRL instance for the currently selected XBRL component. This field is filled automatically when you first insert the XBRL component and assign an XBRL instance file.

To change the location of the file, click **Browse** and select the new file.

To edit the file in your XBRL editor (for example, XMLSpy), click **Edit**.

### Extract Inline XBRL

When this check box is selected, the Inline XBRL content will be extracted from the .html, .htm, or .xhtml file specified as input.

### Output XBRL File

Specifies the file name and path where the XBRL target instance file is placed, if the component is used as a mapping target.

The entry from the Input XBRL File field is automatically copied to this field when you assign the input XBRL instance file. If you do not assign an input XBRL instance file to the component, then this field contains the file name and path of the taxonomy file and the extension "xml".

To change the location of the file, click **Browse** and select the new file.

To edit the file in your XBRL editor (for example, XMLSpy), click **Edit**.

### Taxonomy schema reference

The path of the referenced/associated taxonomy schema file relative to the MFD file. Use this field if you want to specify a
<table>
<thead>
<tr>
<th>Data Sources and Targets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Sources and Targets</td>
</tr>
</tbody>
</table>

### XBRL

**851**

### Setting XBRL Preferences

In MapForce, you can configure the XBRL-specific settings as follows:

- View additional information about component items (see [Enabling Tips and Annotations](#)).
- Configure the general (application-wide) XBRL settings (see [General XBRL Options](#)).

---

#### 7.8.7 Setting XBRL Preferences

| **Cast values to target types** | Allows you to define if the target XML schema types should be used when mapping, or if all data mapped to the target component should be treated as string values. By default, this setting is enabled. |
| **Pretty print output** | Reformats the output XBRL document to give it a structured look. Each child node is offset from its parent by a single tab character. |
| **Create digital signature** | Allows you to add a digital signature to the XBRL output instance file. Adding a digital signature is possible when you select "Built-in" as transformation language (see also [Digital Signatures](#)). |
| **Encoding** | Allows you specify the following settings of the output instance file: |
| | • Encoding name |
| | • Byte order |
| | • Whether the byte order mark (BOM) character should be included. |
| **StyleVision Power Stylesheet file** | This option allows you to select or create an Altova StyleVision stylesheet file. Such files enable you to output data from the XBRL instance file to a variety of formats suitable for reporting, such as HTML, RTF, and others. |
| **Save all file paths relative to MFD file** | When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. This setting affects the following files: |
| | • The XBRL taxonomy file |
| | • The XBRL input file |
| | • The XBRL output file |
| | • The StyleVision stylesheet file |

See also [Using Relative Paths on a Component](#).
7.8.7.1  Enabling Tips and Annotations

When tips are enabled, you can view additional information about each component item if you place the mouse cursor over it, as shown below.

You can also switch on or off the annotation text (if it exists). When enabled, annotations are displayed to the right of the item inside the component. For example, in the screen shot below, the annotation of the `xbrli:xbrl` root element is "XBRL instance root element".

To enable or disable tips, do one of the following:

- On the View menu, click Show Tips.
- Click the Show Tips ( ) toolbar button.

To enable or disable annotations, do one of the following:

- On the View menu, click Show Annotations.
- Click the Show Annotations ( ) toolbar button.

7.8.7.2  General XBRL Options

MapForce enables you to configure the following general (application-wide) XBRL settings:

- Set the label language of XBRL items and their annotations
• Set the preferred label roles for XBRL item names
• Set the specific type of label roles of annotations for XBRL items
• Add custom XBRL taxonomy packages (see XBRL Taxonomy Packages).

All of the above settings can be configured from the Options dialog box, XBRL section.

**To change the general XBRL options:**

1. On the **View** menu, click **XBRL Display Options**.
2. Do one of the following:
   - To change the label language of XBRL items and their annotations, click **XBRL | General**.
   - To change the preferred label roles and annotations applicable for items in the following structured views, click **XBRL | Concept Labels**:
     - Views from presentation and definition link bases
     - All concepts (with context management)
     - All Concepts (raw)
To change the preferred label roles and annotations applicable for items in the **Tables from table linkbase** view, click **XBRL | Generic Labels**.

To add or change XBRL taxonomy packages, click **XBRL | Taxonomy Packages** (see [XBRL Taxonomy Packages](#)).
7.8.8 Working with XBRL Defaults

Defaults are a powerful way to assign values to attributes in the XBRL component without having explicit mapping connections to all of them. For example, if you assign a constant unit identifier to the attribute `unitRef` of the `monetaryItemType` item within the Defaults hierarchy (as shown in the following screen shot), this assigns the default to every monetary XBRL item unit identifier, except where its input is mapped explicitly by some other value. The screen shot below also illustrates the use of the default value for the `xml:lang` attribute which defines the language of a footnote.

By default, the "Defaults" node is visible in a new XBRL component; however, you can hide it if you do not need to map to Defaults. You can display the "Defaults" node at the root level (for the whole XBRL document), or for individual nodes at any hierarchical level of the XBRL component. Note that, since Defaults can be defined at any level in the XBRL structure, different subtrees can have different default values.

You can also map to aspect value defaults from the table linkbase, if your XBRL component uses the table linkbase view instead of the presentation/definition linkbase views. In the screen shot below, the `xbrli:unit`, `xbrli:identifier` and `xbrli:period` elements are aspect value defaults that you can use when mapping to XBRL tables.
To display the "Defaults" node for a particular item:

1. Right-click the item for which you want to display the default units, and select **XBRL | Show defaults.** (Alternatively, click the **Show Context Menu** button if available for the node, and then click **Show Defaults.**)

   This inserts a Defaults item to which you can connect your own default values for the various item types.

Replacing or de-activating a default value

If a default value has been defined for some concept attribute, e.g. **decimals**, it is possible to remove this setting locally for each concept, by using the function **set-empty**. For more information about functions, see **Functions.**
In the example mapping below, the `set-empty` function "deactivates" the default value "-6" for the monetary item "Segment Reporting Information, Revenue". This item will now be reported with the precision of "2" mapped to the "precision" attribute, while the other item, "Segment Reporting Information, Operating Income Loss" will be reported with the default decimals value of "-6".

Consequently, in the mapping output, these two items would look as illustrated below:

```
<us-gaap:SegmentReportingInformationRevenue contextRef="ctx1" unitRef="USD" precision="2">23000988</us-gaap:SegmentReportingInformationRevenue>
<us-gaap:SegmentReportingInformationOperatingIncomeLoss contextRef="ctx1" unitRef="USD" decimals="-6">5605900</us-gaap:SegmentReportingInformationOperatingIncomeLoss>
```
Context handling

The hierarchical structure within XBRL components allows automatic context handling. The generation of the xbrli:context in XBRL output instances is done automatically when reporting related concepts.

The value of the attribute id of a specific context in an XBRL instance is the value of the attribute contextRef in each related XBRL concept. MapForce automatically numbers all created contexts in an output instance.

Customization is possible by assigning text as a prefix into the node "context id (prefix)" under "generated ids".

For example, mapping the constant value "context_{" as a default prefix creates consecutive context-ids in the output instance having the values "context_1", "context_2", "context_3", and so on.

If no default value is defined, MapForce will create all context-ids with the prefix "ctx_{".

If the output XBRL instance contains footnotes, the related concepts must have concept IDs to link to the automatically generated footnote links. These attributes are automatically generated. The item "concept id (prefix)" can be mapped to determine such a prefix.

If the prefix is not mapped within the XBRL component, MapForce will create all concept IDs with the prefix "fact_{".

7.8.9 Working with XBRL Hypercubes

Hypercubes enable automatic context handling. XBRL hypercubes can be of the following types:

- Defined by the taxonomy, e.g., Statement [Table]
- Generated by MapForce to simplify the default dimensions identifier and period, derived from the Presentation linkbase, e.g., Dimensionless (presentation)
- Generated by MapForce within the All concept node

Every hypercube contains two default dimensions, identifier and period, that support the easy reading/writing of these two elements for each context. Additionally defined dimensions in the taxonomy are automatically related to the context elements xbrli:segment and xbrli:scenario.

Hypercubes denoted as "Dimensionless (presentation)" use both default dimensions.
hierarchical order of concepts shown within its context node is taken from the presentation linkbase.

"Dimensionless" hypercube items also use both default dimensions, but do not have any hierarchical concept order and show only the raw list of all concepts defined in the taxonomy.

All other hypercubes are defined within the taxonomy and are designated according the name defined in the Label linkbase of the taxonomy.

Hypercubes as well as their dimensions (or Axes), each have a small icon which opens a pop-up menu allowing you to define the presentation of each of the dimensions in the component. The screen shot below shows a sample taxonomy file which contains both generated hypercubes and hypercubes defined by the taxonomy.

**Note:** MapForce shows all hypercubes which have reportable concepts. If one of the related hypercube dimensions has no domain, it is not shown in the XBRL component.

### 7.8.9.1 Showing Dimensions in a Component

Dimension items in XBRL refer either to **explicit** or **typed** dimension values in the instance. The annotation of each dimension item shows in brackets whether the dimension is reported in the context elements `xbrli:segment` or `xbrli:scenario`.

**Typed** dimension items show the elements of their XML Schema type as children. Their values can be directly mapped.

**Explicit** dimensions in an XBRL taxonomy have a value of type `xs:QName` from a certain domain.
This comprises of the XBRL domain member values and the value of the XBRL domain item itself. Explicit dimensions can be displayed in two different modes, depending on the mapping requirements and the other component/structure you are mapping to or from XBRL.

Initially, the explicit dimension is displayed with a single child node and can be mapped directly using this child, e.g. "Statement, Equity Components [Axis]".

This is useful (for an XBRL target component) when the dimension values can be derived from a field in the source data, e.g. a database field, or a column in an Excel table. As the source data will generally not contain the required QName datatype, MapForce can automatically create them using the value-map function (see Generating Value-Maps for Hypercube Dimensions).

To allow different mappings for the facts related to each dimension member, you can display separate nodes for every single value of the dimension domain.

**To show the dimension values in the component:**

1. Click the icon of the dimension you want to see the values for, and select "Show Dimension Values".

This changes the items visible below the dimension name. The dimension domain and member items are now visible, each with a light green icon. These are all explicit members of a domain which is shown by the "ex" prefix in the item icon.
Each explicit member will now contain the same substructure, allowing different mappings for each.

When the output of a concept is mapped, only those values are used for which the related context element has the appropriate dimension value, e.g. the value of "Net Income (Loss)" in the instance, is mapped only for contexts which contain the dimension value "Comprehensive Income [Member]" for the dimension "Statement, Equity Components [Axis]". There is no additional filtering required.

When writing XBRL output instances, the automatic generation of proper dimension values within the context is supported. E.g. for every reported monetary item "Net Income (Loss)"), the context node xbrli:context, acquires within its context element (xbri:segment), an element for the explicit dimension "Statement, Equity Components [Axis]" containing the value "Comprehensive Income [Member]".
7.8.9.2 Changing the Order of Dimensions

Initially, MapForce displays all dimensions of a hypercube as nested child nodes, automatically creating a hierarchy. The hierarchical order of dimensions within the hypercube can be changed to match the other (non-XBRL) side of the mapping.

Furthermore, where dimension values have to be set specifically for some concepts, MapForce is able to display a dimension, without a hierarchy, and show it as a child element of the context node.

To change the order of dimension items:

1. Click any one of the icons of the respective hypercube, and select “Change Dimension Order” entry in the popup menu.

   This opens a dialog box allowing you to reposition the various dimensions of a hypercube. Note that a hypercube has two default dimensions: identifier and period whose order in the hypercube can also be changed.

2. Click the hypercube dimension and use drag-and-drop to reposition it in the dialog box. A line appears at a position where the dimension can be dropped.
3. Click OK to close the dialog and have the dimension repositioned in the component.

To exclude a dimension from the hierarchy:

- Drag the dimension below the `xbrl:context` line, which will insert it into its context item.
7.8.9.3 Generating Value-Maps for Hypercube Explicit Dimensions

This option transforms input data of any type into a valid QName in the target component. In other words, the input string is converted into the prefixed name (QName).

To generate a value-map lookup table for the selected hypercube dimension:

1. Make sure that the dimension values are not visible for the specific hypercube dimension.
2. Click the icon of the specific dimension and select the “Generate Value-Map as input” option in the popup menu.
This opens the Value-Map Properties dialog box containing automatically generated input and output values based on the dimension default domain and domain members, as defined in the taxonomy.
3. Edit the input values if necessary, and click OK to insert the value-map component. Note that you can edit the column header text (by double-clicking the header) to make them shorter, or more descriptive if you wish.

This inserts the value-map function, showing the input and output parameter names. The output connector is automatically connected to the domain element of the target.

4. Connect the source item that contains the input data to be transformed, to the input parameter of the function.

For a general example of how to use the Value-Map function, see Using Value-Maps.

### 7.8.10 Working with XBRL Tables

If your XBRL taxonomy references the table linkbase (see Table Linkbase Specification 1.0), MapForce can be configured to display the node types applicable to the table linkbase (such as tables or breakdowns). This enables you to map data to or from facts displayed in rendered tables.
To show the "Tables from table linkbase" view on the XBRL component:

1. On the Insert menu, click XBRL Document and browse for the XBRL instance file, see also Adding XBRL Files as Mapping Components.
2. On the root element of the XBRL component, click the Show Context Menu ( ) button, and then Select Structure Views.
3. Select the Tables from table linkbase check box, and click OK.

For information about other views, see Selecting Structure Views.

Notes
- When reading data from an XBRL component (that is, if the XBRL component is a source component), you can choose any combination of structure views from the context menu . However, when writing data to XBRL, if the Tables from table linkbase view is selected, it is not possible, for technical reasons, to write data to any other view except this one.
- The Tables from table linkbase view requires that "Built-in" is set as transformation language (see also Selecting a transformation language).

To render information from the table linkbase, MapForce normally uses the structural model, with the following exceptions:
- Merged rule nodes are visible
- Roll-up nodes without siblings do not form a separate hierarchy level.

The structural model is one of the three data models defined by the Table Linkbase Specification 1.0, §4. To identify the parts of the structural model referenced by the mapping, MapForce also uses information from the definition model.

Since a MapForce component structure is one-dimensional, whereas tables can have up to three dimensions (x, y, z), MapForce displays table dimensions by nesting all breakdowns within each
other, in the following default order: z, y, x. As such, the purpose of breakdowns nodes in MapForce is to inform you where each breakdown starts.

This section includes instructions on how to work with entities from the table linkbase, as follows:

- Showing or Hiding Breakdowns
- Changing the Order of Breakdowns
- Working with Parameters

### 7.8.10.1 Showing or Hiding Breakdowns

In XBRL documents that reference the table linkbase, you can select whether to view or hide breakdowns in the component.

**To show or hide breakdowns:**

1. In the XBRL component, locate a node that contains the Show Context Menu button to the right (for example, the root node).
2. Click the Show Context Menu button, and then click Show Breakdowns.

### 7.8.10.2 Changing the Order of Breakdowns

In XBRL documents that reference the table linkbase, you can change the order of breakdowns.

**To change the order of breakdowns:**

1. On the XBRL component, locate the node for which you want to change the breakdown order.
2. Click the Show Context Menu button to the right of the node, and then click Change Breakdown Order. (If the Change Breakdown Order menu option is not available, this means that this option is not meaningful for the selected node.)
3. Drag the breakdowns to the desired location (for example, in the following screen shot, you can drag the y breakdown on top of the x breakdown).

4. Click **OK**.

### 7.8.10.3 Working with Parameters

In XBRL documents that reference the table linkbase and contain parameters (Table Linkbase Specification 1.0, §5.3), you can change the parameter values. If supported by the context, you can also append new parameter values to existing parameters.
You can change parameter values from the XBRL Table Parameters dialog box (see instructions below). This dialog box displays parameters defined both at global level (anywhere in the Discoverable Taxonomy Set) and in individual tables. When you change any parameter value and close the dialog box, MapForce attempts to regenerate all tables in the component and merge the structure tree nodes. If the structure cannot be regenerated due to missing parameters, an error node is displayed in the relevant node of the component. For more detailed information about the error, check the Messages window.

To change parameter values:

1. Click the Show Context Menu (/button to the right of the root node, and then click Define Parameters. If the Define Parameters menu option is not available, the table linkbase does not contain parameters.

2. Select the parameter record, and type the new value in the Value column. Mandatory parameters have a red exclamation sign (ombok) icon.
To append parameter values to existing parameters:

- Select the parameter record, and then click the **Add parameter value** button under the Value column. Note that appending a value is possible only if the parameter type supports adding multiple values.

To remove an appended value:

- Click the **Remove Value** button next to the value you want to remove.

**Prefix to namespace mappings**

Table parameter names consist of an optional namespace and a local name (that is, they are QNames and take the form `namespace_prefix:localname`). MapForce detects any such mappings automatically and displays them in the lower area of the XBRL Table Parameters dialog box. For example, in the image below, prefix `find` maps to [http://eurofiling.info/xbrl/ext/filling-indicators](http://eurofiling.info/xbrl/ext/filling-indicators).
If you need MapForce to parse a parameter prefix according to some custom namespace, you can change the existing prefix-namespace pair, or add a custom one. To create name-prefix maps, click the **Insert** or **Append** buttons. Note that this does not influence prefix names of XBRL instance items if your mapping writes data to an XBRL instance. This merely serves as an indication to MapForce how to parse and regenerate the XBRL structure after you close the dialog box.

### 7.8.11 XBRL Mapping Examples

This section includes the following XBRL mapping examples:

- **Microsoft Access to XBRL** (shows how data from a Microsoft Access database is mapped to an XBRL taxonomy, producing a valid XBRL instance file)
- **Microsoft Excel to XBRL** (shows how Microsoft Excel spreadsheet data is mapped to a taxonomy, producing a valid XBRL instance file).
7.8.11.1  Microsoft Access to XBRL

This example is available as DB_to_XBRL.mfd in the ..\MapForceExamples folder, and uses various filters and functions to extract the database data.

The taxonomy nanonull.xsd is derived from US:GAAP. The mapping creates an XBRL output instance which contains all contexts, concepts, units and footnotes for one Disclosure and three Statements.

The report "091 - Disclosure - Segment Revenue and Operating Income" shows how MapForce can map dimension values. The hypercube "us-gaap:ScheduleOfSegmentReportingInformationBySegmentTable" contains an explicit dimension "us-gaap:StatementOperatingActivitiesSegmentAxis".

Its domain has been extended in the taxonomy by the three dimension values "nanonull:USA", "nanonull:Europe" and "nanonull:Asia". The mapping shows how a value-map maps the values of the database column "Name" of the table "Region" to the required dimension values of type QName.

The report "106 - Statement - Nanonull and Consolidated Statement of Cash Flows" illustrates how MapForce can be used to write facts into the output instance which relates to both duration and instant periods.

As the mapping shows, proper reporting of facts such as "Cash and cash equivalents at beginning (end) of period" can be achieved by duplicating the period item in the hierarchy structure.

The mapping defines two units in the output instance, "USD" and "perShareItemType". The xbrli:unit element must be duplicated to do this. The related measure elements are created using the functions "xbrl-measure-currency" and "xbrl-measure-shares" from the XBRL library.

The facts in the database tables have been split up depending on whether they relate to an instant or duration period.

- The table FactsForInstantPeriod is a flat table of values.
- The table FactsForDuration is hierarchical and each fact it contains, relates to a specific PeriodID as well as a RegionID.

The Period table uses FromDate and ToDate fields to define the start and end period dates; while the Region table relates each of the facts to a specific region, i.e. Asia, Europe or USA.
Mandatory XBRL items needed in a XBRL instance file:

- `unitRef` and either `decimals` or `precision` in monetary concepts
- `xbrli:identifier` and `scheme` of the identifier dimension
- `xbrli:period` and either `xbrli:instant` or `xbrli:startDate/xbrli:endDate` elements
- `xbrli:id` and `xbrli:measure` in the `xbrli:unit` element

See also Microsoft Excel to XBRL.

### 7.8.11.2 Microsoft Excel to XBRL

This example shows how data from an Excel sheet are mapped to an XBRL taxonomy to generate an XBRL instance file. The actual mapping design file is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\boa-balance-sheet.mfd`.

To keep the example simple, only certain columns from the "Assets" worksheet have been mapped. The result of the mapping is a valid XBRL instance document containing some Assets data for a particular instant, Dec. 31st 2012.

The XBRL taxonomy for the target component is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\Taxonomy\Nanonull.xsd`. This is a demo taxonomy intentionally simplified; unlike a real taxonomy, it contains only a few mappable XBRL items. The target XBRL component is set to display the presentation and definition linkbases (hypercubes), see Selecting Structure Views.
The Excel columns that are mapped directly to the XBRL items/facts in the taxonomy component are as follows: "Goodwill", "Intangible Assets Net Excluding Goodwill", "Other Assets", and "Premises and Equipment".

Note that the item names in the source and target components are not, and do not need to be, identical. The name of the target taxonomy item determines the name used in the resulting XBRL instance. However, having identical source and target item names does have the advantage that you can use the Autoconnect Matching Children option (see Connecting Matching Children).

To generate a valid XBRL instance file, several mandatory items must be mapped. Some mandatory items are supplied by a text file (boa-extras.txt, the middle component on the mapping) whose fields are mapped to child elements of the Defaults item in the Dimensionless (presentation) hypercube.

The Defaults item is inserted by right-clicking a hypercube dimension and selecting XBRL | Show defaults, or by clicking the icon and selecting the option there. For more information, see Working with XBRL Defaults.

The following XBRL items are mapped in order to generate a valid XBRL instance file:

- unitRef and either decimals or precision in monetary concepts
- xbrl:identifier and scheme of the identifier dimension
- `xbrli:period` and either `xbrli:instant` or `xbrli:startDate/xbrli:endDate` elements
- `xbrli:id` and `xbrli:measure` in the `xbrli:unit` element

A text file supplies the data for some of the mandatory items; you can view or change these values by right-clicking the "boa-extras" component and selecting Properties from the context menu.

A constant supplies data for the `id` and `measure` items, which are at the base of the taxonomy component. The `xbrl-measure-currency` function converts the currency code into a value suitable for the `xbrli:measure` field. For more information about MapForce functions, see the Functions chapter.

Click the Output button to run the mapping and preview the resulting XBRL instance file.

- To check the validity of the XBRL instance, click the Validate Output button of the Output toolbar. Messages or warnings are displayed in the Messages window.
- To define various settings applicable to the generated output, click the Text view settings button.
7.9 **HL7 Version 3**

Support for HL7 version 3.x is automatically included in MapForce 2019 as it is XML based.

A separate installer for the HL7 V2.2 - V2.5.1 XML Schemas and configuration files is available on the Libraries page of the Altova website (https://www.altova.com/mapforce/download/libraries). Select the Custom Setup in the installer, to only install the HL7 V3 components and XML Schemas.

Location of HL7 XML Schemas after installation:

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Directory Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit MapForce on 32-bit</td>
<td>C:\Program Files\Altova\Common2019\Schemas\hl7v3</td>
</tr>
<tr>
<td>operating system, or</td>
<td></td>
</tr>
<tr>
<td>64-bit MapForce on 64-bit</td>
<td>C:\Program Files(x86)\Altova\Common2019\Schemas\hl7v3</td>
</tr>
<tr>
<td>operating system</td>
<td></td>
</tr>
</tbody>
</table>

HL7 documents can be used as source and target components in MapForce. This data can also be mapped to any number of XML schema, database, EDI or other components.

This example, HL7V260_To_HL7V3.mfd available in the ...\MapForceExamples folder, partially maps an HL7 V2.6 document to an HL7 V3 XML-based document.

- The ADT_A28 EDI file is available in the ...\MapForceExamples folder.
- The PRPA_IN101301UV02.xsd target XSD file is available in the ...\MapForceExamples\HL7V3_Example_Schemas folder.
The resulting XML instance file is shown below.
7.10 Protocol Buffers

Binary files are a large category of not human-readable files that are proprietary in many cases, and not typically intended for consumption by data mapping software like MapForce. However, starting with 2019 release, MapForce supports mapping data to or from binary files encoded in Protocol Buffers format (https://developers.google.com/protocol-buffers/). Specifically, MapForce (or MapForce Server) can read data from such binary files and convert it to any other format supported by MapForce (for example, XML, CSV, database, and so on). Likewise, you can read data from any format supported by MapForce and write it as a Protocol Buffers binary file. In addition, you can use Protocol Buffers files in Web service requests and responses, see Adding a Web Service Call (REST-Style).

Note the following:

- To read or write data to binary files in Protocol Buffers format, the transformation language of the mapping must be set to "Built-In". Code generation in C#, C++, Java, XSLT, or XQuery is not supported. For more information, see Selecting a Transformation Language.
- To make data exchange possible, binary files in Protocol Buffers format are accompanied by so-called .proto files. MapForce supports Protocol Buffers .proto files version 2 and 3.

A .proto file defines the structure of the encoded binary data, similar to how an XML schema describes the structure of an XML instance file. For example, the following code listing illustrates a .proto file that describes a person object:

```proto
syntax="proto3";

message Person {
  string name = 1;
  int32 id = 2;
  string email = 3;
}
```

**Example .proto file**

The .proto file makes it possible to interpret and process the corresponding binary file. Therefore, in order for MapForce to read data in Protocol Buffers format, you need the following:

1. The source binary file. This file may have an extension like .bin or .dat, or any other extension.
2. The .proto file.

To write data in Protocol Buffers format with MapForce, all you need is the .proto file. You can either write your .proto files manually, or get them from whoever expects you to create data in the respective format. In either case, when you run the mapping, MapForce (or MapForce Server) generates the binary file based on the .proto file.

Under normal circumstances, in order to create or read data from Protocol Buffers files outside MapForce, you would need to generate and write program code. With MapForce, however, you do not need to write code or generate it from .proto files—just add the .proto file to the mapping, and draw the required connections visually. Once you run the mapping, MapForce (or MapForce Server)
Server, if applicable) will read data from the source binary file(s) or generate binary files, according to the mapping design. For example, the image below illustrates a mapping that reads data from a database and writes it as multiple binary files in Protocol Buffers format.

The mapping above is discussed in more detail in Example: Write Data to Protocol Buffers.

### 7.10.1 Adding Binary Files to the Mapping

To map data to or from binary files encoded in Protocol Buffers format, you must first add their accompanying .proto file to the MapForce mapping, as shown below.

To add a .proto file as mapping component:

1. On the **Insert** menu, click **Protocol Buffers File**. Alternatively, click the **Insert Protocol Buffers File** toolbar button.
2. Do one of the following:
   a. To open a .proto file stored on the local disk, browse for the file, and click **Open**.
   b. To open a .proto file from a URL, click **Switch to URL** and enter the URL of the file in the dialog box, see Adding Components from a URL.
   c. To open a .proto file previously defined as a global resource, click **Switch to Global Resource**. For information about global resources, see Altova Global Resources.
3. A .proto file typically contains several message types. When prompted, select the message type that is the top object in the hierarchy (the so-called "root").
4. Click **OK**. A dialog box appears.

![Protocol Buffers Root Selection](image)

5. If you intend to read data from a Protocol Buffers binary file, click **Browse** and select the binary instance from which you want to read data. If you intend to write data to a Protocol Buffers file, or select an instance at a later time, click **Skip**.

If the .proto file can be read successfully, MapForce displays its structure on the mapping similar to how it does for other component types, for example:

![Protocol Buffers structure](image)

So far, you have added a binary component to the mapping; however, the mapping is not yet complete, because it does not read or write any data. For examples that illustrate complete mappings, see:

- **Example: Read Data from Protocol Buffers**
"File/Blob" button

The File/Blob button on the binary component opens a context menu with several commands that are useful in the following situations:

- **Use File Names from Component Settings** - This is the option selected by default and it is meaningful when your mapping reads or generates a single binary file. When this option is set, you can change the name of the file from the "Component Settings" dialog box, see Binary Component Settings.

- **Use Dynamic File Names Supplied by the Mapping** - This command is meaningful when your mapping should read or generate multiple binary files. When this option is set, you can no longer change the file name from the "Component Settings" dialog box. Instead, you set the file names based on some information that is coming from the mapping itself. For an example of use, see Example: Write Data to Protocol Buffers.

- **Parse Blobs to Protocol Buffers** - This command turns the component into a parsing component. This helps you parse a binary value in Protocol Buffers format so that it becomes a mappable tree structure. This works in a similar way as string parsing and serialization, see Parsing and Serializing Strings, only in this case it works not with character strings but with binary byte strings. For example, in the component below, you can connect an item of binary type to the "Blob" input on the left side. At mapping runtime, this binary value will be parsed and it will pass data onto the outputs of the mappable structure on the right side.

- **Serialize Protocol Buffers to Blobs** - This command turns the component into a serialization component. This helps you serialize a mappable tree structure to a binary value in Protocol Buffers format. For example, in the component below, you can connect items of appropriate types to the mappable structure on the left side. At mapping runtime, the structure will be serialized and it will populate the single output (of binary type) on the right side, so that you can connect it further to a target component. This may be useful when you mapping should write data as a BLOB (Binary Large Object) type to some target database. Another use is to write data to binary fields (of type "bytes") of a target Protocol Buffer component.
7.10.2 Binary Component Settings

After you add a binary component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component settings dialog box in one of the following ways:

- Select the component on the mapping, and, on the Component menu, click Properties.
- Double-click the component header.
- Right-click the component header, and then click Properties.

Component Settings dialog box
The available settings are as follows.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Component name</strong></td>
<td>The component name is automatically generated when you create the component. However, you can change the name at any time. The component name can contain spaces and full stop characters. It may not contain slashes, backslashes, colons, double quotes, leading or trailing spaces. If you want to change the name of the component, be aware of the following:</td>
</tr>
<tr>
<td></td>
<td>• If you intend to deploy the mapping to FlowForce Server, the component name must be unique.</td>
</tr>
<tr>
<td></td>
<td>• It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line.</td>
</tr>
<tr>
<td><strong>Protocol Buffers Definition File</strong></td>
<td>Specifies the name or path of the structure definition (.proto) file used by MapForce to map the actual binary data.</td>
</tr>
<tr>
<td></td>
<td>To change the structure definition file, click <strong>Browse</strong> and select the new file.</td>
</tr>
<tr>
<td><strong>Root Message Type</strong></td>
<td>A .proto file typically contains several message types. Click <strong>Choose</strong> to select the message type that should be displayed as the top item in the hierarchy (the &quot;root&quot;).</td>
</tr>
<tr>
<td><strong>Input Binary File</strong></td>
<td>Specifies the binary instance file from which MapForce will read data. This field is meaningful for a source component.</td>
</tr>
<tr>
<td></td>
<td>To change the location of the file, click <strong>Browse</strong> and select the new file.</td>
</tr>
<tr>
<td><strong>Output Binary File</strong></td>
<td>Specifies the binary instance file to which MapForce will write data. This field is meaningful for a target component.</td>
</tr>
<tr>
<td></td>
<td>To change the location of the file, click <strong>Browse</strong> and select the new file.</td>
</tr>
<tr>
<td><strong>Save all file paths relative to MFD file</strong></td>
<td>When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. See also <strong>Using Relative Paths on a Component</strong>.</td>
</tr>
</tbody>
</table>

### 7.10.3 Example: Read Data from Protocol Buffers

This example shows you how to read data from a binary file encoded in Protocol Buffers format. The source binary file is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\assets.bin`. The .proto file describing the binary file is available in the same directory and it looks as follows:
syntax="proto3";

package mapforce.demo;

message CulturalAssets {
  repeated PaintingType painting = 1;
}

message PaintingType {
  string name = 1;
  string period = 2;
  float height = 3; // in cm
  float width = 4; // in cm
  string remarks = 5;
  LocationEnum location = 6;

  enum LocationEnum {
    UNKNOWN = 0;
    MUSEUM = 1;
    TEMPLE = 2;
    PRIVATE = 3;
  }
}

assets.proto

The source binary file contains information about various cultural assets (in this example, a collection of paintings). As illustrated by the .proto file above, each painting has a **height** and **width** expressed in centimeters. There are various other fields that describe a painting, including an enumeration that specifies its location (museum, private collection, or temple). The business requirement is to extract this information to a comma-separated values (CSV) file. Also, all painting dimensions must be converted from centimeters to inch and be represented as a single string, in a format like "24 in x 56.8 in".

The mapping that extracts data according to the requirements above and produces the desired output is illustrated below. This mapping is available in the following directory: <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\. You can open the mapping and run it directly, or follow the steps below to create it from scratch.
As illustrated above, the mapping consists of a source binary component (`assets`) and a target CSV component (`PaintingInfo`). Additionally, it contains the following intermediary components:

- A value-map table responsible for converting enumeration values like 0, 1, 2, 3 to a human-readable string representation (for example, 0 = UNKNOWN, 1 = MUSEUM, and so on).
- The `ConvertCmToInch` function. As suggested by the title, this is a user-defined function that converts centimeters to inches. This function is called twice on the mapping (for `height` and `width`, respectively).
- The `concat` function. This function concatenates returns a string in the format "# in x # in", where # represents the actual height and width of the painting, in inches.

The following steps illustrate how to create the mapping above from scratch.

**Step 1: Add the source binary file**

1. On the **Insert** menu, click **Protocol Buffers File**, and browse for the following file: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\assets.proto`.
2. When prompted, select "CulturalAssets" as root message, and click **OK**.
3. Double-click the title bar of the new component. The "Binary Component Settings" dialog box opens.
4. Click **Browse** next to **Input Binary File**, and select the following file: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\assets.bin`.
Step 2: Add the target CSV component

1. On the Insert menu, click Text File.
2. Select the Use simple processing for standard CSV... check box, and click Continue. The “CSV Component Settings” dialog box opens.
3. Click the Append Field button several times to create five target fields required to store the expected data.
4. Optionally, double-click the header of each field to give it a more suggestive name.

For more information, see Setting the CSV Options.

Step 3: Add the value-map

1. On the Insert menu, click Value-Map.
2. Create an entry for each mapped record, as shown below.
The "Otherwise" condition handles the case when the binary file contains no value for the Location field. In the event this happens, the mapping will write the value "Unknown" to the target instead of no result. For more information, see Using Value-Maps.

Step 4: Create the user-defined function

1. On the Function menu, click Create User-Defined Function and call it "ConvertCmToInch". Leave all other options unchanged.
2. Drag the divide and round-precision built-in functions from the Libraries window into the function's mapping window. See also Add a Built-in Function to the Mapping.
3. Add two constants and make all connections as illustrated below, see also Add a Constant to the Mapping. The first constant supplies the decimal constant required to convert centimeters to inches according to formula, and the second one supplies the rounding precision.

Essentially, the function divides the input value by 2.54, and then rounds the result to one decimal digit, as illustrated below:
To avoid potential errors, set the data type of the input and output components to "decimal". To view or change the data type, double-click the title bar of the component, or right-click the title bar, and select Properties from the context menu.

Once the user-defined function is ready, click Return to main mapping, and drag the function from the Libraries window into the mapping to call it. In this example, it is called two times. For more information, see User-Defined Functions.

Step 5: Add the concatenation function
Start typing "concat" in the Libraries window, and then drag the concat function into the mapping. To add all the required input arguments, click Add parameter several times, see also Add or Delete Function Arguments. The constants " in" and " x " are string values. Essentially, the function returns the painting's height, followed by the string " in ", followed by " x ", followed by the painting's width, and followed by " in" again.

Running the mapping
To preview the generated file in MapForce, click the Output tab. Notice that the third CSV column contains painting dimensions in required format.
To save the mapping output to the disk:

- On the **Output** menu, click **Save Output File**.

**Automation with MapForce Server**

If you have licensed MapForce Server, you can also run the mapping at the command line, on a Linux, macOS, or Windows machine, as follows:

1. Compile the mapping to a MapForce Server execution file (.mfx) with the menu command **File | Compile to MapForce Server Execution File**, see also [Compiling Mappings to MapForce Server Execution Files](#).
2. Copy the .mfx file to the server machine.
3. Run MapForce Server with the command below.

```
mapforceserver run ReadProtocolBuffers.mfx
```

Notes:

- `mapforceserver` is the path to the MapForce Server executable as applicable for your operating system.
- Change the path to the .mfx file as applicable, or copy the .mfx to the same folder as the executable.
- The .mfx file is self-contained; the .proto file is not needed for server execution.

In server execution, you can also run mappings as an API call, or as FlowForce Server jobs, either on demand or on recurring basis. For more information, see [Automation with MapForce Server](#).

### 7.10.4 Example: Write Data to Protocol Buffers

This example shows you how to write data to binary files encoded in Protocol Buffers format. In this demo, the source data comes from a SQLite database (*Nanonull.sqlite*); however, you can use a similar approach to extract data from various other formats supported by MapForce, such as XML, EDI, JSON, and others.

The source demo database stores data about persons, their addresses, and products they ordered from a store. The business requirement is to extract order information from the database and to generate multiple binary files in Protocol Buffers format. One binary file with .dat extension
must be generated for each order. The generated file name must contain the order's unique ID, for example, Order1.dat, Order2.dat, and so on. Also, each binary file must include the order ID, the person's name and email address, date, and shipping address information. More specifically, the .proto file that describes the structure of an order looks as follows:

```protobuf
syntax="proto3";

package mapforce.demo;

message Order {
  int32 id = 1;
  string name = 2;
  string email = 3;
  DateType entry_date = 4;
  // An order can have multiple addresses (shipping, billing)
  repeated AddressType address = 5;
}

message DateType {
  // Must be from 1 to 9999
  int32 year = 1;
  // Must be from 1 to 12
  int32 month = 2;
  // Must be from 1 to 31, and valid for the year and month
  int32 day = 3;
}

message AddressType {
  string city = 1;
  string street = 2;
  int32 number = 3;
  bool shipping = 5;
  bool billing = 4;
}

orders.proto
```

Both the Nanonull.sqlite database and the orders.proto file are available in the following directory: <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\. The mapping design file that performs the transformation is called GenerateOrders.mfd and is available in the same directory. You can open the mapping and run it directly, or follow the steps below to create it from scratch.
As illustrated above, the mapping consists of a source component (the SQLite database) and a target component (the binary component). In the source database, the main table relevant for this mapping is `orders`.

Notice that table `users` is nested under table `orders`, because MapForce detected automatically a foreign key relationship between these two tables. Likewise, table `addresses` is nested under `users`. All the connections drawn from such nested tables ensure that key relationships are preserved by the mapping. For more information, see Handling Database Relationships.

On the target side, the `File<dynamic>` item illustrates that the binary component is set to generate instance files dynamically, based on information received from the mapping. To have a target component generate files dynamically, click the `File: Blob` button and select Use Dynamic File Names Supplied by the Mapping from the context menu. For more information about dynamic file names, see Processing Multiple Input or Output Files Dynamically.

In this mapping, for each `id` in the `orders` database table, a new file will be generated, as shown by the connection to the `File: <dynamic>` item. The name of each generated file is created with the help of the first `concat` function. This function joins the string “Order” with the unique ID of the order from the database, and with the extension “.dat”. Therefore, the generated files will get names like `Order1.dat`, `Order2.dat`, and so on.

The `concat` function is also called a second time by the mapping. This time it returns the name of the person, by concatenating the first and last name, and inserting a space character in between.

Finally, the “GetDate” component in the middle is a user-defined function that populates the date
fields in the binary file. This function takes as input a date expressed as string, parses it, and returns the year, month, and day separately as integer values. This conversion is required to accommodate the date format as it was defined in the .proto file. If you double-click the function's title bar on the mapping, you can view or change the definition of this function.

"GetDate" user-defined function

As illustrated above, the "GetDate" user-defined function parses the input value with the help of the \texttt{parse-dateTime} built-in function. The mask \texttt{[Y]-[M]-[D] [H]:[m]:[s]} matches the date format as it is stored in the database, for example, \texttt{2017-10-19 08:09:54}. For more information about format masks, refer to the description of the \texttt{parse-dateTime} function. Next, various built-in functions are called to extract and return the significant date parts from the parsed date.

Creating the mapping

To create the mapping above (or a similar mapping) from scratch, take the following steps:

1. Insert the source database, as described in Adding Databases to the Mapping.
2. Insert the target .proto file, see Adding Binary Files to the Mapping.
3. Configure the target component to generate file names dynamically, when the mapping runs. To do this, click the \texttt{File/Blob} button and select Use Dynamic File Names Supplied by the Mapping from the context menu. Next, connect the \texttt{File\textlangle dynamic\textrangle} item to some input that provides the file name. In this example, the file name is produced with the help of \texttt{concat} function, as mentioned above. For more information about dynamic file names, see Processing Multiple Input or Output Files Dynamically.
4. Optionally, add any MapForce built-in functions and connect their inputs. In this example, the \texttt{concat} function is used in two occurrences. For more information, see Add a Built-in Function to the Mapping. Note that, when you call built-in functions such as \texttt{concat}, it is likely that some of the input data will come from string constants. For information about how to add such constants, see Add a Constant to the Mapping.
5. Optionally, add any user-defined functions if required. These help you abstract away some of the mapping details and focus on the big picture. In this example, the "GetDate" user-defined function converts a string into date parts, as mentioned, and its implementation logic is hidden from the main mapping. For more information about creating custom functions, see User-Defined Functions.

If you are completely new to MapForce, see the Tutorials chapter for simple step-by-step examples.
Running the mapping

As outlined in the business requirement at the beginning of this example, the mapping is expected to produce multiple .dat files. To preview the generated files, click the Output tab.

As illustrated above, the Output pane displays all the generated files, and you can navigate through them by clicking the directional buttons, or by picking an item from the list. To save the generated output, do one of the following:

- On the Output menu, click Save All Output Files ( ).
- Click the Save all generated outputs ( ) toolbar button.

When you preview output binary files generated with MapForce, their structure is displayed as JSON since this format is human-readable as opposed to binary format. Viewing the output as JSON is to help you test the mapping. When you save the output file, it will be saved as a binary file, however.

Automation with MapForce Server

If you have licensed MapForce Server, you can also run the mapping at the command line, on a Linux, macOS, or Windows machine, as follows:

1. Compile the mapping to a MapForce Server execution file (.mfx) with the menu command File | Compile to MapForce Server Execution File, see also Compiling Mappings to MapForce Server Execution Files.
2. Copy the .mfx file to the server machine.
3. Since this mapping reads data from a SQLite database file, copy the `Nanonull.sqlite` database file to the same directory as the `.mfx` file on the server. For other database kinds, see Database mappings in various execution environments.

4. Run MapForce Server with the command below.

```
mapforceserver run GenerateOrders.mfx
```

Notes:

- `mapforceserver` is the path to the MapForce Server executable as applicable for your operating system.
- Change the path to the `.mfx` file as applicable, or copy the `.mfx` to the same folder as the executable.

In server execution, you can also run mappings as an API call, or as FlowForce Server jobs, either on demand or on recurring basis. For more information, see Automation with MapForce Server.
8 Functions

Functions represent a powerful way to transform data according to your specific needs. This section provides instructions on working with functions (regardless if they are built-in to MapForce, defined by you, or reused from external sources). Use the following roadmap for quick access to specific tasks related to functions:

<table>
<thead>
<tr>
<th>I want to...</th>
<th>Read this topic...</th>
</tr>
</thead>
</table>
| Add MapForce built-in functions or constants to the mapping | • Add a Built-in Function to the Mapping  
• Add a Constant to the Mapping  
• Search for a Function  
• View a Function's Type and Description  
• Add or Delete Function Arguments |
| Create my own functions in MapForce | User-Defined Functions |
| Add custom XSLT functions to MapForce | Importing Custom XSLT 1.0 or 2.0 Functions |
| Add custom .NET DLL and Java .class libraries to MapForce | • Importing Custom Java and .NET Libraries  
• Referencing Java, C# and C++ Libraries Manually |
| Write my own Java library for use with MapForce | Create a Java library |
| Write my own C# library for use with MapForce | Create a C# library |
| Write my own C++ library for use with MapForce | Create a C++ library |
| View all built-in MapForce functions, or look up the description of a specific function. | Function Library Reference |
8.1 How To...

8.1.1 Add a Built-in Function to the Mapping

MapForce includes a large number of readily available built-in functions that you can add to the mapping as described below. For reference to all available built-in functions, see Function Library Reference.

To use a function in a mapping:

1. Select the transformation language (see Selecting a transformation language). Note that the list of available functions depends on the selected transformation language.
2. Click the required function in the Libraries window and drag it to the mapping area. To filter functions by name, start typing the function name in the text box located in the lower part of the window:
Alternatively, you can also quickly add a function to the mapping as follows:

1. Double-click anywhere on the empty area of the mapping and start typing the function name. A combo box appears with the same functions as in the Libraries window, filtered by the text you entered. To see a tooltip with more details about each function, select any function in the list.
2. Select the required function, and press Enter to add it to the mapping. To close the combo box without selecting a function, press Escape, or click anywhere outside the box.

**Note**: Using the "double-click" alternative way described above, you can also add user-defined functions to the mapping.

### 8.1.2 Add a Constant to the Mapping

Constants enable you to supply custom text or numbers to the mapping. A constant's value, as the name implies, will remain the same for the duration of the mapping lifetime.

**To add a constant to the mapping**:

1. Do one of the following:
   a. On the Insert menu, click **Constant**.
   b. Right-click the mapping, and select **Insert Constant** from the context menu.

2. Enter the value of the constant, select the data type ("String", "Number", "All other"), and click **OK**.

Alternatively, you can also quickly add a constant as follows:

1. Double-click anywhere on an empty mapping area.
2. Do one of the following:
   a. To add a string constant, start typing a double quote followed by the constant value. The closing double quote is optional.
To add a numeric constant, just type the number.

3. Press Enter.

8.1.3 Search for a Function

To search for a function in the Libraries window:

1. Start typing the function name in the text box located in the lower part of the Libraries window.

By default, MapForce searches by function name and description text. If you want to exclude the function description from the search, click the down-arrow and disable the Include function descriptions option.
To cancel the search, press the Esc key or click ×.

The functions available in the Libraries window depend on the transformation language currently selected, see Selecting a Transformation Language.

To find all occurrences of a function within the currently active mapping:

- Right-click the function name in the Libraries window, and select Find All Calls from the context menu. The search results are displayed in the Messages window.

8.1.4 View a Function’s Type and Description

To view the data type of a function input or output argument:

1. Make sure that the Show tips toolbar button is enabled.
2. Move your mouse over the argument part of a function.

To view the description of a function:

1. Make sure that the Show tips toolbar button is enabled.
2. Move your mouse of the function (this works both in the Libraries pane and on the mapping area)
8.1.5 Add or Delete Function Arguments

To add or delete function arguments (for functions where that is applicable):

- Click **Add parameter** (⁺) or **Delete parameter** (⁻) next to the parameter you want to add or delete, respectively.

Dropping a connection on the symbol automatically adds the parameter and connects it.
8.2 Defaults and Node Functions

When MapForce reads or writes data, it is often the case that either the source or destination file or database has empty or null fields. To handle such cases, MapForce provides various built-in functions, if-else conditions, and other mechanisms that let you replace missing or null data with something else, or perhaps throw an exception when missing fields are encountered.

Furthermore, you may want to set a default value for multiple items simultaneously (for example, all children of an XML element). Alternatively, you may want to create a simple function that substitutes an empty value with some text (for example, "n/a"), and then apply this function to multiple items. Under normal circumstances, in order to do this, you would need to copy-paste the same function multiple times on the mapping. However, this would also add clutter to the mapping and make it more difficult to understand. As a simpler alternative, you could use defaults and node functions, which are the subject of this chapter.

Note: Defaults and node functions are supported when the target language of the mapping is BUILT-IN. Running such mappings from generated C#, C++, Java program code, or with generated XSLT/XQuery transformations is not supported. On the server side, you can execute such mappings with MapForce Server Advanced Edition.

The term "node function" means that the function applies at node level, be it an XML node or CSV, JSON, EDI, or database field. The node function may apply either to a single item or to multiple items at once. Likewise, the term "default" refers to a default value that you want to apply at node level, for either a single item or multiple items. Note that, at mapping runtime, a node function or default is called once for each item in a sequence.

Defaults and node functions are particularly useful when you want to apply the same processing logic to multiple descendant items in a structure, for example:

- Every time when an empty or null value is encountered, replace it with some other value, and do this recursively for all descendant items
- Every time when a specific value is encountered (for example, "N/A"), replace it with some other value (or with an empty string), and do this recursively for all descendant items
- Replace all database null values with empty string or custom text (or with 0, in case of numeric fields) when reading from a database table
- Trim all trailing spaces for all values that are coming from some source database
- Append a custom prefix or suffix to all values that are written to a target file or database
- Produce a null value each time when a specific value is encountered

Note: It is important to distinguish between "null" and "empty" values, since they are not the same. A null value means "nothing" (the absence of a value), whereas an empty value is typically an empty string (""). MapForce provides various ways to handle both, including (but not limited to) node functions and defaults.

8.2.1 Creating Defaults and Node Functions

You can create node functions or defaults for nearly any item (node) on the mapping. Let's call this process defining a rule.
In order to create a rule, first determine the item (node, or field) where you want to define the rule. This can be either a "leaf" item (with no descendants) or an item that has descendants. In the latter case, you can apply the rule to all descendant items as well. Moreover, you can filter descendant items on which you want the rule to apply, by data type. You can also look up qualifying items by type and name, with the help of regular expressions.

Prerequisites:

- You can create defaults or node functions either on an input side of a target component, or on the output side of a source component. To establish which side is right for your needs, see Choosing the Input or Output Side.
- Defaults and node functions require that the connection type between source and target is either "Source Driven" or Target Driven". "Copy-All" connections are not supported. Specifically, node functions and defaults are not applied to descendants of "Copy-All" connections. The node that has the "Copy-all" connection itself will apply node functions and defaults, but only if it has a simple value, for example, an XML element with simple-type content and attributes. Therefore, if you want to define a function or set a default on a node with descendants, the connection type between source and target must not be "Copy-All". To view or change the connection type, right-click the connection and select Target Driven (Standard) from the context menu. For more information, see Connection Types.
- Note that creating defaults or node functions is not supported for the "File" node. This node lets you create or read file names dynamically, see Processing Multiple Input or Output Files Dynamically.

To create a rule:

1. Right-click the item (node) of interest, and select Node Functions and Defaults | Input Node Functions and Defaults from the context menu (or Output Node Functions and Defaults, depending on the case). Alternatively, right-click a connector—in this case, MapForce will show the node function command for that side only. The Mapping pane displays a grid at the top, for example:

   ![Node functions and defaults grid](image)

   If the item where you define the rule has a parent, the parent may also have rules (node functions or defaults) defined against it. To inherit such rules, select the Inherit rules from ancestors check box. For more information about inheritance, see How Defaults and Node Functions Work.

2. Do one of the following:
   a. To add a default, click Add Default ( ).
   b. To add a function, click Add Function ( ).

   This creates a new rule (a row in the grid at the top of the Mapping pane where you can choose the criteria for this rule). Configure the rule as follows:
Apply To
Select whether the rule should apply to the current item, or to all descendant items regardless of their depth, or to direct child items only.

If the item you selected in step 1 has no descendants, then "Current item" is the only choice.

Data Type
Click the Ellipsis button and select a data type from the dialog box. The rule (default or node function) will apply only to items that have this data type (or a derived data type). For more information, see Applying Node Functions and Defaults Conditionally.

If the item you selected in step 1 has no descendants, then the item's data type is the only choice.

Default Value / Function Description
If you are defining a default, type here the default value that you wish to set for the selected item (and all descendants, if applicable). To set an empty string as default, leave this field empty.

If you are defining a function, this field is for information purpose only. It displays a summary of the function. You can define the function's body in the next step.

3. If you are defining a function, the mapping area changes to display the function's input (illustrated below as "raw_value") and output ("result"). This mapping area is a mini-mapping, and the same general rules apply here as when you define a standard mapping. For example, the body of a function could look as follows:

The node function illustrated above replaces any empty value with the value "n/a". For more information about this example, see Example: Replace Empty CSV Fields.

Note the following:

- Inside a node function, only certain MapForce components meaningful in this context are supported, such as built-in functions, variables, if-else conditions, and others. Complex structures such as XML, JSON, EDI, or databases are not supported. Adding inline user-defined functions or join components to a node function is also not supported.
- A node function can have one input parameter at most, or no parameter at all. The input parameter is always called "raw_value". Never delete the input parameter ("raw_value"), even if you don't need an input for your function; otherwise, validation errors will appear when you run the mapping. The same applies for the function's output. Should you need to restore an accidentally deleted input component, run the menu command Function |
Insert Input.

- If an input parameter ("raw_value") is already present, any subsequent input parameters that you add become metadata parameters. You can use such metadata to retrieve additional information about the current node and use it inside the function, see Supplying Node Metadata to Node Functions.
- In some cases, you might find it more convenient to create a node function as follows: drag a function from the Libraries window to an input or output connector. This makes sense only for simple functions like right-trim or uppercase.

**To exit to the main mapping:**

- Click the Go back button in the upper-left corner of the Mapping pane, or press Escape.

See also Editing and Deleting Existing Rules.

### 8.2.1.1 Choosing the Input or Output Side

Because any MapForce component always has an input and an output side, you can define a node function or a default value on either side, depending on your needs. To understand this better, let's recall how a mapping works: it first reads data from a source component (for example, a database or a file), then optionally processes it in some way (for example, using functions or filters), and finally writes data to some target component (for example, a database or a file). Considering this, you can apply node functions and set defaults at various stages:

- Immediately after data is read from the source file or database (but before it is further processed by your mapping). For example, in the mapping below the function or default is defined on the output side of the source component (notice the icon, which denotes that node functions or defaults are present):

- Immediately before data is written to the target file or database (and after it finished all intermediary processing). For example, in the mapping below, the function or default is defined on the input side of the target component:
At an intermediary stage in the mapping process. For example, if the mapping contains an intermediary variable of complex type (say, an XML structure), you could trim all values before they are supplied to the XML structure, or immediately after they are returned by the XML structure).

To summarize the above, you can define node functions either on the "input" or the "output" side of a component. Functions (or defaults) defined on the input side will process data before it enters the corresponding item on the component. Conversely, when defined on the output side, they will process data immediately after it is returned by the corresponding item. If the item where you defined the function has child items, then you can optionally propagate the default function to apply to all children as well.

8.2.2 Editing and Deleting Existing Rules

When a mapping contains rules for defaults or node functions, these are indicated by the the \fX icon (black or red color) next to nodes of interest. You can easily view, modify, or delete such rules as shown below.

To open a default or node function for viewing or editing:

1. In the main mapping, click the \fX icon (black or red color) next to the node of interest. This icon is present on any node where you previously defined a rule for defaults or node functions. For example, in the mapping below, rules are defined on the articles item of the source component, and on the rows item of the target component. You can find this demo mapping at the following path: <Documents>\Altova\MapForce2019\MapForceExamples\OrderInUSD.mfd.
2. After you clicked the \( f_R \) icon, observe the grid at the top, which contains a row for each default or node function that is defined on this node. Select the corresponding row, and make changes as required.

3. In case of functions, click the \( f_R \) button if you want to change the implementation (mapping) of the function. If the \( f_R \) button is not present, then the function is most likely defined on some ancestor. In this case, make sure that you click the \( f_R \) icon on the item where the rule is defined (see the note below).

The \( f_R \) icon merely indicates where the rule applies, not where it is defined. For more information about the meaning of icons, see How Defaults and Node Functions Work.

To delete a rule:

- Select the rule from the grid at the top, and then click the Remove function / default \( \times \) button.
To exit to the main mapping:

- Click the Go back button in the upper-left corner of the Mapping pane, or press Escape.

### 8.2.3 How Defaults and Node Functions Work

As explained in How to Create Defaults and Node Functions, you can create node functions or defaults for nearly any item (node) on the mapping. Let’s call this process defining a rule. Rules have the following important characteristics that make them extremely flexible:

- **Inheritance.** When you define a rule on an item that has descendants, the rule will be inherited by descendants by default, unless you choose to disable this option. If the item where you define the function has multiple levels of child items nested under it, you can choose to apply the rule only to direct child items, or to all descendant items.

- **Type Filtering.** MapForce applies rules conditionally, based on the data type of each item. This makes it possible, for example, to apply a certain default value (or a function) for all items of string type, and a different default (or a function) for all items of numeric type. You can also apply rules only to nodes of a specific type that match a specific name or regular expression.

The behavior described above has implications. Namely, it is important to make a difference between defining a rule and actually applying one. When you define a rule on some item, it does not necessarily mean that the rule will affect that item. The rule will apply to the item or its descendants only if the rule criteria (data type and inheritance) allow it.

To help you understand which rules are defined and which ones apply, MapForce provides the following visual clues on the mapping:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![RuleIcon]</td>
<td>This icon (black color) indicates that a rule is defined for this item, and may affect all its descendants. Click the icon to modify or delete the rule.</td>
</tr>
<tr>
<td>![RuleIcon]</td>
<td>This icon (red brick color) indicates that the item qualifies (is eligible) for a rule defined at some ancestor level. In other words, there exists a rule that applies to (and may affect) this item.</td>
</tr>
<tr>
<td>![RuleIcon]</td>
<td>This icon (bold, red brick color) indicates that a rule is defined for this item, and at the same time a rule applies to this item. This icon usually appears when a default is defined for a single node.</td>
</tr>
<tr>
<td>![RuleIcon]</td>
<td>This icon (grayed out) indicates that, even though a rule applies to this item, it is inactive. For example, this icon may appear for items that are not connected yet on the mapping.</td>
</tr>
</tbody>
</table>

**Note:** This icon is displayed only if inheritance is blocked and no other rules are defined at this node. If a rule from an ancestor does apply, the ![RuleIcon] icon has priority.
In general, when multiple node functions or defaults exist for one and the same item, keep in mind the following rule of thumb:

For any single item on the mapping, MapForce always applies only one node function and only one default, regardless of how many node functions or defaults qualify to apply for that item.

In practice, this translates as follows:

- When multiple rules exist for one and the same item, MapForce will apply to an item the rule that is closer to that item. For example, let's assume that you have defined a node function three times: on a root XML node called `Company`, on its child node called `Department`, and on the grandchild `Employee`. In this case, MapForce will apply to the `Employee` item the function defined on the `Employee` item, since it is closer. Had there been no function there, it would look up to find the function of the immediate ancestor, `Department`. If there is no function for `Department`, then it looks further up to the root node, which in this case is `Company`. Inheritance is optional; to disable it, clear the Inherit rules from ancestors check box. When this check box is cleared, the item gets the “blocked rule” icon.

- When one and the same item has multiple rules, then MapForce applies the first matching rule from the grid at the top of the Mapping pane. To change the order of rules in this grid, click a rule and then drag and drop to a new position within the grid. Note that you can drag a rule in the grid only if it is defined for the current item. You cannot change the position of inherited rules; you can only enable or disable inheritance.

To better illustrate how this works, we will use a mapping available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\MissingFields.mfd`.

As shown above, this mapping reads data from a source XML file into a target text file (fixed-length fields). In the source XML file, the element `Article` has child elements of different type: "integer", "string", and "decimal". Note that each child element is optional (minOccurs="0"). Therefore, if any of these elements does not exist in the source XML, you will want to provide a default value; otherwise, you will see empty fields in the target CSV file, for example:

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-Shirt</td>
<td>25</td>
<td>Available in all sizes</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2.3</td>
</tr>
<tr>
<td>Pants</td>
<td></td>
<td>Limited stock</td>
</tr>
</tbody>
</table>
Below we illustrate various ways to handle missing data by means of rules, along with explanations of how rules affect the mapping result. They will also help you understand or control which rule should prevail when multiple rules exist for a given item.

**Example 1: Provide defaults for all string items**

Given the mapping *MissingFields.mfd*, let’s assume that you have the following requirement: *If any child of Article is of type “string” and is missing, use “n/a” as default value.*

To satisfy this requirement, take the steps below:

1. Right-click the Article item, and select **Node Functions and Defaults | Output Node Functions and Defaults** from the context menu.
2. Click **Add default**.
3. Under **Default value**, type “n/a” and press **Enter**.

In the mapping above, the rule criteria are set as follows: *For all descendant items of Article, if the data type is "string", and if the source XML element is missing, use the default value “n/a"*. In this example, there are two items of type "string", Name and Description, so the rule will apply to both.

As stated before, the item where a rule is defined has the $\rightarrow$ icon next to it. Items where the rule will apply have the $\leftarrow$ icon. If you preview the mapping at this stage, you can see that all missing strings have now been replaced with “n/a” in the output:
Example 2: Provide defaults conditionally based on data type

Let's now assume that, in addition to defaults for string items, you must also supply a default value 0 for any item of numeric type. To satisfy this requirement, take the steps below:

1. Click the Article item.
2. Click Add default and add a second rule with the following criteria:

For all descendant items of Article, if the data type is "string", and if the source XML element is missing, use the default value "n/a"

For all descendant items of Article, if the data type is numeric, and if the source XML element is missing, use the default value "0".

Consequently, the output looks as follows:

<table>
<thead>
<tr>
<th></th>
<th>T-Shirt</th>
<th>25</th>
<th>Available in all sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>n/a</td>
<td>2.3</td>
<td>n/a</td>
</tr>
<tr>
<td>3</td>
<td>Pants</td>
<td>0</td>
<td>Limited stock</td>
</tr>
<tr>
<td>4</td>
<td>Jacket</td>
<td>57.5</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Note: The data type "numeric" is actually a type category, because it includes both the "integer" and "decimal" data types. It also includes the types "float" and "double", although such types are not present here. In this example, the rule will apply to both Number and SinglePrice elements. If you select "decimal" as data type, the rule will still apply to both Number and SinglePrice, because type "integer" derives from type "decimal", in the XML schema type hierarchy (see §3 in "XML Schema Part 2: Datatypes Second Edition", https://www.w3.org/TR/xmlschema-2). If you select "integer" as data type, however, the rule will apply only to Number.
Example 3: Block rule for a specific item

Let's now assume that you still want to apply defaults for all string and numeric items, like in the previous example. However, you do not want to set any default to the `SinglePrice` item.

To satisfy this requirement, click the item `SinglePrice`, and then clear the check box `Inherit rules from ancestors`.

In the mapping above, the item `SinglePrice` no longer inherits rules from its parent, `Article`. Therefore, a "blocked rule" icon ❌ appears next to it.

Consequently, the corresponding field still appears empty in the output:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T-Shirt</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>n/a</td>
<td>2.3</td>
</tr>
<tr>
<td>3</td>
<td>Pants</td>
<td>57.5</td>
</tr>
<tr>
<td>4</td>
<td>Jacket</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Example 4: Override inherited rule for a specific item

Let's assume that you still want to supply defaults for all string and numeric values; however, for item `SinglePrice` exclusively, you want to set a default value of 9999.

To satisfy this requirement, take the steps below:

1. Click the item `SinglePrice`.
2. Click `Add default` (⌘) and type a default value of 9999.
3. Optionally, select the `Inherit rules from ancestors` check box. This step is merely to illustrate that, in this case, the inherited rules will be overridden anyway.
Inherited rules have yellow background.

In the mapping above, there are three rules that may apply for item `SinglePrice`: two inherited ones, and a direct one. In this case, the rule defined directly on the item wins. The inherited rules will be disregarded. Therefore, the output looks as follows:

<table>
<thead>
<tr>
<th>Item</th>
<th>Price</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-Shirt</td>
<td>25</td>
<td>Available in all sizes</td>
</tr>
<tr>
<td>n/a</td>
<td>2.3</td>
<td>n/a</td>
</tr>
<tr>
<td>Pants</td>
<td>9999</td>
<td>Limited stock</td>
</tr>
<tr>
<td>Jacket</td>
<td>57.5</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Example 5: Set the priority of rules

Let's expand the previous example further and assume that you define one more rule for item `SinglePrice`, a default of `8888`. As stated before, the rule defined directly on the current item wins. However, since two rules now exist on the current item (in addition to the inherited ones), the legitimate question is, which of the two defaults will apply, `8888` or `9999`?
When multiple rules exist for the same item like in the mapping above, you can choose the winning rule manually, by dragging it up to the top of the grid. The topmost rule always wins. Therefore, the default value for \texttt{SinglePrice} will be 8888 if this rule is at the top of the grid:

\begin{verbatim}
0  T-Shirt   25    Available in all sizes
2  n/a       2.3   n/a
3  Pants     8888  Limited stock
4  Jacket    57.5  n/a
\end{verbatim}

8.2.4 Applying Node Functions and Defaults Conditionally

Whenever you create a node function or a default (a so-called "rule"), you can define it either on a item that has descendants, or on a "leaf" item. Importantly, if the item has descendants, you must specify a data type for the rule—this means that only descendants of this data type will be affected by the rule. To better understand this concept, open the following demo mapping: `<Documents>\Altova\MapForce2019\MapForceExamples\OrderInUSD.mfd`. Next, click the icon next to \texttt{Rows} item in the target component.
The mapping above reads data from an XML file and writes it to a target CSV file. In the target CSV component, a rule is defined on the input side of the `Rows` item. Notice that the `Rows` item has children of type `string` and `integer`. Since the data type of the rule is `string`, the rule will affect only items of this type. In other words, the icon is present only for items of type `string`, but not for those of type `integer`. The icon indicates that the rule is blocked for some items—this fact is not relevant for the moment, see How Defaults and Node Functions Work for more information.

**Tip:** To display the data type next to each item like in the mapping above, toggle on the **Show Data Types** toolbar button, and toggle off the **Show Annotations** toolbar button. These toggle commands are also available in the **View** menu.

To change the data type of a rule, click the **Ellipsis** button in the grid above the mapping. This opens a dialog box where you can choose the required type.
For example, if you select integer as data type, the rule will apply only to item Amount (since this is the only item of type integer). The component is now redrawn accordingly to illustrate this fact. Notice that no other items except Amount are now affected by rule since they are all of type string.
Change a rule's data type only when this is meaningful in your node function. In the demo OrderInUSD.mfd mapping, the node function is designed to process string data, not numeric data (that is, it performs concatenation of two strings). Consequently, this mapping will fail to execute if you change the node function's data type to integer.

Type inheritance
Since MapForce works not only with XML data, the data types available for selection on the "Filter Node Functions and Defaults" dialog box do not correspond exactly to XML Schema types. Some of them are in fact categories of types, meaning that they will match a larger selection of types. For example, the type string matches various other data types derived from string, such as normalizedString, token, NCName, NMTOKEN, IDREF, ENTITY, and others. Likewise, the type decimal will match the derived types integer, long, short, and others.

The hierarchy of types is according to the XML Schema W3C recommendation. For a diagram that fully illustrates this hierarchy, see §3 in "XML Schema Part 2: Datatypes Second Edition", https://www.w3.org/TR/xmlschema-2.

Advanced filtering
Optionally, you can apply rules based on even more advanced criteria:

- Apply rule if the node is of specific type AND the node name matches some custom text
- Apply rule if the node is of specific type AND the type name matches some custom text

For example, the settings below apply a rule only if the node type is string AND the node name is "Title". The image on the right illustrates an example structure where such a rule might be useful (only the node "Title" is matched therefore it gets the icon):
Alternatively, you can use a regular expression to match multiple node names or type names that qualify. For example, the settings below apply a rule to all items of type `string` whose name ends with "total". Notice that a regular expression is used. The regular expression `.+total` literally means: match one or more occurrences of any character, followed by the text "total". Also, the search is case-insensitive, which means that "SOMETOTAL" and "SomeTotal" will both be matched. The image on the right illustrates an example structure where such a rule might be useful (namely, the three "Total" items qualify for the rule and, therefore, get the $\mathfrak{f}_x$ icon):
The regular expressions dialect is the same as that of XML schema, see also [Regular Expressions](#). Note the following:

- The anchors ^ and $ are implicit and must not be entered in the Match to box.
- Case sensitivity is provided by the Match Case check box, so the i flag is not supported.
- Matching on multiple lines is not meaningful for node filtering, so the m flag is not supported.

### 8.2.5 Supplying Node Metadata to Node Functions

There might be cases when you want a node function to do something based on some information about the current node (let's call this information "node metadata"). For example, you might need a node function with the following logic: if the node name contains the word "Total", then append the dollar sign to the node value; otherwise, return the node value as is.

In the example just mentioned, "node name" is an example of node metadata. Generally speaking, "metadata" means something which describes data itself, that is, "data about data". By "node metadata", therefore, we understand miscellaneous information about the node on which the function applies, such as node name, value length or precision in case of numeric database types, and others.

The following table lists all possible metadata that you can use in a node function. Note that some metadata listed below is meaningful only for nodes of specific kind (for example, XML or database fields). Consequently, MapForce will display a warning when you attempt to use metadata that is
incompatible with the current node.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node_name</td>
<td>Provides the name of the current node. This metadata is applicable to all nodes. In case of XML, this is the name of the current element or attribute. In case of CSV, this is the name of the CSV field. In case of databases, it is the name of the table column.</td>
</tr>
<tr>
<td>node_annotation</td>
<td>Provides the annotation text displayed next to an item when you click the Show Annotations toolbar button. This metadata is applicable to all nodes.</td>
</tr>
<tr>
<td>node_minLength</td>
<td>Provides the value of minLength facet of the node's data type. Applicable to XML and text nodes with appropriate types.</td>
</tr>
<tr>
<td>node_maxLength</td>
<td>Provides the value of maxLength facet of the node's data type. Applicable to XML and text nodes with appropriate types.</td>
</tr>
<tr>
<td>node_totalDigits</td>
<td>Provides the value of the totalDigits facet of the node's data type. Applicable to XML nodes with appropriate types.</td>
</tr>
<tr>
<td>node_fractionDigits</td>
<td>Provides the value of the fractionDigits facet of the node's data type. Applicable to XML nodes with appropriate types.</td>
</tr>
<tr>
<td>node_length</td>
<td>Provides the length of the node's data type. Applicable to database fields with appropriate types.</td>
</tr>
<tr>
<td>node_precision</td>
<td>Provides the precision of the node's data type. Applicable to database fields with appropriate types.</td>
</tr>
<tr>
<td>node_scale</td>
<td>Provides the scale of the node's data type. Applicable to database fields with appropriate types.</td>
</tr>
</tbody>
</table>

To supply metadata to a node function:

1. Start creating a new node function (see Creating Defaults and Node Functions) or open an existing one for editing (see Editing and Deleting Existing Rules). For example, the function illustrated below concatenates the string "$" with the node value and returns the result back to the outer mapping. You can find the mapping of this function at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\OrderInUSD.mfd`. To open the function's mapping, click the icon next to the Rows item of the target component, and then click the button on the grid.
2. Do one of the following:

- Click **Add Node Specifics**.
- Right-click an empty area in the mapping, and select **Insert Input** from the context menu.
- Click the **Insert Input** toolbar button.
- On the **Function** menu, click **Insert Input**.
3. Select the required metadata from the dialog box (for example, "node_name").

**Note:** When you select a metadata parameter to insert, MapForce analyzes all currently expanded nodes on the mapping where the node function already qualifies to apply and determines whether the metadata parameter is supported by these nodes. If not supported, the dialog box displays a warning similar to "The selected metadata parameter is not supported by any currently existing node in the scope of this function". Note that, by default, deeply nested structures are not fully scanned, in order to preserve memory and improve the user experience. If the component where you apply the node function has such deeply nested structures, you can expand the relevant nodes on the mapping so as to make MapForce aware of them. In this case, MapForce will take the expanded nodes into account when you add a new metadata parameter, and the warning may disappear. Remember that a connection must exist for the node function to apply; expanding unconnected items is not relevant.

4. If the metadata is not supported for the node where the function qualifies to apply, you can decide the behavior of the function as follows:
   a. Select the check box **Return empty sequence from input** if you want to apply the node function and have the metadata parameter return an empty sequence. An empty sequence should not be confused with an empty string. You typically need to use sequence functions such as `substitute-missing` or `exists`, or other component
types to process it further. **Warning:** The empty sequence must be handled; otherwise, the node function might not return a value at all.

b. Select the check box **Do not apply the node function** if you do not want to apply the node function at all when this metadata is not supported by the node.

5. Click **OK**. A new input parameter is now added to the function's mapping, in addition to the default **raw_value** one. You can now connect the new parameter's output connector to some target item where you need this metadata (typically, a function's input connector).

### 8.2.6 Example: Replace Empty CSV Fields

This example shows you how to create a MapForce mapping that reads data from a CSV file and writes data to another CSV file. The goal is to replace all empty fields from the source CSV file with a custom value (“n/a”). In other words, assuming that the source CSV file looks as follows:

```
H,111,332.1,22537.7,,Container ship,,
D,111,A-1579-227,10,3,400,Microtome,,
D,111,B-152-427,7,6,1200,Miscellaneous,,
H,222,978.4,7563.1,,Air freight,,
D,222,ZZ-AW56-1,10,5,10000,Gas Chromatograph,,
```

then the desired mapping output should look as follows:

```
H,111,332.1,22537.7,n/a,Container ship,n/a,n/a,n/a
D,111,A-1579-227,10,3,400,Microtome,n/a,n/a
D,111,B-152-427,7,6,1200,Miscellaneous,n/a,n/a
H,222,978.4,7563.1,n/a,Air freight,n/a,n/a
D,222,ZZ-AW56-1,10,5,10000,Gas Chromatograph,n/a,n/a
```

You can find the mapping created in this example at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ReplaceEmptyFields.mfd`. The source CSV file for this mapping is called `Orders.csv` and is in the same folder. The target CSV file will be generated by MapForce.

To achieve the mapping goal, we will create a single node function that replaces each encountered empty value with "n/a". As shown below, this function is defined only once but it applies to multiple descendant CSV fields.

**Step 1: Add the source CSV file to the mapping**

You can add the source CSV file to the mapping as follows:

1. On the **Insert** menu, click **Text File**.
2. (MapForce Enterprise Edition only) Select the option **Use simple processing for standard CSV (delimited) and/or FLF (fixed-length) fields**, and click **Continue**.
3. Click **Input File** and browse for the following file: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\Orders.csv`.

If the check box **Treat empty fields as absent** is selected, clear it. When selected,
4. Click OK.
5. If prompted to change the component name to “Orders”, click the option you prefer (for example, Leave component name unchanged).

For more information about CSV components in MapForce, see CSV and Text Files.

Step 2: Add the target CSV file to the mapping
You can add the target CSV file to the mapping as follows:

1. On the Insert menu, click Text File.
2. (MapForce Enterprise Edition only) Select the option Use simple processing for standard CSV (delimited) and/or FLF (fixed-length) fields, and click Continue.
3. The target file must have the same number of fields as the source one. Therefore, click the Append Field button multiple times to add nine fields.
4. Click OK.

Step 3: Draw the mapping connections
At this stage, the mapping contains two components: the source CSV file and the target one. Click the output connector next to the Rows item on the source component and drag the cursor to the input connector of the Rows item in the target component. When you do this, MapForce may automatically connect all descendant items and create a so-called "Copy-All" connection, depending on your settings. This happens only if the Auto-connect matching children toolbar option is active. As mentioned previously, node functions are not applied to descendants of "Copy-All" connections. Therefore, the "Copy-All" connection must first be changed to a standard one. To do this, connect Field1 from source to Field1 from target. When prompted, click Replace Connection, and then click Resolve copy-all connection.

If the Auto-connect matching children option is not active, you can create connections between the source and target as follows:

1. Connect the Rows item in the source to the Rows item in the target.
2. Right-click the connection, and select Connect Matching Children from the context menu.
3. Clear the Create copy-all connections check box.
4. Click OK.

Your mapping should now look as follows:
Step 4: Create the node function

You can create a node function either immediately after data leaves the source, or immediately before it is written to the target. For the purpose of this example, let's create the node function on the input side of the target component; this essentially means “immediately before data is written to the target”. For more information, see Choosing When the Function or Default Should Apply.

Right-click the Rows item on the target component, and select Node Functions and Defaults | Input Node Functions and Defaults from the context menu. An empty grid appears at the top of the Mapping pane.

Next, click the Add function button to the left of the grid. The mapping now displays the function's input ("raw_value") and output ("result").
As mention before, the function's goal is to convert any empty value into the string "n/a". To achieve this, let's add the following additional components to the mapping:

- The MapForce built-in function `empty`. This function returns true if the value supplied as argument is empty; false otherwise. You can drag the function into the mapping from the Libraries window, or just double-click the mapping and type "empty", see also Add a Built-in Function to the Mapping.
- A text value "n/a". To add this value, double-click an empty area on the mapping and enter "n/a" surrounded by double quotes, see also Add a Constant to the Mapping.
- An If-Else Condition. To add it to the mapping, click the If-Else Condition ((toolbar button. For more information about such components, see Example: Returning a Value Conditionally.

With the help of these components, design the function as follows:

The design illustrated above works as follows: first, any input value from the outer mapping enters the function through the `raw_value` input. The raw value is then supplied as input to the `empty` function. Then, the If-Else Component evaluates the Boolean result (true or false) returned by the `empty` function. When the result is true, the constant "n/a" becomes the function's result. When the result is false, the function's raw input value becomes the function's result. The function's `result` (which is either "n/a" or `raw_value`) is then returned to the outer mapping.

Click Exit (or press Escape) to exit the function's editing area.
In the mapping illustrated above, note the following:

- The text at the top of the window clearly indicates where the function is defined. This is particularly useful in situations where multiple node functions are defined for the same component.
- The Apply to option in the grid is set to All descendant items. In this example, this is the intended behavior. That is, all descendant items of Rows must be affected if they qualify. As you can see on the mapping, the left (input) side of the target component displays multiple icons, even though the function was defined only once, for the parent item.
- The Data Type option is set to "string". In this example, since we are dealing with text data, this is the intended behavior. It is also the default behavior.
- The Edit button lets you go back to the function's definition and change it if necessary. If you don't see this button, click the icon first.

Step 5: Run the mapping
To preview the mapping result directly in MapForce, click the Output tab. If any validation errors are encountered, these are displayed in the Messages window, see Validating Mappings. Upon success, the resulting CSV is displayed in the Output pane.

You can also execute such mappings on a server machine, with MapForce Server Advanced Edition, in one of the following ways:

- If you have MapForce Server Advanced Edition standalone license, compile the mapping
to an execution file and then copy it to the target machine, see Compiling Mappings to MapForce Server Execution Files.

- If you have licensed both FlowForce Server and MapForce Server Advanced Edition, you can deploy the mapping directly to FlowForce Server and configure it to run as a scheduled or on-demand job, see Deploying Mappings to FlowForce Server.
8.3 User-Defined Functions

User-Defined Functions (UDFs) are custom functions defined once, and reusable multiple times within the same mapping or across multiple mappings. User-defined functions are like mini-mappings themselves: they typically consist of one or more input parameters, some intermediary components to process data, and an output to return data to the caller. The caller is either the main mapping or another user-defined function.

Tip: It is also possible to create user-defined functions that return multiple outputs. This is supported when the function is defined as “inline”, see Inline and Regular User-Defined Functions.

Apart from being reusable, user-defined functions are also helpful when you want to package parts of the mapping into smaller components and abstract away the implementation details, thus making the main mapping easier to read.

You typically create user-defined functions to process strings, numbers, dates, and other data in a custom way that extends the built-in MapForce functions. For example, you might want to concatenate or split text in a particular way, or perform some advanced calculations, or manipulate dates and times, or simply hide parts of a mapping by packaging them into a reusable function. Another common use of user-defined functions is to look up a field in a file, database or some other data store supported by MapForce.

The following is an example of a user-defined function that splits a string into two separate strings. This user-defined function is part of the following demo mapping: `<Documents>\Altova\MapForce2019\MapForceExamples\ContactsFromPO.mfd`. It takes a name as parameter (for example, “Helen Smith”), applies the built-in functions `substring-before` and `substring-after`, and then returns two resulting values (“Helen” and “Smith”).

![Diagram of User-Defined Functions](image)
As stated before, you can call a user-defined function either from the main mapping, or from another user-defined function. In other words, user-defined functions can be nested if so required, as illustrated below.

For example, the mapping below calls a user-defined function ("LookupPerson") to look up a person's name in an XML file. If you double-click the header of the "LookupPerson" component, its definition opens in the mapping window, and you will notice that this function calls other user-defined functions in its turn: "EqualAnd" and "Person2Details". This mapping is available as a demo at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\PersonListByBranchOffice.mfd`. 

```
 function LookupPerson(xml: string, substr: string) -> string
    call EqualAnd
    call Person2Details
    return result

function EqualAnd(str1: string, str2: string) -> bool
    if str1 equals str2 then return true
    else return false

function Person2Details(name: string) -> string
    if name is not empty then return Last
    else return First
```
User-defined functions can also be called recursively (that is, a user-defined function calls itself). This requires, however, that the user-defined function be defined as a regular (not inline) function, see Inline and Regular User-Defined Functions.

Recursive user-defined functions let you solve various advanced mapping requirements, such as iterating over data structures having a depth of $N$ children, where $N$ is not known in advance, see Example: Recursive Search.

After you create a user-defined function, it is saved in the same mapping where you created it. However, you can import it into other mappings as well and call it from there. For more information, see Calling and Importing User-Defined Functions.

### 8.3.1 Creating User-Defined Functions

You can create user-defined functions either from scratch, or from a selection of components that already exist on the mapping.
To create a user-defined function from scratch:

1. On the Function menu, click Create User-Defined Function. Alternatively, click the Create User-Defined Function toolbar button.

2. Enter information into the required fields (see the reference table below).

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Mandatory field. Enter a name for the user-defined function you wish to create. Valid characters:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Alphanumeric characters (a-z, A-Z, 0-9)</td>
</tr>
<tr>
<td></td>
<td>• Underscore (_ _)</td>
</tr>
<tr>
<td></td>
<td>• Hyphen/dash (-)</td>
</tr>
<tr>
<td></td>
<td>• Colon (:)</td>
</tr>
</tbody>
</table>

| Library Name  | Mandatory field. Enter a library name where the function should belong. The function will be displayed under this library name in the Libraries window. If you don't specify a library, the function will |
be placed into a default library called “user”.

**Syntax**  
Optional field. Enter some text that concisely describes the syntax of the function (for example, the expected parameters). This text will be displayed next to the function in the Libraries window, and it does not affect the implementation of the function.

**Detail**  
Optional field. Enter the free text description of the function. This text will be displayed when you move the cursor over the function in the Libraries window or in other contexts.

**Inlined use**  
Select this check box if the function should be created as inline. Clear the check box to create a regular function. For more information, see Inline and Regular User-Defined Functions.

3. Click **OK**. The function becomes immediately visible in the Libraries window under the library name specified above, for example:

![Libraries window](image)

Also, the mapping window is now redrawn so as to allow you to create the new function (this is a standalone mapping referred to as the "function's mapping"). Since any function requires an output, the function's mapping includes an output component by default.

![Function's mapping](image)

The **Return to main mapping** button in the top-left corner lets you navigate from the function's mapping back to the main mapping. To open the function's mapping at any time, double-click the function in the Libraries window. For more information, see Calling and Importing User-Defined Functions and Navigating User-
4. Add to the function’s mapping all the components required by the function’s definition. You can do this in the same way as for a standard mapping. For example, to add input or output parameters, do one of the following:

- Run the menu command **Function | Insert Input**, or **Function | Insert Output**, respectively.
- Right-click the mapping area, and select **Insert Input** or **Insert Output** from the context menu.
- Click the **Insert Input** or **Insert Output** toolbar buttons.

At the minimum, a function requires one output component to which some data is connected. As for input parameters, a function can have zero, one or more inputs. The input or output parameters can be of simple type (such as string or integer) or complex type (a structure). For more information about simple and complex parameters, see Parameters in User-Defined Functions.

So far, you created the user-defined function, but you haven't used it anywhere yet. To use the function in a mapping, drag the function from the Libraries window onto the main mapping area, see also Calling and Importing User-Defined Functions.

To create a user-defined function from existing components:

1. Select multiple components on the mapping by making a rectangular selection with the mouse. You can also select multiple components by clicking each one while holding the Ctrl key pressed.
2. On the **Function** menu, click **Create User-Defined Function from Selection**. Alternatively, click the **Create User-Defined Function from selection** toolbar button.
3. Follow the steps 2-4 above.

8.3.2 Parameters in User-Defined Functions

When you create a user-defined function, you must specify what input parameters it should take (if any) and what output it should return. While input parameters are sometimes not necessary, an output parameter is mandatory in all cases (that is, a function must always return something). For example, the function below has no inputs and one output which returns the text “hello” to the caller:

![User-defined function example]

Function parameters can be of simple type (such as string or integer) or a complex structure. For
example, the user-defined function “FindArticle” illustrated below has two input and one output parameters.

- **POArtNr** is an input parameter of simple type “string”
- **Amount** is an input parameter of simple type “integer”
- **CompletePO** is an output parameter of complex XML type.

```
 LookupArticle.mfd
```

This mapping above is available as a demo at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\LookupArticle.mfd`.

Adding Parameters

To add an input or output parameter:

1. Create a user-defined function mapping (see Creating User-Defined Functions) or open an existing one (see Editing User-Defined Functions).
2. Do one of the following:

   - Run the menu command **Function | Insert Input** or **Function | Insert Output**.
   - Click the **Insert Input** or **Insert Output** toolbar buttons.
3. In the dialog box above, choose whether input or output parameters should be of simple type (such as string or integer) or a complex structure (such as an XML structure). To create a parameter that is a complex XML type, click **Choose** next to "Structure" and browse for the XML schema that describes the required structure.

If the function's mapping already includes XML schemas, they are available for selection as structures. Otherwise, you can select a completely new schema that should provide the structure of the parameter. The same is true for databases or other complex structures if they are supported by your edition of MapForce.

With XML structures, it is possible to select a root element for your structure, if the XML schema allows it. To specify a root element, click **Choose** next to "Root", and select the root element from the dialog box that opens.

If selected, the check box **Save structure file path relative to MFD file** will change the structure file's absolute path into a path relative to the current mapping, when you save the mapping. For more information, see Using Relative Paths on a Component.

The **Input is required** and **Input is Sequence** check boxes are explained in the following sections.

**Mandatory parameters**

To make a parameter mandatory in a user-defined function, select the **Input is required** check box. When a parameter is mandatory, validation errors will occur if you do not connect an input to it.

To make a parameter optional, clear the **Input is required** check box. On the main mapping, optional parameters have a slightly different appearance—their input connector (small triangle) has
You can also specify a default parameter value by connecting it to the "default" input of a parameter, for example:

The default value will apply only if there is no other value. If the optional parameter receives a value when the function is called, then that value takes precedence over the default.

**Sequence parameters**

You can optionally specify that a function's parameter should be treated as a single value (this is the default behaviour), or as a sequence. To treat the parameter as a sequence as opposed to a single value, select the **Input is sequence** check box. Note that this check box is meaningful and enabled only if the user-defined function is of type "regular", see Inline and Regular User-Defined Functions. Otherwise, the check box is disabled.

A sequence is a range of zero or more values. You might want to treat a parameter as a sequence when your user-defined function expects input data as a sequence, in order to perform some aggregation of values in that sequence (for example, by calling functions such as avg, min, max). For an example, open the following demo mapping: <Documents>\Altova\MapForce2019\MapForceExamples\InputsIsSequence.mfd. In this mapping, the "data" filter is connected to the user-defined function "Calculate". The filter's output is a sequence of items, so the input parameter of the function is set to sequence.

Internally, the "Calculate" function aggregates all the sequence values (as illustrated below, it runs the min, max, and avg aggregate functions on the input sequence).
As a rule of thumb, the input data, either sequence or non-sequence, determines how often the function is called.

- When input data is connected to a sequence parameter, the user-defined function is called only once and the complete sequence is passed into the user-defined function.
- When input data is connected to a non-sequence parameter, the user-defined function is called once for each single item in the sequence.

Connecting an empty sequence to a non-sequence parameter has the result that the function is not called at all.

This can happen if the source structure has optional items, or when a filter condition returns no matching items. To avoid this, either use the substitute-missing function before the function input to ensure that the sequence is never empty, or set the parameter to sequence, and add handling for the empty sequence inside the function.

The Output is sequence check box may be required for output parameters also. When a function passes a sequence of multiple values to its output component, and the output component is not set to sequence, the function will return only the first item in the sequence.

Parameter order

When a user-defined function has multiple input or output parameters, you can change the order in which parameters should appear to callers of this function. For example, the function below has three input parameters, input1, input2, and input3.
The order of parameters in the function's mapping (starting from the top) dictates the order in which they appear to callers of this function:

Note the following:

- Input and output parameters are sorted by their position from top to bottom. Therefore, if you move parameter `input3` to the top in the function's mapping, it will become the first parameter of this function.
- If two parameters have the same vertical position, the leftmost takes precedence.
- In the unusual case that two parameters have exactly the same position, the internal component ID is automatically used.

### 8.3.3 Inline and Regular User-Defined Functions

There are two kinds of user-defined functions: inline and regular. You can specify whether a function should be inline or regular when creating the function, see Creating User-Defined Functions. Inline and regular functions behave differently in terms of code generation, recursiveness, and the ability to have multiple output parameters.

<table>
<thead>
<tr>
<th>Inline functions</th>
<th>Regular functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inline functions are extracted in all instances where they occur in generated code, which makes the code longer but also slightly faster. Note that inline functions can significantly increase the amount of generated program code. The user-defined function code is</td>
<td>Each user-defined function component generates code for a function call, where inputs are passed as parameters, and the output is the function (component) return value. At runtime, all the input parameter values are</td>
</tr>
</tbody>
</table>
### Inline functions vs. Regular functions

<table>
<thead>
<tr>
<th>Inline functions</th>
<th>Regular functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actually inserted at all locations where the function is called, and thus increases the code size substantially - as opposed to using a regular function.</td>
<td>evaluated first, and then the function is called for each occurrence of the input data.</td>
</tr>
<tr>
<td>Inline functions can have multiple outputs and thus return multiple values.</td>
<td>Regular functions can have only one output. To return multiple values, you can declare the output to be of complex type (for example, XML structure), which would allow you to pass multiple values to the caller.</td>
</tr>
<tr>
<td>Inline functions cannot be called recursively.</td>
<td>Regular functions can be called recursively.</td>
</tr>
<tr>
<td>Inline functions do not support setting a priority context on a parameter, see Priority Context node/item.</td>
<td>Regular functions support setting a priority context on a parameter.</td>
</tr>
</tbody>
</table>

On the mapping, inline user-defined functions are displayed with a dashed border. For example, the middle component in the mapping below is an inline user-defined function.

Regular functions are displayed with a solid border. For example, the middle component in the mapping below is a regular user-defined function.
8.3.4 Navigating User-Defined Functions

When a mapping contains user-defined functions, you can easily navigate between the definition (mapping) of each user-defined function and the main mapping as shown below.

To open a user-defined function for viewing or editing:

- Double-click the title bar of a user-defined function on the mapping.
- Double-click the specific user-defined function in the Libraries window.

You can also edit a function by double-clicking its name in the Libraries window. However, only functions in the currently active document can be opened this way. Double-clicking a user-defined function that was created in another mapping opens that mapping in a new window.

Note: If you edit or delete a user-defined function that was imported into multiple mappings, all importing mappings will be affected by the change.

To go back to the main mapping:

- Click the Return to main mapping button in the top-left corner of the mapping window.

In addition, a history is preserved as you navigate through various MapForce tabs, including user-defined functions. To go back and forward between tabs visited previously, click the Back and Forward toolbar buttons. The corresponding keyboard shortcuts for these buttons are Alt+Left and Alt+Right, respectively.

8.3.5 Editing User-Defined Functions

To edit a user-defined function:

1. Open the mapping that contains the user-defined function.
2. Double-click the title bar of the user-defined function on the mapping. The Mapping window changes to display the function's contents where you can add, edit, or remove components as required.
3. To change the function's properties (such as name or description), do one of the following:
   a. Right-click an empty area on the mapping and select Function Settings from the context menu.
   b. Click the User-defined function settings toolbar button.
You can also edit a function by double-clicking its name in the Libraries window. However, only functions in the currently active document can be opened this way. Double-clicking a user-defined function that was created in another mapping opens that mapping in a new window.

**Note:** If you edit or delete a user-defined function that was imported into multiple mappings, all importing mappings will be affected by the change.

### 8.3.6 Deleting User-Defined Functions

**To delete a user-defined function:**

1. Double-click the title bar of the user-defined function on the mapping.
2. Click the **Erase** button in the top-right corner of the Mapping window.
3. If the function is used in the currently open mapping, a dialog box is displayed.

   ![MapForce](image)

   **MapForce**

   ![Instances of this user-defined function, split-name (in library: user), exist in the following:
   - Main mapping
   
   Would you like to replace all instances with internal components?](image)

   Click **Yes** if you want to delete function and replace all instances where it is called with the function's components. This lets you keep the main mapping valid even if the function is deleted. However, if the deleted function is used in any other external mappings, those will become not valid.

   Click **No** if you want to delete the function and all its internal components permanently (in this case, all the mappings where the function is used will become not valid).
8.3.7 Calling and Importing User-Defined Functions

After you create a user-defined function, you can call it either from the same mapping where you created it, or from any other MapForce mapping. For information about creating a function, see Creating User-Defined Functions.

To call a user-defined function from the same mapping:

1. Find the function in the Libraries window. You can find the function under the library that you specified when you created the function. If you created the function in the default "user" library, look for the function in the "user" library. To quickly find the function by its name, start typing the name in the Libraries window.

2. Drag the function from the Libraries window into the main mapping. You can now connect to it all the required parameters. The result of the function is provided by its output parameter (or several parameters, if applicable).

To import a user-defined function from another mapping:

1. Click the Add/Remove Libraries button at the base of the Libraries window.
2. In the Options dialog box, in the Libraries page, click Add, and browse for the mapping file (.mfd) that contains the user-defined function.
3. Click OK. The function now appears in the Libraries window from where you can drag it into the main mapping.

By default, the Libraries window displays only built-in functions and any user-defined functions in the current .mfd mapping file. If you import other .mfd files as libraries into the current mapping as shown above, it will display user-defined functions from any imported files as well. Moreover, if you
used the same library name across multiple *.mfd files or custom libraries (see Importing Custom Java and .NET Libraries), functions from all available sources appear under the same library name in the Libraries window.

8.3.8 Example: Look-up and Concatenation

There are several demo mappings available with MapForce that illustrate typical usage of user-defined functions. One of these mappings is the PersonListByBranchOffice.mfd file available in the \<Documents>\Altova\MapForce2019\MapForceExamples\ folder.

![PersonListByBranchOffice.mfd](image)

This mapping has the following business requirements:

- Extract data from a source XML file and write it to a target XML file. Data consists of employee details, such as first name and last name.
- Look up certain data about each employee in a separate XML file (phone, email address, position).
- Process data in a desired way before writing it to the target. Namely, the phone, email and position of each person must be represented as a single string (comma-separated) and written to the Details element of the target XML.
- Extract only XML elements that match certain criteria—in this case, information about employees from a specific branch office. Callers of the mapping must be able to specify the office name as a parameter at the command line, for example, when the mapping is executed by MapForce Server.

Let's now examine the components that implement the requirements above:

- The input parameter of the mapping ("OfficeName") is a simple input component. A default value ("Nanonull, Inc.") is provided by a constant—this value will be used if the caller of the mapping does not provide a parameter value. To find more about simple input components, see Supplying Parameters to the Mapping.
- To filter only employees that belong to a specific office, the mapping uses a filter
component ("Office"). Essentially, the filter checks whether the office name supplied by
the parameter is equal to the office name in the source XML file. If yes, the filter passes
data from the source Office item to the target component. For more information about
filters, see Filters and Conditions.

- To look up information from the second source XML file, the mapping calls a user-defined
  function, "LookupPerson". The logic of this function is discussed in more detail below.
- To process employee data, the "LookupPerson" function calls internally other functions
  that retrieve and concatenate information about each employee in a suitable way. All
  these operations are in the function's own mapping and not visible in the main mapping—a
  typical example of encapsulation. The "LookupPerson" function then populates the
  Details element in the target XML.

Look-up implementation
The look-up functionality is provided by the "LookupPerson" function, whose definition is illustrated
below.

As shown above, the function includes the source XML file from where data should be retrieved.
Next, it has three input parameters that provide the look-up values: Office_Name, First_Name,
and Last_Name. All input parameters are set as mandatory (that is, the check box Input is
required is selected in the Properties dialog box).

The "EqualAnd function" is a separate user-defined function enclosed into the current one. This
function returns a Boolean value. Calling this function in the sequence illustrated above provides
the following Boolean logic:
The function's value (TRUE or FALSE) is passed to the filter each time a new item is processed. When the filter gets value TRUE, the look-up operation is successful and the employee's details are retrieved and returned to the outer mapping. Otherwise, the next item in context is examined, and so on until the loop finishes.

In the first occurrence of "EqualAnd" function, connector b has a circle around it—this indicates that this parameter is set as priority context. Priority context is an optional feature that optimizes the execution of the mapping. Namely, it ensures that the person data of the specific office supplied by the input parameter a is processed first. To set a parameter as priority context, right-click it and choose Priority from the context menu. For more information, see Priority Context node/item.
Concatenation implementation

The "Person2Details" function is another function nested into "LookupPerson" function. This function returns a string value. It concatenates the three values received as parameters and two text constants, as illustrated below:

```
concat
```

The `concat` function is a MapForce built-in function that can take as many parameters as required, see Add or Delete Function Arguments.

Running the mapping

To preview the mapping execution in MapForce, click the Output tab. The mapping runs with the default input parameter ("Nanonull, Inc.") and consequently retrieves employee data only for this office. To retrieve data for another office, change the constant connected to the input parameter from "Nanonull, Inc." to "Nanonull Partners, Inc." and run the mapping again.

If you have licensed MapForce Server, you can also run the mapping at the command line on a Linux, macOS, or Windows machine. First, compile the mapping to a MapForce Server execution file (.mfx) with the menu command File | Compile to MapForce Server Execution File, see also Compiling Mappings to MapForce Server Execution Files. Next, copy the .mfx file to the server machine and run MapForce Server with the command below. The named parameter `-p=OfficeName` supplies the input value:

```
mapforceserver run PersonListByBranchOffice.mfx -p=OfficeName:"Nanonull, Inc."
```

Notes:

- `mapforceserver` is the path to the MapForce Server executable as applicable for your operating system.
- Change the path to the .mfx file as applicable, or copy the .mfx to the same folder as the executable.

8.3.9 Example: Recursive Search

This example illustrates a mapping that searches for data in a source XML file with the help of a recursive user-defined function. The mapping file is available at the following path: <Documents>\Altova\MapForce2019\MapForceExamples\RecursiveDirectoryFilter.mfd.
The source XML file contains information about files and directories, as illustrated by the code listing below (note that the listing omits some data for simplicity):

```xml
<?xml version="1.0" encoding="UTF-8"?>
<directory name="Examples">
  <directory name="ExampleSite">
    <file name="blocks.sps" size="7473"/>
    <file name="block_file.xml" size="992"/>
    <directory name="output">
      <file name="examplesite1.css" size="3174"/>
      <directory name="images">
        <file name="blank.gif" size="88"/>
        <file name="block_file.gif" size="13179"/>
      </directory>
    </directory>
  </directory>
</directory>
```

Source XML file

Both the source and the target XML files use the same schema, `Directory.xsd`. Since, on a file system, a directory can contain either a file or another directory, this is also reflected in the schema. Importantly, the schema specifies that the `directory` element is recursive (see the line `<xs:element ref="directory"/>`).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="directory">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="file">
          <xs:complexType>
            <xs:attribute name="name" type="xs:string"/>
            <xs:attribute name="size" type="xs:unsignedLong"/>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
The business requirement of the mapping is to filter out only files with a specific extension. The nested structure of all directories must be preserved. For example, if extension is ".xml", the expected output (for the source XML file listed previously) should look as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<directory name="Examples">
  <directory name="ExampleSite">
    <file name="block_file.xml" size="992"/>
    <directory name="output">
      <directory name="images"/>
    </directory>
  </directory>
</directory>
```

Expected XML output

Secondly, callers of the mapping must be able to supply the file extension as a parameter. By default, if a caller does not supply a parameter value, the mapping will filter out files with .xml extension.

To address the requirements above, the mapping contains a simple input parameter, "SearchFor", which supplies the default file extension by means of a text constant. This parameter is optional (the Input is required check box is not selected in the Properties dialog box):
For more information about input parameters, see [Supplying Parameters to the Mapping](#).

Next, the mapping includes a user-defined function, "FilterDirectory". This function is recursive, that is, it includes a call to itself. Because it is connected to the recursive element directory, this function will be called as many times as there are nested directory elements in the source XML instance. To support recursive calls, this function was created as regular, not inline (the Inlined use option is not selected in the function's properties). To view the function's properties, right-click an empty area in the mapping and select Function Settings from the context menu, see also [Editing User-Defined Functions](#).

As illustrated above, the function takes two parameters as input:

1. A complex parameter, Directory, which defines the XML structure to be searched (this parameter is the "haystack").
2. A string parameter, SearchFor, which specifies the file extension to search for (this parameter is the "needle").

Double-click the title bar of any of the input or output parameters on the mapping to view their settings.

The function also includes a filter component to which the MapForce built-in function contains is connected. The contains function returns true only when the search value matches the "name" attribute (the file name) in the source structure. A true value instructs the filter to copy the current item to the output; otherwise, it is skipped. For more information about filters, see [Filters and Conditions](#).

The source and target files of the mapping, as well as the function's directory parameter (both input and output), have all the same schema, Directory.xsd. Since MapForce detected all these types to be assignment compatible, the connection type between the input parameters and the function is "Copy-All", see [Copy-All Connections](#).

Running the mapping

To preview the mapping execution in MapForce, click the Output tab. The mapping runs with the default input parameter (".xml") and consequently retrieves only results that match this search criterion. To supply a different search criterion, change the constant connected to the input parameter from ".xml" to ".sps", for example, and run the mapping again.
If you have licensed MapForce Server, you can also run the mapping at the command line on a Linux, macOS, or Windows machine. First, compile the mapping to a MapForce Server execution file (.mfx) with the menu command **File | Compile to MapForce Server Execution File**, see also [Compiling Mappings to MapForce Server Execution Files](#). Next, copy the .mfx file to the server machine and run MapForce Server with the command below. The named parameter `-p=SearchFor` supplies the input value:

```bash
mapforceserver run RecursiveDirectoryFilter.mfx -p=SearchFor:".sps"
```

**Notes:**

- `mapforceserver` is the path to the MapForce Server executable as applicable for your operating system.
- Change the path to the .mfx file as applicable, or copy the .mfx to the same folder as the executable.
8.4 Importing Custom XSLT 1.0 or 2.0 Functions

You can extend the XSLT 1.0 and 2.0 function libraries available in MapForce with your own custom functions, provided that your custom functions return simple types.

Only custom functions that return simple data types (for example, strings) are supported.

To import functions from an XSLT file:

1. On the Tools menu, click Options. (Alternatively, click Add/Remove Libraries in the lower area of the Libraries window.)
2. Next to Libraries, click Add and browse for the .xsl or .xslt file.

Imported XSLT files appear as libraries in the Libraries window, and display all named templates as functions below the library name. If you do not see the imported library, ensure you selected XSLT as transformation language (see Selecting a Transformation Language).

Note the following:

- To be eligible for import into MapForce, functions must be declared as named templates conforming to the XSLT specification in the XSLT file. You can also import functions that occur in an XSLT 2.0 document in the form `<xsl:function name="MyFunction">`. If the imported XSLT file imports or includes other XSLT files, then these XSLT files and functions will be imported as well.
- The mappable input connectors of imported custom functions depends on the number of parameters used in the template call; optional parameters are also supported.
- Namespaces are supported.
- If you make updates to XSLT files that you have already imported into MapForce, changes are detected automatically and MapForce prompts you to reload the files.
- When writing named templates, make sure that the XPath statements used in the template are bound to the correct namespace(s). To see the namespace bindings of the mapping, preview the generated XSLT code.

Datatypes in XPath 2.0

If your XML document references an XML Schema and is valid according to it, you must explicitly construct or cast datatypes that are not implicitly converted to the required datatype by an operation.

In the XPath 2.0 Data Model used by the Altova XSLT 2.0 Engine, all atomized node values from the XML document are assigned the `xs:untypedAtomic` datatype. The `xs:untypedAtomic` type works well with implicit type conversions.

For example,

- the expression `xs:untypedAtomic("1") + 1` results in a value of 2 because the `xdt:untypedAtomic` value is implicitly promoted to `xs:double` by the addition operator.
- Arithmetic operators implicitly promote operands to `xs:double`.
- Value comparison operators promote operands to `xs:string` before comparing.
8.4.1 Example: Adding Custom XSLT Functions

This example illustrates how to import custom XSLT 1.0 functions into MapForce. The files needed for this example are available in the `<Documents>\Altova\MapForce2019\MapForceExamples` directory.

- **Name-splitter.xslt**: This XSLT file defines a named template called "tokenize" with a single parameter "string". The template works through an input string and separates capitalized characters with a space for each occurrence.

```
2  <xs:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3    xmlns="http://www.w3.org/1999/XSL/Transform"
4    xmlns:xs="http://www.w3.org/2001/XMLSchema"
5    xmlns:xml="http://www.w3.org/1999/XSL/Transform"
6    xmlns:xslt="http://www.w3.org/1999/XSL/Transform"
7  <xsl:template match="*">
8    <xsl:for-each select=".">
9      <xsl:call-template name="tokenize">
10     <xsl:with-param name="string" select="."/>
11    </xsl:call-template>
12  </xsl:for-each>
13  </xsl:template>
14  <xsl:template name="tokenize">
15    <xsl:param name="string" select="."/>
16    <xsl:variable name="caps" select="translate($string,-abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ,ABCDEFGHIJKLMNOPQRSTUVWXYZ)"/>
17    <xsl:variable name="token" select="string-length($caps)"/>
```

- **Name-splitter.xml** (the source XML instance file to be processed)
- **Customers.xsd** (the source XML schema)
- **CompletePO.xsd** (the target XML schema)

To add a custom XSLT function:

1. Select XSLT as transformation language (see Selecting a Transformation Language).
2. Click the Add/Remove Libraries button, in the lower area of the Libraries window. Alternatively, on the Tools menu, click Options, and then select Libraries.
3. Click Add, and browse for the XSL, or XSLT file, that contains the named template you want to act as a function, in this case Name-splitter.xslt.
4. Click **OK**. The XSLT file name appears in the Libraries window, along with the functions defined as named templates (in this example, **Name-splitter** with the **tokenize** function).

![Libraries window](image)

**To use the XSLT function in your mapping:**

1. Drag the **tokenize** function into the Mapping window and map the items as show below.

![Mapping window](image)

2. Click the XSLT tab to see the generated XSLT code.
Note: As soon as a named template is used in a mapping, the XSLT file containing the named template is included in the generated XSLT code (xsl:include href...), and is called using the command xsl:call-template.

3. Click the Output tab to see the result of the mapping.

To remove custom XSLT libraries from MapForce:

1. Click the Add/Remove Libraries button, in the lower area of the Libraries window.
2. Click the XSLT library to be deleted, and then click Delete.
8.4.2 Example: Summing Node Values

This example shows you how to process multiple nodes of an XML document and have the result mapped as a single value to a target XML document. Specifically, the goal of the mapping is to calculate the price of all products in a source XML file and write it as a single value to an output XML file. The files used in this example are available in the <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\ folder:

- Summing-nodes.mfd — the mapping file
- input.xml — the source XML file
- input.xsd — the source XML schema
- output.xsd — the target XML schema
- Summing-nodes.xslt — A custom XSLT stylesheet containing a named template to sum the individual nodes.

There are two different ways to achieve the goal of the mapping:

- By using the sum aggregate function of the core library. This function is available in the Libraries window (see also Working with Functions).
- By importing a custom XSLT stylesheet into MapForce.

Solution 1: Using the "sum" aggregate function

To use the sum aggregate function in the mapping, drag it from the Libraries window into the mapping. Note that the functions available in the Libraries window depend on the XSLT language version you selected (XSLT 1 or XSLT 2). Next, create the mapping connections as shown below.

For more information about aggregate functions of the core library, see also core | aggregate functions.

Solution 2: Using a custom XSLT Stylesheet

As mentioned above, the aim of the example is to sum the Price fields of products in the source XML file, in this case products A and B.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Input xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="input.xsd"/>
```
The image below shows a custom XSLT stylesheet which uses the named template "Total" and a single parameter string. The template works through the XML input file and sums all the values obtained by the XPath expression /Product/Price.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  
  <xsl:template match="*">
    <xsl:for-each select="."/>
    <xsl:call-template name="Total">
      <xsl:with-param name="string" select="."/>
    </xsl:call-template>
  </xsl:for-each>
</xsl:template>

  <xsl:template name="Total">
    <xsl:param name="string"/>
    <xsl:value-of select="sum($string/Product/Price)"/>
  </xsl:template>
</xsl:stylesheet>
```

**Note:** To sum the nodes in XSLT 2.0, change the stylesheet declaration to version="2.0".

To import the XSLT stylesheet into MapForce:

1. Select XSLT as transformation language. For more information, see Selecting a Transformation Language.
2. In the Libraries window, click Add/Remove Libraries.
3. On the Options dialog box, click the Libraries tab.
4. Click Add and browse for <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\Summing-nodes.xslt.
5. Drag the Total function from the newly created "Summing-nodes" library into the mapping, and create the mapping connections as shown below.
To preview the mapping result, click the **Output** tab. The sum of the two **Price** fields is now displayed in the **Total** field.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Output xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="output.xsd">
    <Total>25</Total>
    <Product>
        <Name>ProductA</Name>
        <Amount>10</Amount>
        <Price>5</Price>
    </Product>
    <Product>
        <Name>ProductB</Name>
        <Amount>5</Amount>
        <Price>20</Price>
    </Product>
</Output>
```
8.5 Importing Custom XQuery 1.0 Functions

When XQuery is selected as mapping transformation language, MapForce displays the built-in function libraries available for XQuery in the Libraries window. If necessary, you can extend this list with custom XQuery functions, by importing custom XQuery 1.0 library modules into MapForce.

To be eligible for import into MapForce, an XQuery file must satisfy the following requirements:

- It must be a valid library module according to XQuery specification. In other words, it must start with a module declaration such as `module namespace <prefix>="<namespace name>"
- All functions declared in the imported library module must return atomic data types (for example, `xs:string`, `xs:boolean`, `xs:integer`, etc). Function parameters must also have atomic types.

To import an XQuery library module:

1. On the Tools menu, click Options. (Alternatively, click Add/Remove Libraries in the lower area of the Libraries window.)
2. Next to Libraries, click Add and browse for the .xq or .xquery library file.

If the imported library module is not supported, a message box prompts you. Otherwise, the imported library modules appear in the Libraries window, and then you can drag specific functions into the mapping area and use them like any other MapForce function component.

If you do not see the imported XQuery library module, make sure that XQuery is selected as transformation language (see Selecting a Transformation Language).

See also:
- XQuery engine implementation

8.5.1 Example: Import Custom XQuery Function

This example shows you how to import a demo XQuery library module into MapForce and call its functions from a mapping. The demo module in this example consists of only one function which calculates tax on decimal amounts as 20% of the amount. In a production scenario, an XQuery module may contain multiple functions.

All functions declared in the XQuery module must return atomic types and their parameters must also be of atomic data types. Otherwise, the module is not eligible for import into MapForce.

You can find the demo XQuery module file at the following path relative to your personal “Documents” folder on the computer where MapForce is installed: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\module.xq`. 
xquery version "1.0";

module namespace demo="http://www.altova.com/mapforce/demo";

declare function demo:calculatetax($val as xs:decimal) as xs:decimal {
    $val*0.2
};

module.xq

After you import the XQuery module file into MapForce, you will be able to call the `demo:calculatetax` function from a mapping. Note that calculating the tax amount with the help of an XQuery function is just for demo purposes—you can achieve the same result by using MapForce built-in functions.

A demo mapping which calls the `demo:calculatetax` function above is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\CalculateTax_XQuery.mfd`. When you open this mapping initially, MapForce displays a warning that it contains one or more components that are not available in XQuery. This warning is normal and it occurs because the mapping references a function from a custom XQuery library module that was not imported yet. To remove the warning and run the mapping, we will import the missing XQuery module into MapForce as shown below.

To import the XQuery module into MapForce:

1. Scroll down to the end of the Libraries window and click Add/Remove Libraries. The Options dialog box opens.
2. Click Add and browse for the following file: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\module.xq`.
3. Click OK to close the dialog box. A message box appears informing you that a new library has been added.

The imported library and the `demo:calculatetax` function are now visible in the Libraries window.
Also, the mapping can now be validated and run without warnings. The `demo:calcutetax` function illustrated in the image below originates from the imported XQuery module, and it can be added to the mapping just like any other built-in function, see [Add a Built-in Function to the Mapping](#).

The mapping explained

The `CalculateTax_XQuery.mfd` mapping illustrated above takes as input an XML file that stores articles. Each article has a single price expressed as a decimal value, for example:

```xml
<Articles>
  <Article>
    <Number>1</Number>
    <Name>T-Shirt</Name>
    <SinglePrice>25</SinglePrice>
  </Article>
  <Article>
    <Number>2</Number>
    <Name>Socks</Name>
    <SinglePrice>2.30</SinglePrice>
  </Article>
</Articles>
```
The mapping produces an XML file which abides by the same schema as the source XML file. Therefore, both the source and target components have the same structure on the mapping. As the mapping connections suggest, nearly all elements are mapped in a straightforward way from target to source—for example, for each Article in the source there will be an Article in the target. The values of all items are copied verbatim from the source XML, except for SinglePrice. The value of SinglePrice is calculated with the help of two functions:

- The demo:calculatetax XQuery function calculates the tax amount by taking the original SinglePrice as input.
- The MapForce built-in add function add the tax amount to the original SinglePrice amount and returns the final amount.

Importantly, the data type of the SinglePrice item is xs:decimal—which corresponds to the input parameter type and the return type of the XQuery function.

The output produced by the mapping when you click the Output tab is illustrated below. Notice the increase of 20% applied to each price compared to the source XML.
8.6 Importing Custom Java and .NET Libraries

Compiled Java class files as well as .NET DLL assemblies (including .NET 4.0 assemblies) can be imported into MapForce. If the imported libraries contain functions that use basic data types as parameters and return simple types, such functions appear in the Libraries window, and can be used in mappings as any other function available in MapForce. The mapping output of imported Java and .NET functions can be previewed in the Output pane and the functions are available in generated code.

Notes:

- To import custom Java or .NET functions, you need compiled Java classes (.class) or the .NET.dll assembly files. Importing Java .jar files or .dll files that are not a .NET assembly is not supported.
- Compiled Java class (.class) files are supported when the mapping language is set to Java. Java Runtime Environment 7 or later must be installed on your computer. Only specific types and members are supported (see Java function support).
- .NET assembly files are supported when the mapping language is set to C#. The .NET assemblies may be written in .NET languages other than C# (for example, C++.NET or VB.NET), provided they use only the basic data types from the System Assembly as parameters and return types (see also .NET Function Support).
- Setting the mapping language to C++ is not supported if the mapping uses imported Java .class or .NET DLL assemblies.
- Importing functions from native C++ DLLs is limited and requires a special approach. For more information, see Referencing Java, C# and C++ Libraries Manually.
- Setting the mapping language to XSLT is not supported if the mapping uses imported Java .class or .NET DLL assemblies (a custom XSLT function that acts as an adapter would have to be written).
- All functions called from a MapForce mapping should be “idempotent” (this means that they should return the same value each time the function is called with the same input parameters). The exact order and the number of times a function is called by MapForce is undefined.
- In case of Java, the imported class files and their packages do not need to be added to the CLASSPATH variable, since the Built-in execution engine, as well as generated Java code, will automatically add imported packages to the Java engine’s classpath or to Ant, respectively. However, any dependencies of the imported class files and packages will not be handled automatically. Therefore, if imported Java class files or packages depend on other class files, be sure to add the parent directories of all dependent packages to the CLASSPATH environment variable.

Java function support

Top-level classes, static member classes and non-static member classes are supported:

- `new <classname>(<arg1>, <arg2>, ...)`
- `<object>.new <member-class>(<arg1>, <arg2>, ...)`

Member functions and static functions are supported:

- `<function>({<arg1>, <arg2>, ...})`
- `<object>.<method>({<arg1>, <arg2>, ...})`
Supported connections between XML Schema and Java types:

<table>
<thead>
<tr>
<th>Schema type</th>
<th>Java type</th>
</tr>
</thead>
<tbody>
<tr>
<td>xs:string</td>
<td>String</td>
</tr>
<tr>
<td>xs:byte</td>
<td>byte</td>
</tr>
<tr>
<td>xs:short</td>
<td>short</td>
</tr>
<tr>
<td>xs:int</td>
<td>int</td>
</tr>
<tr>
<td>xs:long</td>
<td>long</td>
</tr>
<tr>
<td>xs:boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>xs:float</td>
<td>float</td>
</tr>
<tr>
<td>xs:double</td>
<td>double</td>
</tr>
<tr>
<td>xs:decimal</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>xs:integer</td>
<td>java.math.BigInteger</td>
</tr>
</tbody>
</table>

Connections in both directions are possible. Other Java types (including array types) are not supported. Methods using such parameters or return values, will be ignored.

Object types are supported by calling their constructor, or as a return value of a method. They can be mapped to other Java methods. Manipulating the object using MapForce means is not possible.

.NET function support

Top-level classes and member classes are supported:

- new <classname>(<arg1>, <arg2>, ...)

Member functions and static functions are supported:

- <function>({<arg1>, <arg2>, ...})
- <object>.<method>({<arg1>, ...})

Supported connections between XML Schema and .NET/C# types:

<table>
<thead>
<tr>
<th>Schema type</th>
<th>.NET type</th>
<th>C# type</th>
</tr>
</thead>
<tbody>
<tr>
<td>xs:string</td>
<td>System.String</td>
<td>string</td>
</tr>
<tr>
<td>xs:byte</td>
<td>System.SByte</td>
<td>sbyte</td>
</tr>
<tr>
<td>xs:short</td>
<td>System.Int16</td>
<td>short</td>
</tr>
<tr>
<td>xs:int</td>
<td>System.Int32</td>
<td>int</td>
</tr>
<tr>
<td>xs:long</td>
<td>System.Int64</td>
<td>long</td>
</tr>
<tr>
<td>xs:unsignedByte</td>
<td>System.Byte</td>
<td>byte</td>
</tr>
<tr>
<td>xs:unsignedShort</td>
<td>System.UInt16</td>
<td>ushort</td>
</tr>
</tbody>
</table>
### Functions

#### Importing Custom Java and .NET Libraries

<table>
<thead>
<tr>
<th>Schema type</th>
<th>.NET type</th>
<th>C# type</th>
</tr>
</thead>
<tbody>
<tr>
<td>xs:unsignedInt</td>
<td>System.UInt32</td>
<td>uint</td>
</tr>
<tr>
<td>xs:unsignedLong</td>
<td>System.UInt64</td>
<td>ulong</td>
</tr>
<tr>
<td>xs:boolean</td>
<td>System.Boolean</td>
<td>bool</td>
</tr>
<tr>
<td>xs:float</td>
<td>System.Single</td>
<td>float</td>
</tr>
<tr>
<td>xs:double</td>
<td>System.Double</td>
<td>double</td>
</tr>
<tr>
<td>xs:decimal</td>
<td>System.Decimal</td>
<td>decimal</td>
</tr>
</tbody>
</table>

Connections in both directions are possible. Other .NET/C# types (including array types) are not supported. Methods using such parameters or return values will be ignored.

Object types are supported by calling their constructor, or as a return value of a method. They can be mapped to other .NET methods. Manipulating the object using MapForce means is not possible.

### 8.6.1 Example: Import Custom Java Class

This example illustrates how to import a custom Java .class file into MapForce.

**Note:** Java SE 7 Runtime Environment or later is required to complete this example.

**To add the Java .class file as MapForce library:**

1. Set the transformation language to Java (see Selecting a Transformation Language).
2. Click the **Add/Remove Libraries** button in the lower area of the Libraries window.
3. Next to Libraries, click **Add**, and select the `Format.class` file from the `...\MapForceExamples\Java\Format` directory. A message appears telling you that a new library has been added. The imported library is now visible in the Libraries window.

![Libraries](image)

**To preview the mapping output in MapForce:**

1. Open the `FormatNumber.mfd` file available in the `...\MapForceExamples\Java` folder.
2. Click the Output button to see the result of the mapping.
To run the mapping in Java:

1. On the File menu, click Generate Code In | Java.
2. Select a target directory where the code should be generated, and click OK.
3. Import the generated libraries into your Java project and build the Java application (for an example, see Example: Build a Java application with Eclipse and Ant).

8.6.2 Example: Import Custom .NET DLL Assembly

This example illustrates how to import into MapForce a custom .NET DLL assembly created in C#. The source code of this sample is available at the following path: <Documents>\Altova\MapForce2019\MapForceExamples\C#\Format. The .dll assembly file that will be imported into MapForce is in the ..\bin\Debug directory. You can also open the solution .sln file in Visual Studio and compile a new .dll file using the latest .NET Framework version.

Note: To run this example, .NET Framework 2.0 or later must be installed on your computer. If it's not already enabled, you can enable it from Control Panel > Programs and Features > Turn Windows features on and off > .NET Framework <version>.

To import the .NET assembly file:

1. Set the transformation language to C# (see Selecting a Transformation Language).
2. Click the Add/Remove Libraries button in the lower area of the Libraries window.
3. Next to Libraries, click Add, and select the Format.dll file from the ...
\MapForceExamples\C#\Format\bin\Debug directory. A message appears telling you that a new library has been added. The imported library is now visible in the Libraries window.
To preview the mapping output:

1. Open the **FormatNumber.mfd** file available in the ...\MapForceExamples\C# folder.
2. Click the Output button to see the result of the mapping.

<table>
<thead>
<tr>
<th></th>
<th>Start date,End date,Region,Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2008-01-01,2008-01-31,CA,&quot;110.400.00&quot;</td>
</tr>
<tr>
<td>2</td>
<td>2008-01-01,2008-01-31,MA,&quot;75.300.00&quot;</td>
</tr>
<tr>
<td>3</td>
<td>2008-02-01,2008-02-29,CA,&quot;114.300.00&quot;</td>
</tr>
<tr>
<td>4</td>
<td>2008-02-01,2008-02-29,MA,&quot;65.200.00&quot;</td>
</tr>
<tr>
<td>5</td>
<td>2008-03-01,2008-03-31,CA,&quot;134.200.00&quot;</td>
</tr>
<tr>
<td>6</td>
<td>2008-03-01,2008-03-31,MA,&quot;96.100.00&quot;</td>
</tr>
<tr>
<td>7</td>
<td>2008-04-01,2008-04-30,CA,&quot;107.300.00&quot;</td>
</tr>
<tr>
<td>8</td>
<td>2008-04-01,2008-04-30,MA,&quot;112.100.00&quot;</td>
</tr>
<tr>
<td>9</td>
<td>2008-05-01,2008-05-31,CA,&quot;114.400.00&quot;</td>
</tr>
<tr>
<td>10</td>
<td>2008-05-01,2008-05-31,MA,&quot;93.800.00&quot;</td>
</tr>
</tbody>
</table>

To run the mapping from a custom C# application:

1. On the **File** menu, click **Generate Code In | C#**.
2. Select a target directory where the code should be generated, and click OK.
3. Build the application with Visual Studio, and run the generated console application (see also **Generating C# code**).
8.7 Referencing Java, C# and C++ Libraries Manually

As an alternative approach to importing custom libraries into MapForce directly, you can create references to them using a custom .mff file (MapForce Function File) recognized by MapForce. The .mff library file is essentially an XML file where you manually define the linking between class definitions in your custom code and MapForce. Once you create the custom .mff file, you can import it into MapForce, similar to how you would import a .NET DLL or Java class file.

Notes:

- For an imported function to appear in the Libraries window, its return type and parameters must be of a simple type. For a list of data types available for each language, see Data Type Mapping.
- When you import function libraries from custom .mff files, the preview of the mapping directly in MapForce (by clicking the Output button) is limited. For libraries written in C++, preview of the mapping in MapForce is not supported. In case of Java and C#, preview is available when your library uses native language types, but it is not available if your library imports the Altova generated classes. Note, however, that you can generate code in the specific language targeted by your library. The custom functions will be available in the generated code, enabling you to run the mapping from the generated code.
- The exact order in which functions are called by the generated mapping code is undefined. MapForce may cache calculated results for reuse, or evaluate expressions in any order. It is therefore strongly recommended to use only custom functions that have no side effects.
- It is important to distinguish between user-defined functions and custom function libraries. User-defined functions are created graphically in a mapping, and they cannot and need not be saved to an *.mff file, because they are saved together with the mapping .mfd file where they have been created. For more information, see Calling and Importing User-Defined Functions.
- If you are upgrading from a MapForce version earlier than 2010, you may need to update the data types used in your custom functions (see Data Type Mapping).

For instructions on how to create and configure a custom .mff file, see Configuring the .mff File. For examples, see:

- Example: Create a Custom C# Library
- Example: Create a Custom C++ Library
- Example: Create a Custom Java Library

8.7.1 Configuring the .mff File

The MapForce Function File (.mff) is a configuration file in XML format that makes it possible to adapt functions from custom Java, C#, or C++ libraries into MapForce, so that they appear in the Libraries window. An .mff file essentially mediates between your custom libraries and MapForce, and it must be configured to specify a) the interfaces to the custom functions and b) where the implementation can be found in generated code. This topic provides instructions on how to do this.

Note: The *.mff library files must be valid against the mff.xsd schema file found in the Altova \MapForce2019\MapForceExamples folder, relative to your (My) Documents folder.
The `mff.xsd` schema defines the custom library configuration and is for internal use only. Altova GmbH retains the right to change this file format with new releases.

The following code listing illustrates a sample `.mff` file for C#:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:xs="http://www.w3.org/2001/XMLSchema"
xsi:noNamespaceSchemaLocation="mff.xsd" version="8" library="helloworld">
  <implementations>
    <implementation language="cs">
      <setting name="namespace" value="HelloWorldLibrary"/>
      <setting name="class" value="Greetings"/>
      <setting name="reference" value="C:\HelloWorldLibrary\HelloWorldLibrary.dll"/>
    </implementation>
  </implementations>
  <group name="string functions">
    <component name="hello">
      <sources>
        <datapoint name="greeting_type" type="xs:boolean"/>
      </sources>
      <targets>
        <datapoint name="result" type="xs:string"/>
      </targets>
      <implementations>
        <implementation language="cs">
          <function name="HelloFunction"/>
        </implementation>
      </implementations>
      <description>
        <short>result = hello(greeting_type)</short>
        <long>Returns a greeting sentence according to the given greeting_type.\</long>
      </description>
    </component>
  </group>
</mapping>
```

The image below shows a custom `.mff` file may look after import into MapForce. Notice that the custom library "helloworld" appears as a library entry (sorted alphabetically), containing the "hello" string function.
The steps needed to adapt the mff file to suit your needs are described below.

**Configuring the library name**

The library name is found in the .mff file line shown below. By convention, the library name is written in lowercase letters.

```xml
<mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:xs="http://www.w3.org/2001/XMLSchema"
xsi:noNamespaceSchemaLocation="mff.xsd" version="8" library="helloworld">

The entry that will appear in the Libraries window will be called "helloworld".

**Configuring the language implementations**

The `<implementations>` element is mandatory element which specifies which languages your library should support, and it must be added as child of `<mapping>`, for example:

```xml
<implementations>
  <implementation language="cs">
    <setting name="namespace" value="HelloWorldLibrary"/>
    <setting name="class" value="Greetings"/>
    <setting name="reference" value="C:\HelloWorldLibrary\HelloWorldLibrary.dll"/>
  </implementation>
</implementations>
```

The settings within each `<implementation>` allow the generated code to call the specific functions defined in Java, C++ or C#.
An .mff file can be written so that it targets more than one programming language. In this case, every additional language must contain an additional `<implementation>` element. The specific settings for each programming language are discussed below.

### Java

```xml
...<implementation language="java">
  <setting name="package" value="com.hello.functions"/>
  <setting name="class" value="Hello"/>
</implementation>
...
```

It is important for the generated code to be able to find your `Hello.class` file. Therefore, make sure that your class is in the Java classpath. The default Java classpath is found in the system environment variables.

Note that it is only possible to have one class per *.mff file when working with custom Java libraries.

### C#

```xml
...<implementation language="cs">
  <setting name="namespace" value="HelloWorldLibrary"/>
  <setting name="class" value="Hello"/>
  <setting name="reference" value="C:\HelloWorldLibrary\HelloWorldLibrary.dll"/>
</implementation>
...
```

For C#, it is important that the namespace in the code corresponds to the namespace defined in the .mff file (in the code listing above, the namespace is `HelloWorldLibrary`). The same is true for the class name (in the code listing above, the class name is `Hello`). The third setting, `reference`, provides the path of the dll that is to be linked to the generated code.

### C++

```xml
...<implementation language="cpp">
  <setting name="namespace" value="helloworld"/>
  <setting name="class" value="Greetings"/>
  <setting name="path" value="C:\HelloWorldLibrary"/>
  <setting name="include" value="Greetings.h"/>
  <setting name="source" value="Greetings.cpp"/>
</implementation>
...
```

For the C++ sample listing above, note the following:

- `namespace` is the namespace in which your `Greetings` class will be defined. It must be
equal to the library name.

- path is the path in which the include and the source files are to be found.
- When code for a mapping is generated, the include and source files will be copied to the directory targetdir/libraryname (defined when selecting the menu option File | Generate xxx code, and selecting the directory) and included in the project file.

All the include files you supply will be included in the generated algorithm.

Adding a component

In the Libraries window of the MapForce graphics user interface, each function appears nested under a function group, for example "string functions". In the .mff file, a function corresponds to a <component> element. Conversely, each <component> must be nested under a <group> element, for example:

```
...<group name="string functions">
  <component name="hello">
    ...
  </component>
</group>
...```

The code shown below defines a sample function (component) called hello.

```
...<component name="hello">
  <sources>
    <datapoint name="greeting_type" type="xs:boolean"/>
  </sources>
  <targets>
    <datapoint name="result" type="xs:string"/>
  </targets>
  <implementations>
    ...
  </implementations>
  <description>
    <short>result = hello(greeting_type)</short>
    <long>Returns a greeting sentence according to the given greeting_type.</long>
  </description>
</component>
...```

Here is how the component above would look in MapForce:

![MapForce component](image)

In the code listing above, a <datapoint> can be loosely defined as the input or output parameter of a function (also known as input or output connector). The type argument of the <datapoint>
specifies the data type of the parameter (or the data type of the return value).

Only one target datapoint is allowed for each function. There is no limitation as to how many source datapoints you can define.

The data type of each datapoint must be one of the XML Schema types (for example, `xs:string`, `xs:integer`, etc.) These data types have to correspond to the data types of the function’s parameters you defined in your Java, C++ or C# library. For the mapping of XML Schema datatypes to language types, see Data Type Mapping.

Functions are accompanied by short and long descriptions in the library window. The short description is always shown to the right of the function name, while the long description is displayed as a ToolTip when you place the mouse cursor over the short description.

Short description:

<table>
<thead>
<tr>
<th>helloworld</th>
<th>result = hello(greeting_type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello</td>
<td></td>
</tr>
</tbody>
</table>

Long description:

<table>
<thead>
<tr>
<th>helloworld</th>
<th>Returns a greeting sentence according to the given greeting_type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello</td>
<td></td>
</tr>
</tbody>
</table>

Defining language implementations

We are now at the point where we need to make a connection between the function in the Libraries window, and the function in the custom Java, C# or C++ classes. This is achieved through the `<implementation>` element.

As previously stated, one function may have multiple implementation elements – one for each supported programming language. A function may be called “helloFunction” in Java, or “HelloFunctionResponse” in C++. This is why you need to specify a separate function name for each programming language. A function for each of the three programming languages might look like the following:

```xml
...<component name="hello">
...<implementations>
  <implementation language="cs">
    <function name="HelloFunction"/>
  </implementation>
  <implementation language="java">
    <function name="helloFunction"/>
  </implementation>
  <implementation language="cpp">
    <function name="HelloFunctionResponse"/>
  </implementation>
...</implementations>
..."
The value you supply as function name must exactly match the name of the method in the Java, C# or C++ class.

8.7.2 Importing the .mff File Into MapForce

After you have created a custom .mff file (see Configuring the .mff File), you can import it into MapForce as follows:

1. On the Tools menu, click Options. (Alternatively, click Add/Remove Libraries in the lower area of the Libraries window.)
2. Next to Libraries, click Add, and select the custom .mff file.

The imported library becomes visible in the Libraries window after you set the mapping language to a language targeted by the custom library.

If you save the *.mff file in the ...\Altova\MapForce2019\MapForceLibraries folder relative to the Program Files (or Program Files (x86) folder), then the library is automatically loaded into Libraries window when you start MapForce. Libraries and their functions can be toggled on or off, by deleting or adding the respective library file (*.mff).

8.7.3 Data Type Mapping

The following table lists the data types supported as function return types and parameter types when you create custom .mff files that adapt your Java, C#, and C++ libraries to MapForce. The table lists both native and non-native data types. Note that, if you need support for non-native data types such as Altova date, time and duration types, your custom Java and C# libraries must include a reference to Altova libraries. In case of C++, Altova libraries must always be imported. For information about how to generate the Altova libraries, see Code Generator.

<table>
<thead>
<tr>
<th>XML Schema Type</th>
<th>Java Type</th>
<th>C# Type</th>
<th>C++ Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>anyAtomicType</td>
<td>String</td>
<td>string</td>
<td>string_type</td>
</tr>
<tr>
<td>anySimpleType</td>
<td>String</td>
<td>string</td>
<td>string_type</td>
</tr>
<tr>
<td>anyURI</td>
<td>String</td>
<td>string</td>
<td>string_type</td>
</tr>
<tr>
<td>base64Binary</td>
<td>byte[]</td>
<td>byte[]</td>
<td>altova::mapforce::blob</td>
</tr>
<tr>
<td>boolean</td>
<td>boolean</td>
<td>bool</td>
<td>bool</td>
</tr>
<tr>
<td>byte</td>
<td>int</td>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td>XML Schema Type</td>
<td>Java Type</td>
<td>C# Type</td>
<td>C++ Type</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------</td>
<td>------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>date</td>
<td><code>com.altova.types.DateTime</code></td>
<td><code>Altova.Types.DateTime</code></td>
<td><code>altova::DateTime</code></td>
</tr>
<tr>
<td>dateTime</td>
<td><code>com.altova.types.DateTime</code></td>
<td><code>Altova.Types.DateTime</code></td>
<td><code>altova::DateTime</code></td>
</tr>
<tr>
<td>dayTimeDuration</td>
<td><code>com.altova.types.Duration</code></td>
<td><code>Altova.Types.Duration</code></td>
<td><code>altova::Duration</code></td>
</tr>
<tr>
<td>decimal</td>
<td><code>java.math.BigDecimal</code></td>
<td>decimal</td>
<td>double</td>
</tr>
<tr>
<td>double</td>
<td>double</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>duration</td>
<td><code>com.altova.types.Duration</code></td>
<td><code>Altova.Types.Duration</code></td>
<td><code>altova::Duration</code></td>
</tr>
<tr>
<td>ENTITIES</td>
<td><code>String</code></td>
<td>string</td>
<td><code>string_type</code></td>
</tr>
<tr>
<td>ENTITY</td>
<td><code>String</code></td>
<td>string</td>
<td><code>string_type</code></td>
</tr>
<tr>
<td>float</td>
<td>double</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>gDay</td>
<td><code>com.altova.types.DateTime</code></td>
<td><code>Altova.Types.DateTime</code></td>
<td><code>altova::DateTime</code></td>
</tr>
<tr>
<td>gMonth</td>
<td><code>com.altova.types.DateTime</code></td>
<td><code>Altova.Types.DateTime</code></td>
<td><code>altova::DateTime</code></td>
</tr>
<tr>
<td>gMonthDay</td>
<td><code>com.altova.types.DateTime</code></td>
<td><code>Altova.Types.DateTime</code></td>
<td><code>altova::DateTime</code></td>
</tr>
<tr>
<td>gYear</td>
<td><code>com.altova.types.DateTime</code></td>
<td><code>Altova.Types.DateTime</code></td>
<td><code>altova::DateTime</code></td>
</tr>
<tr>
<td>gYearMonth</td>
<td><code>com.altova.types.DateTime</code></td>
<td><code>Altova.Types.DateTime</code></td>
<td><code>altova::DateTime</code></td>
</tr>
<tr>
<td>hexBinary</td>
<td><code>byte[]</code></td>
<td><code>byte[]</code></td>
<td><code>altova::mapforce::blob</code></td>
</tr>
<tr>
<td>ID</td>
<td><code>String</code></td>
<td>string</td>
<td><code>string_type</code></td>
</tr>
<tr>
<td>IDREF</td>
<td><code>String</code></td>
<td>string</td>
<td><code>string_type</code></td>
</tr>
<tr>
<td>IDREFS</td>
<td><code>String</code></td>
<td>string</td>
<td><code>string_type</code></td>
</tr>
<tr>
<td>int</td>
<td>int</td>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td>integer</td>
<td><code>java.math.BigInteger</code></td>
<td>decimal</td>
<td><code>__int64</code></td>
</tr>
<tr>
<td>language</td>
<td><code>String</code></td>
<td>string</td>
<td><code>string_type</code></td>
</tr>
<tr>
<td>long</td>
<td>long</td>
<td>long</td>
<td><code>__int64</code></td>
</tr>
<tr>
<td>Name</td>
<td><code>String</code></td>
<td>string</td>
<td><code>string_type</code></td>
</tr>
</tbody>
</table>
### 8.7.4 Example: Create a Custom C# Library

This topic describes how to create a sample C# library and configure the .mff file so that it appears in the Libraries window of MapForce.

1. Create a new class library project in Visual Studio. Notice that the function has been defined as `public static`.

#### XML Schema Type | Java Type | C# Type | C++ Type
--- | --- | --- | ---
NCName | String | string | string_type
negativeInteger | java.math.BigInteger | decimal | __int64
NMTOKEN | String | string | string_type
NMTOKENS | String | string | string_type
nonNegativeInteger | java.math.BigInteger | decimal | unsigned __int64
nonPositiveInteger | java.math.BigInteger | decimal | __int64
normalizedString | String | string | string_type
NOTATION | String | string | string_type
positiveInteger | java.math.BigInteger | decimal | unsigned __int64
QName | javax.xml.namespace.QName | Altova.Types.QName | altova::QName
short | int | int | int
string | String | string | string_type
time | com.altova.types.DateTime | Altova.Types.DateTime | altova::DateTime
token | String | string | string_type
unsignedByte | long | ulong | unsigned __int64
unsignedInt | long | ulong | unsigned __int64
unsignedLong | java.math.BigInteger | ulong | unsigned __int64
unsignedShort | long | ulong | unsigned __int64
untypedAtomic | String | string | string_type
yearMonthDuration | com.altova.types.Duration | Altova.Types.Duration | altova::Duration
using System;

namespace HelloWorldLibrary
{
    public class Greetings
    {
        public static string HelloFunction(bool GreetingType)
        {
            if (GreetingType)
                return "Hello World!";
            return "Hello User!";
        }
    }
}

2. If you need special XML Schema types (such as date and duration), you will need add a reference from your Visual Studio project to the Altova.dll library. To obtain this library, generate C# code from a mapping without custom functions. The Altova.dll file will be located in the ..\Altova\bin\debug directory relative to the directory where the code was generated. To add the reference to Altova.dll in Visual Studio, on the Project menu, click Add Reference and browse for the Altova.dll file. Also, add to your code the following line: using Altova.Types; . For information about how XML Schema types map to C# types, see Data Type Mapping.

3. Build your Visual Studio project. The HelloWorldLibrary.dll is generated in your project output directory.

4. Using an XML editor, create a new .mff file and validate it against the ..\Program Files\MapForceLibraries\mff.xsd folder. Make sure that the text highlighted below points to the HelloWorldLibrary.dll file. For more information, see Configuring the .mff File.

```xml
<?xml version="1.0" encoding="UTF-8"?>
    <implementations>
        <implementation language="cs">
            <setting name="namespace" value="HelloWorldLibrary"/>
            <setting name="class" value="Greetings"/>
            <setting name="reference" value="C:\Projects\HelloWorldLibrary.dll"/>
        </implementation>
    </implementations>
    <group name="Greetings">
        <component name="HelloFunction">
            <sources>
                <datapoint name="greeting_type" type="xs:boolean"/>
            </sources>
            <targets>
                <datapoint name="result" type="xs:string" />
            </targets>
            <implementations>
                <implementation language="cs">
```
You have now finished creating a custom library and the .mff file which adapts it to MapForce. The custom .mff file can now be used in MapForce (see Importing the .mff File Into MapForce).

8.7.5 Example: Create a Custom C++ Library

This topic describes how to create a sample C++ library and configure a .mff file for it so that the library appears in the Libraries window of MapForce.

1. Create a header (.h) file for your class library. The following code listing illustrates a sample header file called Greetings.h.

```cpp
#ifndef HELLOWORLDLIBRARY_GREETINGS_H_INCLUDED
#define HELLOWORLDLIBRARY_GREETINGS_H_INCLUDED

#pragma once

// _MSC_VER > 1000

using namespace altova;

namespace helloworld {

class ALTOVA_DECLSPECIFIER Greetings {

public:
    static string_type HelloFunctionResponse(bool greetingType);
};

} // namespace HelloWorldLibrary

#endif // HELLOWORLDLIBRARY_GREETINGS_H_INCLUDED
```

Notice that the function has been declared as static, and that the namespace `altova` is imported. Remember to write `ALTOVA_DECLSPECIFIER` in front of the class name, this ensures that your classes will compile correctly—whether you use dynamic or static linkage in subsequently generated code.

2. Create a .cpp file with the same name as the header file. The .cpp file must be in the same directory as the .h file. The following code listing illustrates a sample .cpp file.
called **Greetings.cpp** that includes the **Greetings.h** file created previously:

```cpp
#include "StdAfx.h"
#include "../Altova/Altova.h"
#include "../Altova/AltovaException.h"
#include "../Altova/SchemaTypes.h"
#include "Greetings.h"

namespace helloworld {

    string_type Greetings::HelloFunctionResponse(bool greetingType)
    {
        if ( greetingType )
            return _T("Hello World!");
        return _T("Hello User!");
    }
}
```

Notice the lines that import the **StdAfx.h** and several Altova libraries. These lines must be left unchanged. The paths to the Altova libraries is correct; in the generated code, these paths will point to the respective files.

In contrast to Java or C#, you do not need to compile your source C++ files. They will be copied to the generated code, and are compiled with the rest of the generated mapping code.

3. Using an XML editor, create a new **.mff** file and validate it against the **..\Program Files\MapForceLibraries\mff.xsd** folder. Make sure that the text highlighted below points to the directory of the header and cpp files created previously. Remember that the namespace and function names and data types defined here must correspond to those in the C++ code, as described in **Configuring the .mff File**. For information about data type support, see **Data Type Mapping**.

```xml
<?xml version="1.0" encoding="UTF-8"?>
    <implementations>
        <implementation language="cpp">
            <setting name="namespace" value="helloworld"/>
            <setting name="class" value="Greetings"/>
            <setting name="path" value="C:\Projects\HelloWorld"/>
            <setting name="include" value="Greetings.h"/>
            <setting name="source" value="Greetings.cpp"/>
        </implementation>
    </implementations>
    <group name="Greetings">
        <component name="HelloFunctionResponse">
```

© 2019 Altova GmbH

Altova MapForce 2019 Enterprise Edition
You have now finished creating a custom library and the .mff file which adapts it to MapForce. The custom .mff file can now be used in MapForce (see Importing the .mff File Into MapForce). Remember that, in order to execute mappings that use native C++ libraries, you will need to generate C++ code and run the mapping from your C++ code or application.

Resolving C++ compile errors

If you get a compiler error at the line shown below, modify the project properties to include a reference to the msado15.dll file.

```c
#import "msado15.dll" rename("EOF", "EndOfFile")
```

In Visual Studio 2008:

1. On the Tools menu, click Options.
2. Expand Projects and Solutions > VC++ Directories.
3. Under "Show directories for", select Include files, and add a new entry that points to the directory where msado15.dll file is located (usually, C:\Program Files\Common Files\System\ADO).
4. Build the project.

### 8.7.6 Example: Create a Custom Java Library

This topic describes how to create a sample Java library and configure a .mff file for it so that the library appears in the Libraries window of MapForce.

1. Create a new Java project in your preferred development environment (for example, Eclipse).
2. Add to the project a new package called com.hello.functions which consists of a class called Hello. In the code listing below, notice that the HelloFunction function has been defined as public static.
Referencing Java, C# and C++ Libraries Manually

3. Optionally, if your project needs support for special schema types such as date, time, and duration, import the com.altova.types package. To obtain this package, generate Java code from a mapping without custom functions.

```java
package com.hello.functions;

public class Hello {
    public static String HelloFunction ( boolean greetingType ) {
        if ( greetingType )
            return "Hello World!";
        return "Hello User!";
    }
}
```

4. Compile your custom library to a class file, and add it to the Java classpath.
5. Using an XML editor, create a new .mff file and validate it against the ..\Program Files\MapForceLibraries\mff.xsd folder. Make sure that the text highlighted below points to the namespace and class defined previously in the Java code. For more information, see Configuring the .mff File.

```xml
<?xml version="1.0" encoding="UTF-8"?>
    <implementations>
        <implementation language="java">
            <setting name="namespace" value="com.hello.functions"/>
            <setting name="class" value="Hello"/>
        </implementation>
    </implementations>
    <group name="Greetings">
        <component name="HelloFunction">
            <sources>
                <datapoint name="greeting_type" type="xs:boolean"/>
            </sources>
            <targets>
                <datapoint name="result" type="xs:string"/>
            </targets>
            <implementations>
                <implementation language="java">
                    <function name="HelloFunction"/>
                </implementation>
            </implementations>
            <description>
                <short>result = hello(greeting_type)</short>
                <long>Returns a greeting according to the given greeting type.</long>
            </description>
        </component>
    </group>
</mapping>
```
You have now finished creating a custom library and the .mff file which adapts it to MapForce. The custom .mff file can now be used in MapForce (see Importing the .mff File Into MapForce).
8.8 Regular Expressions

MapForce can use regular expressions in the pattern parameter of the match-pattern and tokenize-regexp functions, to find specific strings. Some regular expression functionality is also available when you need to filter the nodes on which a node function or default should apply, see Applying Node Functions and Defaults Conditionally.

The regular expression syntax and semantics for XSLT and XQuery are identical to those defined in https://www.w3.org/TR/xmlschema-2/. Please note that there are slight differences in regular expression syntax between the various programming languages.

**Terminology**
- **input**: the string that the regex works on
- **pattern**: the regular expression
- **flags**: optional parameter to define how the regular expression is to be interpreted
- **result**: the result of the function

**Tokenize-regexp** returns a sequence of strings. The connection to the Rows item creates one row per item in the sequence.

**regex syntax**

**Literals** e.g. a single character:
- e.g. The letter “a” is the most basic regex. It matches the first occurrence of the character “a” in the string.

**Character classes []**
- This is a set of characters enclosed in square brackets.
- **One**, and only one, of the characters in the square brackets are matched.

- pattern  `[aeiou]`
  Matches a lowercase vowel.

- pattern  `[mj]ust`
  Matches must or just

Please note that “pattern” is case sensitive, a lower case a does not match the uppercase A.

**Character ranges [a-z]**
- Creates a range between the two characters. Only one of the characters will be matched at one
time.

pattern \[a-z\]
Matches any lowercase characters between a and z.

negated classes [^]
using the caret as the first character after the opening bracket, negates the character class.

pattern \[^a-z\]
Matches any character not in the character class, including newlines.

Meta characters "."  
Dot meta character matches any single character (except for newline)

pattern .
Matches any single character.

Quantifiers ? + * {}
Quantifiers define how often a regex component must repeat within the input string, for a match to occur.

? 
zero or one preceding string/chunk is optional

+ 
one or more preceding string/chunks may match one or more times

* 
zero or more preceding string/chunks may match zero or more times

{} 
min / max repetitions no. of repetitions a string/chunks has to match  
e.g. mo{1,3} matches mo, moo, moo.

() 
subpatterns parentheses are used to group parts of a regex together.

|  
Alternation/or allows the testing of subexpressions form left to right.  
(horse|make) sense - will match "horse sense" or "make sense"

Flags  
These are optional parameters that define how the regular expression is to be interpreted.  
Individual letters are used to set the options, i.e. the character is present. Letters may be in any order and can be repeated.
If present, the matching process will operate in the "dot-all" mode.

The meta character "." matches any character whatsoever. If the input string contains "hello" and "world" on two different lines, the regular expression "hello*world" will only match if the s flag/character is set.

If present, the matching process operates in multi-line mode.

In multi-line mode the caret ^ matches the start of any line, i.e. the start of the entire string and the first character after a newline character.

The dollar character $ matches the end of any line, i.e. the end of the entire string and the character immediately before a newline character.

Newline is the character #x0A.

If present, the matching process operates in case-insensitive mode.

The regular expression [a-z] plus the i flag would then match all letters a-z and A-Z.

If present, whitespace characters are removed from the regular expression prior to the matching process. Whitespace chars. are #x09, #x0A, #x0D and #x20.

**Exception:** Whitespace characters within character class expressions are not removed e.g. [#x20].

**Note:** When generating code, the advanced features of the regex syntax might differ slightly between the various languages, please see the specific regex documentation for your language.
8.9  Function Library Reference

This reference chapter describes the MapForce built-in functions available in the Libraries pane, organized by library.

The availability of function libraries in the Libraries pane depends on the transformation language you have selected (see Selecting a transformation language ). The core library is a collection of functions available in C++, C#, Java languages and in at least one of the following: XQuery, XPath, or XSLT. The lang library is dedicated to functions available in C++, C#, and Java languages. Other libraries contain functions associated with each separate type of output.

XPath 2.0 restrictions: Several XPath 2.0 functions dealing with sequences are currently not available.

8.9.1  core | aggregate functions

Aggregate functions perform operations on a set, or sequence, of input values. The input data for min, max, sum and avg is converted to the decimal data type for processing.

- The input values must be connected to the values parameter of the function.
- A context node (item) can be connected to the parent-context parameter to override the default context from which the input sequence is taken. The parent-context parameter is optional.
- The result of the function is connected to the specific target item.

The mapping shown below is available as Aggregates.mfd in the ...\Tutorial folder and shows how these functions are used.

Aggregate functions have two input items.
- values (nodes/rows) is connected to the source item that provides the data, in this case Number.
- parent-context is connected to the item you want to iterate over, i.e. the context, in this case over all Customers. The parameter is, however, optional.
The input instance in this case is an XML file containing the following data:

```
<Customers>
  <Customer>
    <Number>1</Number>
    <FirstName>Fred</FirstName>
  </Customer>
  <Customer>
    <Number>2</Number>
    <FirstName>John</FirstName>
  </Customer>
  <Customer>
    <Number>3</Number>
    <FirstName>Ann</FirstName>
  </Customer>
  <Customer>
    <Number>4</Number>
    <FirstName>Long</FirstName>
  </Customer>
</Customers>
```

- The source data supplied to the values item is the number sequence 2,4,6,8.
- The output component supplied to the values item is a simple text file.
  Clicking the Output tab for the above mapping delivers the following result:

<table>
<thead>
<tr>
<th></th>
<th>2, 8, 4, 20, 5</th>
</tr>
</thead>
</table>

min=2, max=8, count=4, sum=20 and avg=5.

### 8.9.1.1 avg

Returns the average value of all values within the input sequence. The average of an empty set is an empty set. Not available in XSLT1.
### Functions

**Function Library Reference**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>parent-context</strong></td>
<td>Optional argument. Supplies the parent context. See also Overriding the Mapping Context.</td>
</tr>
<tr>
<td><strong>values</strong></td>
<td>This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric.</td>
</tr>
</tbody>
</table>

For an example of usage, see the mapping GroupTemperaturesByYear.mfd in the `<Documents>\Altova\MapForce2019\MapForceExamples` directory.

#### 8.9.1.2 count

Returns the number of individual items making up the input sequence. The count of an empty set is zero. Limited functionality in XSLT1.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>parent-context</strong></td>
<td>Optional argument. Supplies the parent context. See also Overriding the Mapping Context.</td>
</tr>
<tr>
<td><strong>nodes/rows</strong></td>
<td>This argument must be connected to the source item to be counted.</td>
</tr>
</tbody>
</table>

#### 8.9.1.3 max

Returns the maximum value of all numeric values in the input sequence. Note that this function returns an empty set if the strings argument is an empty set. Not available in XSLT1.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>parent-context</strong></td>
<td>Optional argument. Supplies the parent context. See also Overriding the Mapping Context.</td>
</tr>
<tr>
<td>Argument</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>values</td>
<td>This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric. To get the maximum from a sequence of strings, use the <strong>max-string</strong> function.</td>
</tr>
</tbody>
</table>

For an example of usage, see the mapping `GroupTemperaturesByYear.mfd` in the `<Documents>\Altova\MapForce2019\MapForceExamples` directory.

### 8.9.1.4 max-string

Returns the maximum value of all string values in the input sequence. For example, `max-string("a", "b", "c")` returns "c". This function is not available in XSLT1.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent-context</td>
<td>Optional argument. Supplies the <strong>parent context</strong>. See also Overriding the Mapping Context.</td>
</tr>
<tr>
<td>strings</td>
<td>This argument must be connected to a source item which supplies the actual data. The supplied argument value must be a sequence (zero or many) of <code>xs:string</code>.</td>
</tr>
</tbody>
</table>

Note that the function returns an empty set if the `strings` argument is an empty set.

### 8.9.1.5 min

Returns the minimum value of all numeric values in the input sequence. The minimum of an empty set is an empty set. Not available in XSLT1.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent-context</td>
<td>Optional argument. Supplies the <strong>parent context</strong>. See also Overriding the Mapping Context.</td>
</tr>
<tr>
<td>values</td>
<td>This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric. To get the minimum from a sequence of strings, use the <strong>min-string</strong> function.</td>
</tr>
</tbody>
</table>
For an example of usage, see the mapping GroupTemperaturesByYear.mfd in the \<Documents>\Altova\MapForce2019\MapForceExamples\ directory.

### 8.9.1.6 `min-string`

Returns the minimum value of all string values in the input sequence. For example, `min-string("a", "b", "c")` returns "a". This function is not available in XSLT1.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>parent-context</code></td>
<td>Optional argument. Supplies the parent context. See also Overriding the Mapping Context.</td>
</tr>
<tr>
<td><code>strings</code></td>
<td>This argument must be connected to a source item which supplies the actual data. The supplied argument value must be a sequence (zero or many) of <code>xs:string</code>.</td>
</tr>
</tbody>
</table>

Note that the function returns an empty set if the `strings` argument is an empty set.

### 8.9.1.7 `string-join`

Concatenates all the values of the input sequence into one string delimited by whatever string you choose to use as the delimiter. The string-join of an empty set is the empty string. Not available in XSLT1.

The example below contains four separate customer numbers 2 4 6 and 8. The constant character supplies a hash character "#" as the delimiter.

Result = 2#4#6#8
If you do not supply a delimiter, then the default is an empty string, i.e. no delimiter of any sort. Result = 2468.

### 8.9.1.8 sum

Returns the arithmetic sum of all values in the input sequence. The sum of an empty set is zero.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent-context</td>
<td>Optional argument. Supplies the parent context. See also Overriding the Mapping Context.</td>
</tr>
<tr>
<td>values</td>
<td>This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric.</td>
</tr>
</tbody>
</table>

See also [Example: Summing Node Values](#).

### 8.9.2 core | conversion functions

To support explicit data type conversion, several type conversion functions are available in the conversion library. Note that, in most cases, MapForce creates necessary conversions automatically and these functions need to be used only in special cases.

If the input nodes are of differing types, e.g. integer and string, you can use the conversion functions to force a string or numeric comparison.
In the example above the first constant is of type string and contains the string "4". The second constant contains the numeric constant 12. To be able to compare the two values explicitly the types must agree.

Adding a number function to the first constant converts the string constant to the numeric value of 4. The result of the comparisons is then "true".

Note that if the number function were not be used, i.e 4 would be connected directly to the a parameter, a string compare would occur, with the result being false.

8.9.2.1 boolean

Converts an input numeric value into a boolean (as well as a string to numeric - true to 1). E.g. 0 to "false", or 1 to "true", for further use with logical functions (equal, greater etc.) filters, or if-else functions.

8.9.2.2 format-date

Converts an xs:date input value into a string and formats it according to specified options.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The date to be formatted.</td>
</tr>
<tr>
<td>format</td>
<td>A format string identifying the way in which the date is to be formatted. This argument is used in the same way as the format argument in the formatDateTime function.</td>
</tr>
<tr>
<td>language</td>
<td>Optional argument. When supplied, the name of the month and the day of the week are returned in a specific language. Valid values:</td>
</tr>
</tbody>
</table>
In the following example, the output result is: "21 August 2014, Thursday". To translate this value to Spanish, set the value of the language argument to es.

8.9.2.3  format-dateTime

Converts a date and time value (xs:dateTime) into a string. The string representation of date and time is formatted according to the value of the format argument.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The xs:dateTime value to be formatted.</td>
</tr>
<tr>
<td>format</td>
<td>A format string identifying the way in which value is to be formatted.</td>
</tr>
<tr>
<td>language</td>
<td>Optional argument. When supplied, the name of the month and the day of the week are returned in a specific language. Valid values:</td>
</tr>
<tr>
<td>en (default)</td>
<td>English</td>
</tr>
<tr>
<td>es</td>
<td>Spanish</td>
</tr>
<tr>
<td>de</td>
<td>German</td>
</tr>
<tr>
<td>ja</td>
<td>Japanese</td>
</tr>
</tbody>
</table>

Note: If the function’s output (result) is connected to a node of type other than string, the formatting may be lost as the value is cast to the target type. This automatic cast can be disabled by unchecking the Cast target values to target types check box in the Component Settings of the target component (see Changing the Component Settings).

The format argument consists of a string containing so-called variable markers enclosed in
square brackets. Characters outside the square brackets are literal characters to be copied into
the result. If square brackets are needed as literal characters in the result, then they should be
doubled.

Each variable marker consists of a component specifier identifying which component of the date
or time is to be displayed, an optional formatting modifier, another optional presentation modifier
and an optional width modifier, preceded by a comma if it is present.

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Description</th>
<th>Default Presentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>year (absolute value)</td>
<td>four digits (2010)</td>
</tr>
<tr>
<td>M</td>
<td>month of the year</td>
<td>1-12</td>
</tr>
<tr>
<td>D</td>
<td>day of month</td>
<td>1-31</td>
</tr>
<tr>
<td>d</td>
<td>day of year</td>
<td>1-366</td>
</tr>
<tr>
<td>F</td>
<td>day of week</td>
<td>name of the day (language dependent)</td>
</tr>
<tr>
<td>W</td>
<td>week of the year</td>
<td>1-53</td>
</tr>
<tr>
<td>w</td>
<td>week of month</td>
<td>1-5</td>
</tr>
<tr>
<td>H</td>
<td>hour (24 hours)</td>
<td>0-23</td>
</tr>
<tr>
<td>h</td>
<td>hour (12 hour)</td>
<td>1-12</td>
</tr>
<tr>
<td>P</td>
<td>A.M. or P.M.</td>
<td>alphabetic (language dependent)</td>
</tr>
<tr>
<td>m</td>
<td>minutes in hour</td>
<td>00-59</td>
</tr>
<tr>
<td>s</td>
<td>seconds in minute</td>
<td>00-59</td>
</tr>
<tr>
<td>f</td>
<td>fractional seconds</td>
<td>numeric, one decimal place</td>
</tr>
<tr>
<td>Z</td>
<td>timezone as a time offset from UTC</td>
<td>+08:00</td>
</tr>
<tr>
<td>z</td>
<td>timezone as a time offset using GMT</td>
<td>GMT+n</td>
</tr>
</tbody>
</table>

The formatting modifier can be one of the following:

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>decimal numeric format with no leading zeros: 1, 2, 3, ...</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>01</td>
<td>decimal format, two digits: 01, 02, 03, ...</td>
<td>01, 02, 03</td>
</tr>
<tr>
<td>Character</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>N</td>
<td>name of component, upper case</td>
<td>MONDAY, TUESDAY 1)</td>
</tr>
<tr>
<td>n</td>
<td>name of component, lower case</td>
<td>Monday, Tuesday 1)</td>
</tr>
<tr>
<td>Nn</td>
<td>name of component, title case</td>
<td>Monday, Tuesday 1)</td>
</tr>
</tbody>
</table>

**Note:** N, n, and Nn modifiers only support the following components: M, d, D.

The width modifier, if present, is introduced by a comma. It takes the form:

```
, min-width (":" max-width)?
```

The table below illustrates some examples of formatting `xs:dateTime` values with the help of the `format-dateTime` function. The "Value" column specifies the value supplied to the `value` argument. The "Format" column specifies the value of the `format` argument. The "Result" column illustrates what is returned by the function.

<table>
<thead>
<tr>
<th>Value</th>
<th>Format</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2003-11-03T00:00:00</td>
<td>[D]/[M]/[Y]</td>
<td>3/11/2003</td>
</tr>
<tr>
<td>2003-11-03T00:00:00</td>
<td>[Y]-[M,2]-[D,2]</td>
<td>2003-11-03</td>
</tr>
<tr>
<td>2003-11-03T00:00:00</td>
<td>[Y]-[M,2]-[D,2] [H,2]:[m]:[s]</td>
<td>2003-11-03 00:00:00</td>
</tr>
<tr>
<td>2010-06-02T08:02</td>
<td>[Y] [MNNn] [D01] [F,3-3] [d] [H]:[m]:[s].[f]</td>
<td>2010 June 2 Wed 153 8:02:12.054</td>
</tr>
<tr>
<td>2010-06-02T08:02</td>
<td>[Y] [MNNn] [D01] [F,3-3] [d] [H]:[m]:[s].[f] [z]</td>
<td>2010 June 2 Wed 153 8:02:12.054 GMT+02:00</td>
</tr>
<tr>
<td>2010-06-02T08:02</td>
<td>[Y] [MNNn] [D1] [F] [H]:[m]:[s].[f] [Z]</td>
<td>2010 June 2 Wednesday 8:02:12.054 +02:00</td>
</tr>
<tr>
<td>2010-06-02T08:02</td>
<td>[Y] [MNNn] [D] [F,3-3] [H01]:[m]:[s]</td>
<td>2010 June 2 Wed 08:02:12</td>
</tr>
</tbody>
</table>

### 8.9.2.4 `format-number`

Converts a number into a string. The function is available for XSLT 1.0, XSLT 2.0, Java, C#, C++ and Built-in execution engine.
The `format-number` function is used to format a number according to a specified format. It takes several arguments:

- **value**: Mandatory argument. Supplies the number to be formatted.
- **format**: Mandatory argument. Supplies a format string that identifies the way in which the number is to be formatted. This argument is used in the same way as the `format` argument in the `format-dt` function.
- **decimal-point-format**: Optional argument. Supplies the character to be used as the decimal point character. The default value is the full stop (.) character.
- **grouping-separator**: Optional argument. Supplies the character used to separate groups of numbers. The default value is the comma (,) character.

### Note
If the function’s output (i.e. result) is connected to a node of type other than string, the formatting may be lost as the value is cast to the target type. This automatic cast can be disabled by unchecking the **Cast target values to target types** check box in the component settings of the target component.

### Format
```
format := subformat (;subformat)?
subformat := (prefix)? integer (.fraction)? (suffix)?
prefix := any characters except special characters
suffix := any characters except special characters
integer := (#)* (0)* (allowing ',' to appear)
fraction := (0)* (#)* (allowing ',' to appear)
```

The first `subformat` is used for formatting positive numbers, and the second subformat for negative numbers. If only one `subformat` is specified, then the same subformat will be used for negative numbers, but with a minus sign added before the `prefix`.

### Special Character
<table>
<thead>
<tr>
<th>Special Character</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero-digit</td>
<td>0</td>
<td>A digit will always appear at this point in the result</td>
</tr>
<tr>
<td>digit</td>
<td>#</td>
<td>A digit will appear at this point in the result string unless it is a redundant leading or trailing zero</td>
</tr>
<tr>
<td>Special Character</td>
<td>default</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>decimal-point</td>
<td>.</td>
<td>Separates the integer and the fraction part of the number.</td>
</tr>
<tr>
<td>grouping-separator</td>
<td>,</td>
<td>Separates groups of digits.</td>
</tr>
<tr>
<td>percent-sign</td>
<td>%</td>
<td>Multiplies the number by 100 and shows it as a percentage.</td>
</tr>
<tr>
<td>per-mille</td>
<td>‰</td>
<td>Multiplies the number by 1000 and shows it as per-mille.</td>
</tr>
</tbody>
</table>

The characters used for decimal-point-character and grouping-separator are always "." and "," respectively. They can, however, be changed in the formatted output, by mapping constants to these nodes.

The result of the format number function shown above.

- The decimal-point character was changed to a "+".
- The grouping separator was changed to a "-".

```
</Article>
```

### Rounding

The rounding method used for this function is "half up", e.g. the value gets rounded up if the fraction is greater than or equal to 0.5. The value gets rounded down if the fraction is less than 0.5. This method of rounding only applies to generated code and the built-in execution engine.

In XSLT 1.0, the rounding mode is undefined. In XSLT 2.0, the rounding mode is "round-half-to-even".

<table>
<thead>
<tr>
<th>Number</th>
<th>Format String</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234.5</td>
<td>#,##0.00</td>
<td>1,234.50</td>
</tr>
<tr>
<td>123.456</td>
<td>#,##0.00</td>
<td>123.46</td>
</tr>
<tr>
<td>1000000</td>
<td>#,##0.00</td>
<td>1,000,000.00</td>
</tr>
<tr>
<td>Number</td>
<td>Format String</td>
<td>Result</td>
</tr>
<tr>
<td>--------</td>
<td>--------------</td>
<td>--------</td>
</tr>
<tr>
<td>-59</td>
<td>#.,##0.00</td>
<td>-59.00</td>
</tr>
<tr>
<td>1234</td>
<td>###0.0###</td>
<td>1234.0</td>
</tr>
<tr>
<td>1234.5</td>
<td>###0.0###</td>
<td>1234.5</td>
</tr>
<tr>
<td>.00025</td>
<td>###0.0###</td>
<td>0.0003</td>
</tr>
<tr>
<td>.00035</td>
<td>###0.0###</td>
<td>0.0004</td>
</tr>
<tr>
<td>0.25</td>
<td>#00%</td>
<td>25%</td>
</tr>
<tr>
<td>0.736</td>
<td>#00%</td>
<td>74%</td>
</tr>
<tr>
<td>1</td>
<td>#00%</td>
<td>100%</td>
</tr>
<tr>
<td>-42</td>
<td>#00%</td>
<td>-4200%</td>
</tr>
<tr>
<td>-3.12</td>
<td>.00;(#.00)</td>
<td>(3.12)</td>
</tr>
<tr>
<td>-3.12</td>
<td>.00;#.00CR</td>
<td>3.12CR</td>
</tr>
</tbody>
</table>

### 8.9.2.5 format-time

Converts an xs:time input value into a string. The `format` argument is used in the same way as the `format` argument in the `format-dateTime` function.

```
<xs:time>12:15:33</xs:time>
```

```
[|s|-|m|-|h|]
```

Result: 33-15-12
8.9.2.6  number

Converts an input string into a number. Also converts a boolean input to a number.

```
<table>
<thead>
<tr>
<th>arg</th>
<th>result</th>
</tr>
</thead>
</table>
```

8.9.2.7  parse-date


```
<table>
<thead>
<tr>
<th>value</th>
<th>result</th>
</tr>
</thead>
</table>
```

Converts a string into a date, while ignoring the time component. This function uses the parse-dateTime function as a basis, while ignoring the time component. The result is of type xs:date.

For an example of usage, see Example: Mapping Data from an RSS Feed.

8.9.2.8  parse-dateTime


```
<table>
<thead>
<tr>
<th>value</th>
<th>result</th>
</tr>
</thead>
</table>
```

Converts a date/time value expressed as a string into a value of type dateTime. This function takes the following arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The string value to be converted.</td>
</tr>
<tr>
<td>format</td>
<td>Specifies the format mask to apply to value.</td>
</tr>
</tbody>
</table>

For example, in the mapping below, the string value **315 2004 +01:00** specifies the 315th day of year 2004, in the time zone GMT+01:00. This value is converted into its dateTime equivalent, by applying the format mask `[d] [Y] [Z]`. 
A format mask can consist of the following components:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Default Presentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>year (absolute value)</td>
<td>four digits (2010)</td>
</tr>
<tr>
<td>M</td>
<td>month of the year</td>
<td>1-12</td>
</tr>
<tr>
<td>D</td>
<td>day of month</td>
<td>1-31</td>
</tr>
<tr>
<td>d</td>
<td>day of year</td>
<td>1-366</td>
</tr>
<tr>
<td>H</td>
<td>hour (24 hours)</td>
<td>0-23</td>
</tr>
<tr>
<td>h</td>
<td>hour (12 hour)</td>
<td>1-12</td>
</tr>
<tr>
<td>P</td>
<td>A.M. or P.M.</td>
<td>alphabetic (language dependent)</td>
</tr>
<tr>
<td>m</td>
<td>minutes in hour</td>
<td>00-59</td>
</tr>
<tr>
<td>s</td>
<td>seconds in minute</td>
<td>00-59</td>
</tr>
<tr>
<td>f</td>
<td>fractional seconds</td>
<td>numeric, one decimal place</td>
</tr>
<tr>
<td>Z</td>
<td>timezone as a time offset from UTC</td>
<td>+08:00</td>
</tr>
<tr>
<td>z</td>
<td>timezone as a time offset using GMT</td>
<td>GMT+n</td>
</tr>
</tbody>
</table>

Some of the components above take modifiers (for example, they can be used to interpret a date either as a single digit or as two digits):

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>decimal numeric format with no leading zeros: 1, 2, 3,</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>Modifier</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>decimal format, two digits: 01, 02, 03, ...</td>
<td>01, 02, 03</td>
</tr>
<tr>
<td>\textit{N}</td>
<td>name of component, upper case</td>
<td>\textit{FEBRUARY, MARCH}</td>
</tr>
<tr>
<td>\textit{n}</td>
<td>name of component, lower case</td>
<td>\textit{february, march}</td>
</tr>
<tr>
<td>\textit{Nn}</td>
<td>name of component, title case</td>
<td>\textit{February, March}</td>
</tr>
</tbody>
</table>

\textbf{Note:} \textit{N}, \textit{n}, and \textit{Nn} modifiers support only the component \textit{M} (month).

The table below lists a few more examples:

<table>
<thead>
<tr>
<th>Value</th>
<th>Format</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>21-03-2002 16:21:12.492 GMT +02:00</td>
<td>[D]-[M]-[Y] [H]:[m]:[s].[f] [z]</td>
<td>2002-03-21T16:21:12.492+02:00</td>
</tr>
<tr>
<td>315 2004 +01:00</td>
<td>[d] [Y] [Z]</td>
<td>2004-11-10T00:00:00 +01:00</td>
</tr>
<tr>
<td>1.December.10 03:2:39 p.m. +01:00</td>
<td>[D].[MNN].[Y,2-2] [h]:[m]:[s] [P] [Z]</td>
<td>2010-12-01T15:02:39 +01:00</td>
</tr>
<tr>
<td>20110620</td>
<td>[Y,4-4][M,2-2][D,2-2]</td>
<td>2011-06-20T00:00:00</td>
</tr>
</tbody>
</table>

### 8.9.2.9 \texttt{parse-number}


\begin{tabular}{|c|c|c|}
\hline
\texttt{parse-number} & \texttt{value} & \texttt{result} \\
\hline
\texttt{format} & \texttt{format} & \texttt{format} \\
\hline
\texttt{decimal-point-character} & \texttt{decimal-point-character} & \texttt{decimal-point-character} \\
\hline
\texttt{grouping-separator} & \texttt{grouping-separator} & \texttt{grouping-separator} \\
\hline
\end{tabular}

Converts an input string into a decimal number.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The string to be parsed/converted to a number</td>
</tr>
<tr>
<td>format</td>
<td>A format string that identifies the way in which the number is currently formatted (optional). Default is \&quot;#,##0.&quot;</td>
</tr>
<tr>
<td>decimal-point-character</td>
<td>The character to be used as the decimal point character. Default is the \'.' character (optional)</td>
</tr>
</tbody>
</table>
The **grouping-separator** function is used to separate groups of numbers. Default is the "," character (optional).

The **format** string used in parse-number is the same as that used in format-number.

Example in MapForce:

```xml
<expense-item xmlns="http://my-company.com/nam" C:/DOCUME~1/MYDOCU~1/Altova/MapForce2011
  <Misc MiscExpense-Cost="31234.5"/>
</expense-item>
```

**8.9.2.10 parse-time**


Converts a string into a time, while ignoring the date component. This function uses the parse-`date-time` function as a basis, while ignoring the date component. The result is of type xs:time.

**8.9.2.11 string**

Converts an input value into a string. The function can also be used to retrieve the text content of a node.

If the input node is a XML complex type, then all descendents are also output as a single string.

**8.9.3 core | file path functions**

The **file path** functions allow you to directly access and manipulate file path data, i.e. folders, file names, and extensions for further processing in your mappings. They can be used in all
languages supported by MapForce.

8.9.3.1 get-fileext

Returns the extension of the file path including the dot "." character.

```
| get-fileext |
| filepath   |
| extension  |
```

E.g. 'c:\data\Sample.mfd' returns '.mfd'

8.9.3.2 get-folder

Returns the folder name of the file path including the trailing slash, or backslash character.

```
| get-folder |
| filepath   |
| folder     |
```

E.g. 'c:/data/Sample.mfd' returns 'c:/data/

8.9.3.3 main-mfd-filepath

Returns the full path of the mfd file containing the main mapping. An empty string is returned if the mfd is currently unsaved.

```
| main-mfd-filepath |
| filepath          |
```

8.9.3.4 mfd-filepath

If the function is called in the main mapping, it returns the same as main-mfd-filepath function, i.e. the full path of the mfd file containing the main mapping. An empty string is returned if the mfd is currently unsaved.

```
| mfd-filepath |
| filepath     |
```

If called within an user-defined function which is imported by a mfd-file, it returns the full path of the imported mfd file which contains the definition of the user-defined function.
8.9.3.5  remove-fileext

Removes the extension of the file path including the dot-character.

```
remove-fileext
<filepath> result/filepath
```

E.g. 'c:/data/Sample.mfd' returns 'c:/data/Sample'.

8.9.3.6  remove-folder

Removes the directory of the file path including the trailing slash, or backslash character.

```
remove-folder
<filepath> filename
```

E.g. 'c:/data/Sample.mfd' returns 'Sample.mfd'.

8.9.3.7  replace-fileext

Replaces the extension of the file path supplied by the filepath parameter, with the one supplied by the connection to the extension parameter.

```
replace-fileext
<filepath> result/filepath
<extension>
```

E.g. c:/data/Sample.mfd' as the input filepath, and '.mfp' as the extension, returns 'c:/data/Sample.mfp'

8.9.3.8  resolve-filepath

Resolves a relative file path to a relative, or absolute, base folder. The function supports '.' (current directory) and '..' (parent directory).

```
resolve-filepath
<basefolder> result/filepath
<filepath>
```

For an example, see the mapping MergeMultipleFiles_List.mfd available in the ...\MapForceExamples folder.
The core / generator functions library includes functions which generate values.

8.9.4.1 auto-number

The auto-number function generates integers in target nodes of a component, depending on the various parameters you define. The function result is a value starting at start-with and increased by increment. Default values are: start-with=1 and increment=1. Both parameters can be negative.

Make sure that the result connector (of the auto-number function) is directly connected to a target node. The exact order in which functions are called by the generated mapping code is undefined. MapForce may choose to cache calculated results for reuse, or evaluate expressions in any order. It is therefore strongly recommended to take care when using the auto-number function.
**global-id**

This parameter allows you to synchronize the number sequence output of two separate auto-number functions connected to a single target component.

If the two auto-number functions do **not** have the same global-id, then each increments the target items separately. In the example below, each function has a different global-id i.e. a and b.

The output of the mapping is 1,1,2,2. The top function supplies the first 1 and the lower one the second 1.

If both functions have **identical** global-ids, a in this case, then each function "knows" about the current auto-number state (or actual value) of the other, and both numbers are then synchronised/in sequence.

The output of the mapping is therefore 1, 2, 3, 4. The top function supplies the first 1 and the lower one now supplies a 2.

**start-with**

The initial value used to start the auto numbering sequence. Default is 1.
increment
The increment you want auto-number sequence to increase by. Default is 1.

restart on change
Resets the auto-number counter to "start-with", when the content of the connected item changes.

In the example below, start-with and increment are both using the default 1. As soon as the content of Department changes, i.e. the department name changes, the counter is reset and starts at 1 for each new department.

8.9.5 core | logical functions

Logical functions are (generally) used to compare input data with the result being a boolean "true" or "false". They are generally used to test data before passing on a subset to the target component using a filter.

input parameters = a | b, or value1 | value2
output parameter = result

The evaluation result of two input nodes depends on the input values as well as the data types used for the comparison.
For example, the 'less than' comparison of the integer values 4 and 12 yields the boolean value "true", since 4 is less than 12. If the two input strings contain '4' and '12', the lexical analysis results in the output value "false", since '4' is alphabetically greater than the first character '1' of the second operand (12).

If all input data types are of the same type, e.g. all input nodes are numerical types, or strings, then the comparison is done for the common type.

If the input nodes are of differing types (for example, integer and string, or string and date), then the data type used for the comparison is the most general (least restrictive) input data type of the two input types.

Before comparing two values, all input values are converted to a common datatype. Using the previous example; the datatype "string" is less restrictive than "integer". Comparing integer value 4 with the string '12', converts integer value 4 to the string '4', which is then compared with the string '12'.

**Note:** Logical functions cannot be used to test the existence of null values. If you supply a null value as argument to a logical function, it returns a null value. For more information about handling null values, see Nil Values / Nillable.

### 8.9.5.1 equal

Result is true if \( a = b \), else false.

### 8.9.5.2 equal-or-greater

Result is true if \( a \) is equal/greater than \( b \), else false.
8.9.5.3  **equal-or-less**

Result is true if a is equal/less than b, else false.

\[
\begin{array}{c|c|c}
  a & \leq b & \text{result} \\
  \hline
  a & b & \text{result} \\
\end{array}
\]

8.9.5.4  **greater**

Result is true if a is greater than b, else false.

\[
\begin{array}{c|c|c}
  a & > b & \text{result} \\
  \hline
  a & b & \text{result} \\
\end{array}
\]

8.9.5.5  **less**

Result is true if a is less than b, else false.

\[
\begin{array}{c|c|c}
  a & < b & \text{result} \\
  \hline
  a & b & \text{result} \\
\end{array}
\]

8.9.5.6  **logical-and**

If both value1 and value2 of the logical-and function are **true**, then result is true; if different then false.

\[
\begin{array}{c|c|c}
  \text{value1} & \& \text{value2} & \text{result} \\
  \hline
  \text{value1} & \text{value2} & \text{result} \\
\end{array}
\]
8.9.5.7  **logical-not**

Inverts or flips the logical state/result; if input is true, result of logical-not function is false. If input is false then result is true.

The logical-not function shown below, inverts the result of the equal function. The logical-and function now only returns true if boolean values of value1 and value2 are different, i.e. true-false, or false-true.

8.9.5.8  **logical-or**

Requires both input values to be boolean. If either value1 or value2 of the logical-or function are **true**, then the result is true. If both values are false, then result is false.
8.9.5.9  **not-equal**

Result is true if a is not equal to b.

![Diagram of the not-equal function](image)

8.9.6  **core | math functions**

Math functions are used to perform basic mathematical functions on data. Note that they cannot be used to perform computations on durations, or datetimes.

- input parameters = **value1** | **value2**
- output parameter = **result**

Input values are automatically converted to decimal for further processing.

![Diagram of math functions](image)

The example shown above, adds 20% sales tax to each of the articles mapped to the target component.

8.9.6.1  **add**

Result is the decimal value of adding **value1** to **value2**.

![Diagram of the add function](image)
8.9.6.2 ceiling

Result is the smallest integer that is greater than or equal to value, i.e. the next highest integer value of the decimal input value.

E.g. if the result of a division function is 11.2, then applying the ceiling function to it makes the result 12, i.e. the next highest whole number.

8.9.6.3 divide

Result is the decimal value of dividing value1 by value2. The result precision depends on the target language. Use the round-precision function to define the precision of result.

8.9.6.4 floor

Result is the largest integer that is less than or equal to value, i.e. the next lowest integer value of the decimal input value.

E.g. if the result of a division function is 11.2, then applying the floor function to it makes the result 11, i.e. the next lowest whole number.

8.9.6.5 modulus

Result is the remainder of dividing value1 by value2.

In the mapping below, the numbers have been multiplied by 3 and passed on to value1 of the modulus function. Input values are now 3, 6, 9, and 12.
When applying/using modulus 8 as value2, the remainders are 3, 6, 1, and 4.

8.9.6.6 multiply

Result is the decimal value of multiplying value1 by value2.

8.9.6.7 round

Returns the value rounded to the nearest integer. When the value is exactly in between two integers, the "Round Half Towards Positive Infinity" algorithm is used. For example, the value "10.5" gets rounded to "11", and the value "-10.5" gets rounded to "-10".

8.9.6.8 round-precision

Result is the decimal value of the number rounded to the decimal places defined by "decimals".
In the mapping above, the result is 0.429. For the result to appear correctly in an XML file, make sure to map it to an element of xs:decimal type. For a demo, see the following mapping:

\texttt{\<Documents\\Altova\MapForce2019\MapForceExamples\Tutorial\\ReadProtocolBuffers.mfd}. This mapping is discussed in more detail in \textbf{Example: Read Data from Protocol Buffers}.

\subsection{subtract}

Result is the decimal value of subtracting \texttt{value2} from \texttt{value1}.

\subsection{core | node functions}

The node functions allow you to access nodes, or process nodes in a particular way.

\subsubsection{is-xsi-nil}

Returns true (\texttt{<\&OrderID\&>true</\&OrderID\&>) if the element node, of the source component, has the xsi:nil attribute set to "true".

\subsubsection{node-name}

Returns the qualified name (QName) of the connected node. If the node is an XML \texttt{text()} node, an empty QName is returned. This function only works on those nodes that have a name. If XSLT is the target language (which calls \texttt{fn:node-name}), the function returns an empty sequence for nodes which have no names.
Getting a name from database tables/fields is not supported.

XBRL and Excel are not supported.

Getting a name of File input node is not supported.

WebService nodes behave like XML nodes except that:
  - node-name from "part" is not supported.
  - node-name from root node ("Output" or "Input") is not supported.

The MapPerson user-defined function uses `node-name` to return the name of the input node, and place it in the role attribute. The root node of the Employees.xsd, in the user-defined function, has been defined as "Manager".

Manager gets its data from outside the user-defined function, where it can be either: Manager, Programmer, or Support. This is the data that is then passed on to the role attribute in PersonList.
8.9.7.3  set-xsi-nil

Sets the target node to xsi:nil.

```
set-xsi-nil
result
```

8.9.7.4  static-node-annotation

Returns the string with annotation of the connected node. The input must be: (i) a source component node, or (ii) a user-defined function of type "inline" that is directly connected to a parameter, which in turn is directly connected to a node in the calling mapping.

```
static-node-annotation
\node {name}
```

The connection must be direct. It cannot pass through a filter or a non-inlined user-defined function. This is a pseudo-function, which is replaced at generation time with the text acquired from the connected node, and is therefore available for all languages.

8.9.7.5  static-node-name

Returns the string with the name of the connected node. The input must be: (i) a source component node, or (ii) a user-defined function of type "inline" that is directly connected to a parameter, which in turn is directly connected to a node in the calling mapping.

```
static-node-name
\node {name}
```

The connection must be direct. It cannot pass through a filter or a non-inlined user-defined function. This is a pseudo-function, which is replaced at generation time with the text acquired from the connected node, and is therefore available for all languages.

8.9.7.6  substitute-missing-with-xsi-nil

For nodes with simple content, this function substitutes any missing (or null values) of the source component, with the xsi:nil attribute in the target node.

```
substitute-missing-with-xsi-nil
\input {result}
```
8.9.8  core | QName functions

QName functions provide ways to manipulate the Qualified Names (QName) in XML documents.

8.9.8.1  QName

Constructs a QName from a namespace URI and a local part. Use this function to create a QName in a target component. The uri and localname parameters can be supplied by a constant function.

8.9.8.2  local-name-from-QName

Returns the local name part of the QName.

This function is useful when mapping XBRL instance documents containing hypercubes.
What the mapping does is filter those facts where the local name of the content of the explicit member (d-g:Vancouver) is equal to "Vancouver". Note that the content of the member is itself a QName.

```xml
<xbrli:context id="Year2007_Vancouver_BeveragesTotal">
  <xb:entity>
    <xb:identifier scheme="http://www.stockexchange/ticker">ACNE</xb:identifier>
    <xb:segment>
      <xb:explicitMember dimension="d-g:GeographicalBreakdown">d-g:Vancouver</xb:explicitMember>
      <xb:explicitMember dimension="d-b:ByProductsBreakdown">d-b:BeveragesTotal</xb:explicitMember>
    </xb:segment>
  </xb:entity>
</xbrli:context>
```

All the facts that belong to the dimension GeographicalBreakdown are filtered and passed to the target component.
8.9.8.3 namespace-uri-from-QName

Returns the namespace URI part of the QName.

```
namespace-uri-from-QName

<table>
<thead>
<tr>
<th>QName</th>
<th>result</th>
</tr>
</thead>
</table>
```

8.9.9 core | sequence functions

Sequence functions allow processing of input sequences and grouping of their content. The value/content of the key input parameter, mapped to nodes/rows, is used to group the sequence.

- Input parameter **key** is of an arbitrary data type that can be converted to string for group-adjacent and group-by
- Input parameter **bool** is of type Boolean for group-starting-with and group-ending-with
- The output **key** is the key of the current group.

8.9.9.1 distinct-values

Allows you to remove duplicate values from a sequence and map the unique items to the target component.

```
distinct-values

<table>
<thead>
<tr>
<th>values</th>
<th>results</th>
</tr>
</thead>
</table>
```

In the example below, the content of the source component "Title" items, are scanned and each unique title is mapped to the Department / Name item in the target component.
Note that the sequence of the individual Title items in the source component are retained when mapped to the target component.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<department>
  <name>Office Manager</name>
  <name>Account Receivable</name>
  <name>Accounting Manager</name>
  <name>Marketing Manager Europe</name>
  <name>Art Director</name>
  <name>Program Manager</name>
  <name>Software Engineer</name>
  <name>Technical Writer</name>
  <name>IT Manager</name>
  <name>Web Developer</name>
  <name>Support Engineer</name>
  <name>PR &amp; Marketing Manager US</name>
</department>
</Office>
</OrgChart>
```

**8.9.9.2 exists**

Returns true if the node exists, else returns false.
The "HasMarketingExpenses.mfd" file in the `..\MapForceExamples` folder contains the small example shown below.

If an expense-item exists in the source XML, then the "hasExpenses" attribute is set to "true" in the target XML/Schema file.

8.9.9.3 **first-items**

Returns the first "X" items of the input sequence, where X is the number supplied by the "count" parameter. E.g. if the value 3 is mapped to the count parameter and a parent node to the nodes/row parameter, then the first three items will be listed in the output.

8.9.9.4 **generate-sequence**

Creates a sequence of integers using the "from" and "to" parameters as the boundaries.
8.9.9.5  \textit{group-adjacent}

The \textit{group-adjacent} function groups the items connected to the \texttt{nodes/rows} input by the key connected to the \texttt{key} input. Note that this function places items that share the same key into separate groups if they are not adjacent. If multiple consecutive (adjacent) items share the same key, they are placed into the same group.

For example, in the abstract transformation illustrated below, the grouping key is "Department". The left side of the diagram shows the input data while the right side shows the output data after grouping. The following takes place when the transformation runs:

- Initially, the first key, "Administration", creates a new group.
- The next key is different, so a second group is created, "Marketing".
- The third key is also different, so another group is created, "Engineering".
- The fourth key is the same as the third; therefore, this record is placed in the already existing group.
- Finally, the fifth key is different from the fourth, and this creates the last group.

As illustrated below, "Michelle Butler" and "Fred Landis" were grouped together because they have the same key and are adjacent. However, "Vernon Callaby" and "Frank Further" are in separate groups because they are not adjacent, even though they have the same key.

<table>
<thead>
<tr>
<th>Department</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administration</td>
<td>Vernon Callaby</td>
</tr>
<tr>
<td>Marketing</td>
<td>Susi Sanna</td>
</tr>
<tr>
<td>Engineering</td>
<td>Michelle Butler</td>
</tr>
<tr>
<td>Engineering</td>
<td>Fred Landis</td>
</tr>
<tr>
<td>Administration</td>
<td>Frank Further</td>
</tr>
</tbody>
</table>

Arguments

| Name       | Type                      | Description                                                        |
|------------|---------------------------|                                                                   |
| \texttt{nodes/rows} | A sequence of items | A sequence of nodes or rows (for example, the connection may originate from a source XML node with child items, a CSV record, and so on). |
| \texttt{key}    | A variable type           | The key by which to group items.                                   |
Example

Let's assume that your source data is an XML file with the following content (note that, in the code listing below, the namespace and XML declarations were removed for simplicity).

```xml
<company>
  <person department="Administration" name="Vernon Callaby"/>
  <person department="Marketing" name="Susi Sanna"/>
  <person department="Engineering" name="Michelle Butler"/>
  <person department="Engineering" name="Fred Landis"/>
  <person department="Administration" name="Frank Further"/>
</company>
```

The business requirement is to group person records by department, provided they are adjacent. To achieve this, the following mapping invokes the `group-adjacent` function, and supplies `department` as `key`.

The mapping result is as follows:

```xml
<groups>
  <group>
    <record key="Administration" value="Vernon Callaby"/>
  </group>
  <group>
    <record key="Marketing" value="Susi Sanna"/>
  </group>
  <group>
    <record key="Engineering" value="Michelle Butler"/>
    <record key="Engineering" value="Fred Landis"/>
  </group>
  <group>
    <record key="Administration" value="Frank Further"/>
  </group>
</groups>
```

This example, together with other grouping examples, is part of the following mapping file: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\GroupingFunctions.mfd`.

Remember to click the `Preview` button applicable to the function you want to preview, before clicking the `Output` tab.
8.9.9.6  group-by

The group-by function creates groups of records according to some grouping key that you specify. For example, in the abstract transformation illustrated below, the grouping key is "Department". Since there are three unique departments in total, applying the group-by function would create three groups:

<table>
<thead>
<tr>
<th>Department</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administration</td>
<td>Vernon Callaby</td>
</tr>
<tr>
<td>Marketing</td>
<td>Susi Sanna</td>
</tr>
<tr>
<td>Engineering</td>
<td>Michelle Butler</td>
</tr>
<tr>
<td>Engineering</td>
<td>Fred Landis</td>
</tr>
<tr>
<td>Administration</td>
<td>Frank Further</td>
</tr>
</tbody>
</table>

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes/rows</td>
<td>A sequence of items</td>
<td>A sequence of nodes or rows (for example, the connection may originate from a source XML node with child items, a CSV record, and so on).</td>
</tr>
<tr>
<td>key</td>
<td>A variable type</td>
<td>The key by which to group items.</td>
</tr>
</tbody>
</table>

Example

Let's assume that your source data is an XML file with the following content (note that, in the code listing below, the namespace and XML declarations were removed for simplicity).

```xml
<company>
  <person department="Administration" name="Vernon Callaby"/>
  <person department="Marketing" name="Susi Sanna"/>
  <person department="Engineering" name="Michelle Butler"/>
  <person department="Engineering" name="Fred Landis"/>
  <person department="Administration" name="Frank Further"/>
</company>
```

The business requirement is to group person records by department. To achieve this, the following mapping invokes the group-by function, and supplies department as key.
The mapping result is as follows:

```xml
<groups>
    <group>
        <record key="Administration" value="Vernon Callaby"/>
        <record key="Administration" value="Frank Further"/>
    </group>
    <group>
        <record key="Marketing" value="Susi Sanna"/>
    </group>
    <group>
        <record key="Engineering" value="Michelle Butler"/>
        <record key="Engineering" value="Fred Landis"/>
    </group>
</groups>
```

This example, together with other grouping examples, is part of the following mapping file:

```xml
<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\GroupingFunctions.mfd
```

Remember to click the Preview button applicable to the function you want to preview, before clicking the Output tab.

### 8.9.9.7 group-ending-with

The `group-ending-with` function takes a Boolean condition as argument. If the Boolean condition is true, a new group is created, ending with the record that satisfies the condition. In the example below, the condition is that "Key" must be equal to "trailing". This condition is true for the third and fifth records, so two groups are created as a result:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>line</td>
<td>A</td>
</tr>
<tr>
<td>line</td>
<td>B</td>
</tr>
<tr>
<td>trailing</td>
<td>Total 1</td>
</tr>
<tr>
<td>line</td>
<td>C</td>
</tr>
<tr>
<td>trailing</td>
<td>Total 2</td>
</tr>
</tbody>
</table>

```xml
Key   Value
line  A
line  B
trailing Total 1
line  C
trailing Total 2
```
Note: One additional group is created if records exist after the last one that satisfies the condition. For example, if there were more "line" records after the last "trailing" record, these would all be placed into a new group.

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes/rows</td>
<td>A sequence of items</td>
<td>A sequence of nodes or rows (for example, the connection may originate from a source XML node with child items, a CSV record, and so on).</td>
</tr>
<tr>
<td>bool</td>
<td>xs:boolean</td>
<td>Specifies the Boolean condition that starts a new group when true.</td>
</tr>
</tbody>
</table>

Example

Let's assume that your source data is an XML file with the following content (note that, in the code listing below, the namespace and XML declarations were removed for simplicity).

```xml
<records>
  <record key="line" value="A"/>
  <record key="line" value="B"/>
  <record key="trailing" value="Total 1"/>
  <record key="line" value="C"/>
  <record key="trailing" value="Total 2"/>
</records>
```

The business requirement is to create groups for each "trailing" record. Each group must also include any "line" records that precede the "trailing" record. To achieve this, the following mapping invokes the `group-ending-with` function. In the mapping below, whenever the `key` name is equal to "trailing", the argument supplied to `bool` becomes true, and a new group is created.

```xml
<groups>
  <record key="line" value="A"/>
  <record key="line" value="B"/>
  <record key="trailing" value="Total 1"/>
  <record key="line" value="C"/>
  <record key="trailing" value="Total 2"/>
</groups>
```

The mapping result is as follows:
This example, together with other grouping examples, is part of the following mapping file:
<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\GroupingFunctions.mfd.
Remember to click the Preview button applicable to the function you want to preview, before clicking the Output tab.

8.9.9.8 group-into-blocks

The group-into-blocks function creates equal groups that contain exactly N items, where N is the value you supply to the block-size argument. Note that the last group may contain N items or less, depending on the number of items in the source. In the example below, block-size is 2. Since there are five items in total, each group contains exactly two items, except for the last one.

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes/rows</td>
<td>A sequence of items</td>
<td>A sequence of nodes or rows (for example, the connection may originate from a source XML node with child items, a CSV record, and so on).</td>
</tr>
<tr>
<td>block-size</td>
<td>xs:integer</td>
<td>Specifies the size of each group.</td>
</tr>
</tbody>
</table>
Example

Let's assume that your source data is an XML file with the following content (note that, in the code listing below, the namespace and XML declarations were removed for simplicity).

```xml
<company>
  <person department="Administration" name="Vernon Callaby"/>
  <person department="Marketing" name="Susi Sanna"/>
  <person department="Engineering" name="Michelle Butler"/>
  <person department="Engineering" name="Fred Landis"/>
  <person department="Administration" name="Frank Further"/>
</company>
```

The business requirement is to group person records into blocks of two items each. To achieve this, the following mapping invokes the `group-into-blocks` function, and supplies the integer value “2” as `block-size`.

The mapping result is as follows:

```xml
<groups>
  <group>
    <record key="Administration" value="Vernon Callaby"/>
    <record key="Marketing" value="Susi Sanna"/>
  </group>
  <group>
    <record key="Engineering" value="Michelle Butler"/>
    <record key="Engineering" value="Fred Landis"/>
  </group>
  <group>
    <record key="Administration" value="Frank Further"/>
  </group>
</groups>
```

Note that the last group contains only one item, since the total number of items (5) cannot be divided evenly by 2.

This example, together with other grouping examples, is part of the following mapping file: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\GroupingFunctions.mfd`. Remember to click the Preview button applicable to the function you want to preview, before clicking the Output tab.
8.9.9.9 \textit{group-starting-with}

The \textit{group-starting-with} function takes a Boolean condition as argument. If the Boolean condition is true, a new group is created, starting with the record that satisfies the condition. In the example below, the condition is that "Key" must be equal to "heading". This condition is true for the first and fourth records, so two groups are created as a result:

\begin{tabular}{|l|l|}
\hline
\textbf{Key} & \textbf{Value} \\
\hline
heading & Intro \\
line & A \\
line & B \\
heading & Body \\
line & C \\
\hline
\end{tabular}

\begin{tabular}{|l|l|}
\hline
\textbf{Key} & \textbf{Value} \\
\hline
heading & Intro \\
line & A \\
line & B \\
heading & Body \\
line & C \\
\hline
\end{tabular}

\textbf{Note:} One additional group is created if records exist before the first one that satisfies the condition. For example, if there were more "line" records before the first "heading" record, these would all be placed into a new group.

\textbf{Arguments}

\begin{tabular}{|l|l|l|}
\hline
\textbf{Name} & \textbf{Type} & \textbf{Description} \\
\hline
nodes/rows & A sequence of items & A sequence of nodes or rows (for example, the connection may originate from a source XML node with child items, a CSV record, and so on). \\
\hline
bool & xs:boolean & Specifies the Boolean condition that starts a new group when \texttt{true}. \\
\hline
\end{tabular}

\textbf{Example}

Let's assume that your source data is an XML file with the following content (note that, in the code listing below, the namespace and XML declarations were removed for simplicity).

\begin{verbatim}
<records>
  <record key="heading" value="Intro"/>
  <record key="line" value="A"/>
  <record key="line" value="B"/>
  <record key="heading" value="Body"/>
  <record key="line" value="C"/>
</records>
\end{verbatim}
The business requirement is to create groups for each "heading" record. Each group must also include any "line" records that follow the "heading" record. To achieve this, the following mapping invokes the `group-starting-with` function. In the mapping below, whenever the key name is equal to "heading", the argument supplied to `bool` becomes `true`, and a new group is created.

The mapping result is as follows:

```xml
<groups>
  <group>
    <record key="heading" value="Intro"/>
    <record key="line" value="A"/>
    <record key="line" value="B"/>
  </group>
  <group>
    <record key="heading" value="Body"/>
    <record key="line" value="C"/>
  </group>
</groups>
```

This example, together with other grouping examples, is part of the following mapping file: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\GroupingFunctions.mfd`.
Remember to click the Preview button applicable to the function you want to preview, before clicking the Output tab.

### 8.9.9.10 item-at

Returns the `nodes/rows` at the position supplied by the `position` parameter. The first item is at position "1".
8.9.9.11  *items-from-till*

Returns a sequence of nodes/rows using the “from” and “till” parameters as the boundaries. The first item is at position "1".

8.9.9.12  *last-items*

Returns the last "X" nodes/rows of the sequence where X is the number supplied by the "count" parameter. The first item is at position "1".

8.9.9.13  *not-exists*

Returns false if the node exists, else returns true.

The example below shows how you can use the not-exists function to map nodes that do not exist in one of a pair of source files.

What this mapping does:

- Compare the nodes of two source XML files
- Filter out the nodes of the first source XML file, that do not exist in the second source XML file
- Map only the missing nodes, and their content, to the target file.
The two XML instance files are shown below, the differences between them are:

- **a.xml** (left) contains the node `<b kind="3">`, which is missing from **b.xml**.
- **b.xml** (right) contains the node `<b kind="4">` which is missing from **a.xml**.

- The equal function compares the kind attribute of both XML files and passes the result to the filter.
- A not-exists function is placed after the initial filter, to select the missing nodes of each of the source files.
- The second filter is used to pass on the missing node and other data only from the **a.xml** file to the target.

The mapping result is that the node missing from **b.xml**, `<b kind="3">`, is passed on to the target component.
8.9.9.14  position

Returns the position of a node inside its containing sequence.

The position function allows you to determine the position of a specific node in a sequence, or use a specific position to filter out items based on that position.

The context item is defined by the item connected to the "node" parameter of the position function, Person, in the example below.

The simple mapping below adds a position number to each Person of each Department.

The position number is reset for each Department in the Office.
Using the position function to filter out specific nodes

Using the position function in conjunction with a filter allows you to map only those specific nodes that have a certain position in the source component.

The filter "node/row" parameter and the position "node" must be connected to the same item of the source component, to filter out a specific position of that sequence.

What this mapping does is to output:

- The second Person in each Department
- of each Office in Altova.
Finding the position of items in a filtered sequence:

As the filter component is not a sequence function, it cannot be used directly in conjunction with the position function to find the position of filtered items. To do this you have to use the "Variable" component.

The results of a Variable component are always sequences, i.e. a delimited list of values, which can also be used to create sequences.

- The variable component is used to collect the filtered contacts where the last name starts with a letter higher than "M".
- The contacts are then passed on (from the variable) to the target component
- The position function then numbers these contacts sequentially
8.9.9.15  replicate-item

Repeat every item in the input sequence the number of times specified in the count argument. If you connect a single item to the node/row argument, the function returns N items, where N is the value of the count argument. If you connect a sequence of items to the node/row argument, the function repeats each individual item in the sequence count times, processing one item at a time. For example, if count is 2, then the sequence (1,2,3) produces (1,1,2,2,3,3).

Note that you can supply a different count value for each item. For example, let's assume that you have a source XML file with the following structure:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<SourceList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="source.xsd">
  <person>
    <name>Michelle</name>
    <count>2</count>
  </person>
  <person>
    <name>Ted</name>
    <count>4</count>
  </person>
  <person>
    <name>Ann</name>
    <count>3</count>
  </person>
</SourceList>
```

With the help of the replicate-item function, you can repeat each person name a different number of times in a target component. To achieve this, connect the count node of each person to the count input of the replicate-item function:
The output is as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<TargetLists xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="target.xsd">
  <TargetList>
    <TargetString>Michelle</TargetString>
    <TargetString>Michelle</TargetString>
  </TargetList>
  <TargetList>
    <TargetString>Ted</TargetString>
    <TargetString>Ted</TargetString>
    <TargetString>Ted</TargetString>
    <TargetString>Ted</TargetString>
  </TargetList>
  <TargetList>
    <TargetString>Ann</TargetString>
    <TargetString>Ann</TargetString>
    <TargetString>Ann</TargetString>
  </TargetList>
</TargetLists>
```

8.9.9.16  

`replicate-sequence`  

Repeats all items in the input sequence the number of times specified in the `count` argument. For example, if `count` is 2, then the sequence `(1, 2, 3)` produces `(1, 2, 3, 1, 2, 3)`. 

```
<replicate-sequence>
  <nodes>rows</nodes>
  <count>results</count>
</replicate-sequence>
```
8.9.9.17  set-empty

Returns an empty sequence. For example, you can use this function to cancel default values of an XBRL document that were defined higher up the XBRL component/taxonomy.

```
set-empty
   empty
```

8.9.9.18  skip-first-items

Skips the first "X" items/nodes of the input sequence, where X is the number supplied by the "count" parameter, and returns the rest of the sequence.

```
skip-first-items
   nodes/rows
   count
   results
```

8.9.9.19  substitute-missing

This function is a convenient combination of exists and a suitable if-else condition. Used to map the current field content if the node exists in the XML source file, otherwise use the item mapped to the "replace-with" parameter.

```
substitute-missing
   node
   replace-with
   result
```

8.9.10  core | string functions

The string functions allow you to use the most common string functions to manipulate many types of source data to: extract portions, test for substrings, or retrieve information on strings.

8.9.10.1  char-from-code

Result is the character representation of the decimal Unicode value of value.

```
char-from-code
   value
   result
```
8.9.10.2  code-from-char

Result is the decimal Unicode value of the first character of value.

```
code-from-char
value  result
```

8.9.10.3  concat

Concatenates (appends) two or more values into a single result string. All input values are automatically converted to type string.

```
cat
value1  result
value2
```

By default, this function has only two parameters, but you can add more. Click Add parameter (.addButton) or Delete parameter (deleteButton) to add or remove parameters, see also Add or Delete Function Arguments.

Example

In the mapping illustrated below, the concat function joins the first name, the constant " ", and the last name. The returning value is then written to the FullName target item. The mapping of this function is available at the following path: <Documents>\Altova\MapForce2019\MapForceExamples\HasMarketingExpenses.mfd.
8.9.10.4  contains

Result is true if data supplied to the value parameter contains the string supplied by the substring parameter.

```
<contains>
  <value>
  </value>
  <substring>
  </substring>
</contains>
```

8.9.10.5  normalize-space

Result is the normalized input string, i.e. leading and trailing spaces are removed, then each sequence of multiple consecutive whitespace characters are replaced by a single whitespace character. The Unicode character for "space" is (U+0020).

```
<normalize-space>
  <string>
  </string>
</normalize-space>
```
8.9.10.6  starts-with

Result is true if the input string "string" starts with substr, else false.

8.9.10.7  string-length

Result is the number of characters supplied by the string parameter.

8.9.10.8  substring

Result is the substring (string fragment) of the "string" parameter where "start" defines the position of the start character, and "length" the length of the substring.

If the length parameter is not specified, the result is a fragment starting at the start position and ending at the end position of the string. Indices start counting at 1.

E.g. substring("56789",2,3) results in 678.

8.9.10.9  substring-after

Result is the remainder of the "string" parameter, where the first occurrence of the substr parameter defines the start characters; the remainder of the string is the result of the function. An empty string is the result, if substr does not occur in string.

E.g. substring-after("2009/01/04","/") results in the substring 01/04. substr in this case is the first "/" character.
For an example of usage, see [Example: Mapping Data from an RSS Feed](#).

### 8.9.10.10 substring-before

Result is the string fragment of the "string" parameter, up to the first occurrence of the substr characters. An empty string is the result, if substr does not occur in string.

E.g. substring-before ("2009/01/04","/") results in the substring 2009. substr in this case is the first "/" character.

### 8.9.10.11 tokenize

Result is the input string split into a sequence of chunks/sections defined by the delimiter parameter. The result can then be passed on for further processing.

E.g. Input string is A,B,C and delimiter is "," - then result is A B C.

**Example**

The `tokenizeString1.mfd` file available in the `...\MapForceExamples` folder shows how the tokenize function is used.

The XML source file is shown below. The Tool element has two attributes: Name and Code, with the Tool element data consisting of comma delimited text.
What the mapping does:

- The tokenize function receives data from the Tool element/item and uses the comma "," delimiter to split that data into separate chunks. I.e. the first chunk "XML editor".
- As the result parameter is mapped to the Rows item in the target component, one row is generated for each chunk.
- The result parameter is also mapped to the left-trim function which removes the leading white space of each chunk.
- The result of the left-trim parameter (each chunk) is mapped to the Feature item of the target component.
- The target component output file has been defined as a CSV file (AltovaToolFeatures.csv) with the field delimiter being a semicolon (double click component to see settings).

Result of the mapping:

- For each Tool element of the source file
- The (Tool) Name is mapped to the Tool item in the target component
- Each chunk of the tokenized Tool content is appended to the (Tool Name) Feature item
- E.g. The first tool, XMLSpy, gets the first Feature chunk "XML editor"
- This is repeated for all chunks of the current Tool and then for all Tools.
- Clicking the Output tab delivers the result shown below.
8.9.10.12 **tokenize-by-length**

Result is the input string split into a sequence of chunks/sections defined by the length parameter. The result can then be passed on for further processing.

E.g. Input string is ABCDEF and length is "2" - then result is AB CD EF.

**Example**

The **tokenizeString2.mfd** file available in the ...\MapForceExamples folder shows how the **tokenize-by-length** function is used.

The XML source file is shown below, and is the same as the one used in the previous example.
The **MissionKit** element also has two attributes: **Edition** and **ToolCodes**, but no MissionKit element content.

The aim of the mapping is to generate a list showing which Altova tools are part of the respective MissionKit editions.

How the mapping works:

- The SelectMissionKit Input component receives its default input from a constant component, in this case "Enterprise XML Developers".
- The **equal** function compares the input value with the "Edition" value and passes on the result to the **bool** parameter of the ToolCodes filter.
- The **node/row** input of the ToolCodes filter is supplied by the **ToolCodes** item of the source file. The value for the Enterprise XML Developers edition is: XSMFSVDDSASW.
- The XSMFSVDDSASW value is passed to the **on-true** parameter, and further to the **input** parameter of the **tokenize-by-length** function.

What the **tokenize-by-length** function does:

- The **ToolCodes** input value XSMFSVDDSASW, is split into multiple chunks of two characters each, defined by **length** parameter, which is 2, thus giving 6 chunks.
- Each chunk (placed in the **b** parameter) of the **equal** function, is compared to the 2 character **Code** value of the source file (of which there are 9 entries/items in total).
- The result of the comparison (true/false) is passed on to the **bool** parameter of the filter.
- Note that all chunks, of the tokenize-by-length function, are passed on to the **node/row** parameter of the **exists** function.
- The **exists** functions now checks for existing/non-existing nodes passed on to it by the **on-true** parameter of the filter component. Existing nodes are those where there is a match between the **ToolCodes** chunk and the **Code** value. Non-existing nodes are where there was no **ToolCodes** chunk to match a **Code** value.
- The **bool** results of the **exists** function are passed on to the **if-else** function which passes on a **Y** to the target if the node exists, or a **N**, if the node does not exist.

Result of the mapping:
Result is the input string split into a sequence of strings, where the supplied regular expression pattern match defines the separator. The separator strings are not output by the result parameter. Optional flags may also be used.

In the example shown above, input string is a succession of characters separated by spaces and/or commas, i.e.

```
a, b c,d
```

The regex pattern defines a character class ["space""comma"] - of which one and only one character will be matched in a character class, i.e. either space or comma.

The + quantifier specifies "one or more" occurrences of the character class/string.

The result string is:
Notes:
- There are slight differences in regular expression syntax between the various languages.
- `_tokenize-regexp` in C++ is only available in Visual Studio 2008 SP1 and later.

For more information, see [Regular expressions](#).

8.9.10.14  **translate**

The characters of `string1` (search string) are replaced by the characters at the same position in `string2` (replace string), in the input string "value".

When there are no corresponding characters in string2, the character is removed.

*E.g.*

input string is 123145  
(search) `string1` is 15  
(replace) `string2` is xy

So:

- each 1 is replaced by x in the input string value
- each 5 is replaced by y in the input string value

Result string is **x23x4y**

If string2 is empty (fewer characters than string1) then the character is removed.
E.g.2
input string aabaacbcba
  string1 is "a"
  string2 is "" (empty string)
result string is "bcbc"

E.g.3
input string aabaacbcba
  string1 is "ac"
  string2 is "ca"
result string is "ccbccabac"

8.9.11  db

The db library contains functions that allow you to define the mapping results when encountering null fields in databases.

8.9.11.1  is-not-null

Returns false if the field is null, otherwise returns true.

8.9.11.2  is-null

Returns true if the field is null, otherwise returns false.

8.9.11.3  set-null

Used to set a database column to null. This function will also overwrite a default value with null. If connected to something else i.e. not a database field, it will behave like an empty sequence.

Please note:
Connecting set-null to a different function will usually result in the other function not being called at all. Connecting set-null to a sequence function, e.g. count, will call the function with an empty sequence.

Connecting to special functions, Filters and IF-Else conditions works as expected, fields are set to null. For filters this means the "node/row" input.

Using set-null as an input for a simpleType element will not create that element in the target component.

8.9.11.4 substitute-null

Used to map the current field content if it exists, otherwise use the item mapped to the replace-with parameter.

The image below shows an example of the substitute-null function in use, and is available as “DB-ApplicationList” in the ...

The first function checks if a Category entry exists in the Applications table. As one does not exist for the Notepad application, "Misc" is mapped to the Category item of the Text file.

The second function checks if a Description entry exist, and maps the string "No description" if one does not exist, which is also the case with the Notepad application.
8.9.12 edifact

The edifact library contains functions that convert EDI date, time, periods into xs:date formats.

8.9.12.1 auto-format

Result is the Date / Time / Datetime value extracted from the coded source (F2380) given format code (F2379).

8.9.12.2 to-date

Result is the Date value extracted from the coded source (F2380) given format code (F2379).

8.9.12.3 to-datetime

Result is the Datetime value extracted from the coded source (F2380) given format code (F2379).

E.g.
Given the EDI file shown in the screenshot below, the Date/Time/Period value is shown in the DTM field.
The value is converted to an xs:date format using the to-datetime function.

```
<?xml version="1.0" encoding="UTF-8"?>
<Order xsi:noNamespaceSchemaLocation="C:\DOCUME=1\...
  <Header>
    <Number>ADC123456XYZ</Number>
    <Received>2004-04-30T17:42:00-09:00</Received>
  </Header>
  <Customer>
    <Number>123</Number>
  </Customer>
```

The result of the conversion is shown below. This sample is available in the `MapForceExamples` as `EDI-Order.mfd`. 
8.9.12.4  to-duration

Result is the Duration value extracted from the coded source (F2380) given format code (F2379).

\[
\text{F2380} \quad \text{result} \quad \text{F2379}
\]

8.9.12.5  to-time

Result is the Time value extracted from the coded source (F2380) given format code (F2379).

\[
\text{F2380} \quad \text{result} \quad \text{F2379}
\]

8.9.13  lang | QName functions

The lang library contains functions that are available when selecting either Java, C#, or C++ languages.

8.9.13.1  QName-as-string

Result is the unique string representation of the QName.

\[
\text{QName} \quad \text{result}
\]

8.9.13.2  string-as-QName

Converts the string representation of a QName back to a QName.

\[
\text{string} \quad \text{result}
\]
8.9.14  lang | datetime functions

The lang library contains functions that are available when selecting either Java, C#, or C++ languages.

8.9.14.1  age

Returns the age of the person in full years. The now argument is optional and the default is the current system date. The result is then the full amount of years between the birthdate and now. If a value is mapped to the now argument, the result is the difference between the two dates in full years.

```
<age>
  <birthdate>result</birthdate>
  <now>result</now>
</age>
```

8.9.14.2  convert-to-utc

Converts the local "time" input parameter into Coordinated Universal Time, or GMT/Zulu time. (The function takes the timezone component, e.g. +5:00, into account).

```
<convert-to-utc>
  <time>time</time>
</convert-to-utc>
```

E.g. Instance document datetime:
```
departuredatetime="2001-12-17T09:30:02+05:00"
```

Result:
```
departuredatetime="2001-12-17T04:30:02"
```

Please note:
If the source dateTime is in the form `departuredatetime="2001-12-17T09:30:02Z"` then no
conversion will take place because the trailing "Z" defines this time to be Zulu time, i.e. UTC. The result will be `departuredatetime="2001-12-17T09:30:02"`.

### 8.9.14.3 `date-from-datetime`

Result is the date part of a datetime input argument. The time part of the dateTime, starting with T in the instance document, is set to zero. Note that the timezone increment is not changed.

```
<date-from-datetime datetime="departuredatetime" result="date />
```

E.g. Instance document datetime:
```
departuredatetime="2001-12-17T09:30:02+05:00"
```

Result:
```
departuredatetime="2001-12-17T00:00:00+05:00"
```

### 8.9.14.4 `datetime-add`

Result is the datetime obtained by adding a duration (second argument) to a datetime (first argument).

```
<datetime-add value1="departuredatetime" value2="duration" result="datetime />
```

Durations must be entered in the form: P1Y2M3DT04H05M. Periods can be made negative by adding the minus character before the P designator, e.g. -P1D.

P is the period designator, and is mandatory; the rest of above period is therefore: 1 Year, 2 Months, 3 Days T(time designator), 04 Hours, 05 Minutes.

The example shown below, adds 10 days to the departuredatetime, i.e. P10D.
E.g. Instance document datetime:
departuredatetime="2001-12-17T09:30:02+05:00"

Result:
departuredatetime="2001-12-27T09:30:02+05:00"

To extract yesterdays date from dateTime input, use the "now" function to input the current date/time including timezone. A period can be made negative by using the minus character before the P designator, e.g. -P1D (minus 1 day).

E.g. datetime now is 28th Feb 2012, 17:19:54.748(millisecond)+01timezone.
now="2012-02-28T17:19:54.748+01:00"

Result:
departuredatetime="2012-02-27T17:19:54.748+01:00"
i.e. 27th Feb 2012, 17:19:54.748(millisecond)+01timezone

### 8.9.14.5 `datetime-diff`

Result is the duration obtained by subtracting datetime2 (second argument) from datetime1 (first argument). The result can be mapped to a string, or duration, datatype.
Note that the arrivaldatetime has been connected to datetime1 and departuredatetime to datetime2.

E.g. We want to find the difference, as a duration, between the departure and arrival times.

```
datetime1 arrivaldatetime="2001-12-17T19:30:02+05:00"
datetime2 departuredatetime="2001-12-17T09:30:02+05:00"
```

Result: the difference between the two is 10 hours:
```
result= PT10H
```

### 8.9.14.6 `datetime-from-date-and-time`

Result is a datetime built from a datevalue (first argument) and a timevalue (second argument). The first argument must be of type xs:date and the second xs:time. The result can be mapped to a string or dateTime datatype.
E.g.

date="2012-06-29"
time="11:59:55"

Result:

datetime="2012-06-29T11:59:55"

8.9.14.7  dat time from parts

Result is a datetime built from any combination of the following parts as arguments: year, month, day, hour, minute, second, millisecond, and timezone. This function automatically normalizes the supplied parameters e.g. 32nd of January will automatically be changed to 1st February.

All of the arguments are of type xs:int except for millisecond, which is of type xs:decimal. The datetime result parameter is of type xs:dateTime.

The screenshot shown below is of the user-defined function "convertIDOCDate" available in the IDoc_Order.mfd mapping in the ...\MapForceExamples folder. It assembles the datetime from an input string using left-trim and substring functions.

Note that year, month, and day are mandatory parameters, the rest are optional.
The date and time fields are supplied by the IDOC instance file:

IDOC:Date  19990621
IDOC:Time  0930
Result      1999-06-21T09:30:00

8.9.14.8  *day-from-datetime*

Result is the day from the datetime argument.
E.g.
datetime="2001-12-17T10:30:03+01:00"

Result: 17

8.9.14.9 **day-from-duration**

Result is the day from the duration argument.

e.g.
duration="P1Y2M3DT10H30M"

Result: 3
8.9.14.10  **duration-add**

Result is the duration obtained by adding two durations.

\[
\text{duration-add} \quad \text{duration1} + \text{duration2} \rightarrow \text{result}
\]

E.g.
\[
duration1 = "P0Y0M3DT03H0M" \text{ (3 days 3 hours)}
duration2 = "P0Y0M3DT01H0M" \text{ (3 days 1 hour)}
\]
Result: \( P6DT4H \) (6 days 4 hours)

8.9.14.11  **duration-from-parts**

Result is a duration calculated by combining the following parts supplied as arguments: year, month, day, hour, minute, second, millisecond, negative.

\[
\text{duration-from-parts} \quad \text{year} + \text{month} + \text{day} + \text{hour} + \text{minute} + \text{second} + \text{millisecond} + \text{negative} \rightarrow \text{duration}
\]

Durations are in the form \( P1Y2M3DT04H05M06.07S \) i.e. \( P(eri)od \) 1 Year, 2 Months, 3 Days, \( T(ime) \) designator, 04 Hours, 05 Minutes, 06.07 seconds.milliseconds.

All of the arguments are of type \( \text{xs:int} \) except for millisecond, which is of type \( \text{xs:decimal} \), and negative, which is of type \( \text{xs:boolean} \) (i.e. 1 for true, 0 for false). The duration parameter is of type \( \text{xs:duration} \).
Parts: 1971 year, 11 month, 19 day, 11 hour, 05 minutes, 15.06 seconds, negative period "false".

Result:

duration="P1971Y11M19DT11H5M15.00006S"

8.9.14.12  duration-subtract

Result is the duration obtained by subtracting duration2 from duration1.

Durations must be entered in the form: P1Y2M3DT04H05M. Periods can be made negative by using the minus character before the P designator, e.g. -P1D.

P is the period designator, and is mandatory; the rest of period is therefore: 1 Year, 2 Months, 3 Days T(time designator), 04 Hours, 05 Minutes.

The example shown below, subtracts 1 hour from flighttime, i.e. PT1H.
E.g.  
duration1="P0Y0M0DT05H07M"  
duration2="PT1H"  

Result: PT4H7M

**8.9.14.13  hour-from-datetime**

Result is the hour part of the datetime argument.

E.g.  
datetime="2001-12-17T09:30:02+05:00"  
hour= 9

**8.9.14.14  hour-from-duration**

Result is the hour component of the duration argument.

E.g.  
duration="P0Y0M0DT05H07M"  
hour= 5
8.9.14.15  leapyear

Result is true or false depending on whether the year of the supplied dateTime is in a leap year.

```
leapyear
  value result
```

E.g.
```
arrivaldatetime="2001-12-17T19:30:02+05:00"
result="false"
```

8.9.14.16  millisecond-from-datetime

Result is the millisecond part of the dateTime argument.

```
millisecond-from-datetime
  dateTime millisecond
```

E.g.
```
datetime="2001-12-17T09:30:02.544+05:00"
millisecond= 544
```

8.9.14.17  millisecond-from-duration

Result is the millisecond component of the duration argument.

```
millisecond-from-duration
  duration millisecond
```

E.g.
8.9.14.18  **minute-from-datetime**

Result is the minute part of the datetime argument.

\[
\text{minute} = \text{minute-from-datetime (datetime)}
\]

**E.g.**

\[
\text{datetime} = "2001-12-17T09:30:02.544+05:00"
\]

\[
\text{minute} = 30
\]

8.9.14.19  **minute-from-duration**

Result is the minute component of the duration argument.

\[
\text{minute} = \text{minute-from-duration (duration)}
\]

**E.g.**

\[
\text{duration} = "P0Y0M0DT05H07M02.227S"
\]

\[
\text{minute} = 7
\]

8.9.14.20  **month-from-datetime**

Result is the month part of the dateTime argument.

\[
\text{month} = \text{month-from-datetime (datetime)}
\]

**E.g.**

\[
\text{datetime} = "2001-12-17T09:30:02.544+05:00"
\]

\[
\text{month} = 12
\]

8.9.14.21  **month-from-duration**

Result is the month component of the duration argument.

\[
\text{month} = \text{month-from-duration (duration)}
\]
E.g.
duration="P0Y0M0DT05H07M02.227S"
month= 4

8.9.14.22 now

Result is the current dateTime (including timezone).

\[
\text{now}
\]

E.g.
result= 2012-03-06T14:44:57.567+01:00

For an example on how to extract yesterday’s date, see the [core | lang | datetime-add](#) function.

8.9.14.23 remove-timezone

Removes the timezone component, e.g. +5:00, from the time input parameter.

\[
\text{remove-timezone}
\]

E.g.
departuredatetime="2001-12-17T09:30:02+05:00"
time: 2001-12-17T09:30:02

8.9.14.24 second-from-datetime

Result is the seconds part of the dateTime argument.

\[
\text{second-from-datetime}
\]

E.g.
datetime="2001-12-17T09:30:02.544+05:00"
second= 2
8.9.14.25  second-from-duration

Result is the seconds component of the duration argument.

```
£ second-from-duration
<duration> second>
```

E.g.
```
duration="P0Y04M0DT05H07M02.227S"
second= 2
```

8.9.14.26  time-from-datetime

Result is the time part of the dateTime argument.

```
£ time-from-datetime
<datetime> time>
```

E.g.
```
datetime="2001-12-17T09:30:02.544+05:00"
time= 09:31:02+05:00
```

8.9.14.27  timezone

Returns the timezone (i.e. +05:00 here) relative to UTC of the dateTime value. NB timezone unit is minutes.

```
£ timezone
<datetime> timezone>
```

E.g.
```
datetime="2001-12-17T09:30:02.544+05:00"
timezone= 300
```

8.9.14.28  weekday

Returns the weekday of the dateTime value, starting with Monday=1 to Sunday=7.

```
£ weekday
<datetime> weekday>
```

E.g.
```
datetime="2001-12-17T09:30:02.544+05:00"
weekday= 1
```
8.9.14.29  weeknumber

Returns the week number within the year specified by the dateTime value.

E.g.

datetime="2001-12-17T09:30:02.544+05:00"
weeknumber= 51

8.9.14.30  year-from-datetime

Result is the year part of the dateTime argument.

E.g.

datetime="2001-12-17T09:30:02.544+05:00"
year= 2001

8.9.14.31  year-from-duration

Result is the year component of the duration argument.

E.g.

duration="P01Y04M0DT05H07M02.227S"
year= 1

8.9.15  lang | generator functions

The lang library contains functions that are available when selecting either Java, C#, or C++ languages. The generator functions generate values for database fields, which do not have any input data from the Schema, database or EDI source component.
8.9.15.1 create-guid

Result is a globally-unique identifier (as a hex-encoded string) for the specific field.

8.9.16 lang | logical functions

The lang library contains functions that are available when selecting either Java, C#, or C++ languages.

8.9.16.1 logical-xor

Result is true if value1 is different than value2, otherwise false.

8.9.16.2 negative

Result is true if value is negative, i.e. less than zero, otherwise false.

8.9.16.3 numeric

Result is true if value is a number, otherwise false. The input will usually be a string.
8.9.16.4  positive

Result is true if value is positive, i.e. equal to or greater than zero, otherwise false.

<table>
<thead>
<tr>
<th>value</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8.9.17  lang | math functions

The lang library contains functions that are available when selecting either Java, C#, or C++ languages.

8.9.17.1  abs

Result is the absolute value of the input value.

<table>
<thead>
<tr>
<th>value</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8.9.17.2  acos

Result is the arc cosine of value.

<table>
<thead>
<tr>
<th>value</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8.9.17.3  asin

Result is the arc sine of value.

<table>
<thead>
<tr>
<th>value</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
8.9.17.4  atan

Result is the arc tangent of value.

\[
\text{atan}(\text{value}, \text{result})
\]

8.9.17.5  cos

Result is the cosine of value.

\[
\cos(\text{value}, \text{result})
\]

8.9.17.6  degrees

Result is the conversion of value in radians into degrees.

\[
\text{degrees}(\text{value}, \text{result})
\]

8.9.17.7  divide-integer

Result is the integer result of dividing value1 by value2. E.g. 15 divide-integer 2, integer result is 7.

\[
\text{divide-integer}(\text{value1}, \text{value2}, \text{result})
\]

8.9.17.8  exp

Result is \(e\) (base natural logarithm) raised to the value\(^{th}\) power.

\[
\text{exp}(\text{value}, \text{result})
\]
8.9.17.9  log

Result is the natural logarithm of value.

8.9.17.10  log10

Result is logarithm (base 10) of value.

8.9.17.11  max

Result is the numerically larger value of value1 compared to value2.

8.9.17.12  min

Result is the numerically smaller value of value1 compared to value2.

8.9.17.13  pi

Result is the value of pi.
8.9.17.14  pow

Result is the value of a raised to the power b\textsuperscript{th} power.

\[
\begin{array}{c}
\text{pow} \\
\text{a} \\
\text{b} \\
\text{result}
\end{array}
\]

8.9.17.15  radians

Result is the conversion of value in degrees to radians.

\[
\begin{array}{c}
radians \\
\text{value} \\
\text{result}
\end{array}
\]

8.9.17.16  random

Result is a pseudorandom value between 0.0 and 1.0

\[
\begin{array}{c}
\text{random} \\
\text{result}
\end{array}
\]

8.9.17.17  sin

Result is the sine of value.

\[
\begin{array}{c}
sin \\
\text{value} \\
\text{result}
\end{array}
\]

8.9.17.18  sqrt

Result is the square root of value.

\[
\begin{array}{c}
sqrt \\
\text{value} \\
\text{result}
\end{array}
\]
8.9.17.19  tan

Result is the tangent of value.

8.9.17.20  unary-minus

Result is the negation of the signed input value. E.g. +3 result is -3, while -3 result is 3.

8.9.18  lang | string functions

The lang library contains functions that are available when selecting either Java, C#, or C++ languages.

8.9.18.1  capitalize

Result is the input string value, where the first letter of each word is capitalized (initial caps).

8.9.18.2  charset-decode

The charset-decode function takes as input binary data encoded as Base64 text. It decodes data according to the specified character set (for example, "utf-8") and returns the resulting string value. If you need to encode binary data as Base64 text, use the charset-encode function.
### Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary-data</td>
<td>xs:base64Binary</td>
<td>The binary data as Base64 text.</td>
</tr>
<tr>
<td>encoding</td>
<td>xs:string</td>
<td>The character set used for encoding (for example, &quot;utf-8&quot;).</td>
</tr>
<tr>
<td>error-abort</td>
<td>xs:boolean</td>
<td>Optional argument that specifies how processing should continue when errors are encountered. Valid values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• true - End processing with an exception on the invalid character.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• false - Continue processing, and replace invalid characters with the replacement character 🛠.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default value is true.</td>
</tr>
</tbody>
</table>

### Example

Let's suppose that you would like to decode binary data originating from the following source XML file. Notice that the `message` element contains binary data encoded as Base64 text.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<message xsi:noNamespaceSchemaLocation="message.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">TG9yZW0gaXBzdW0=</message>
```

The data type of the `message` element is `xs:base64Binary`, as illustrated by the schema:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="message" type="xs:base64Binary"/>
</xs:schema>
```

A mapping that decodes the message above looks as follows:
The mapping in this example outputs the text "Lorem ipsum".

A mapping can also process text or XML files encoded as Base64 data, with the help of a MapForce serialization component. For example, the mapping illustrated below has an input parameter which expects Base64 text data. Assuming that the Base64 data was created from an XML file as shown in the charset-encode example, you can recreate the original XML file as shown in the mapping below:

In this mapping, the error-abort argument gets a false value, which was produced with the help of the boolean built-in function. This ensures that processing will continue even if invalid characters are encountered. The string result of the function is then passed to an XML parsing component which converts it to an XML file. Note that, in order for XML parsing to be possible, you must have the XSD schema file. For more information, see Parsing and Serializing Strings.

8.9.18.3 charset-encode

The charset-encode function takes as input string data and encodes it as Base64 text. Data is encoded in the specified character set (for example, "utf-8") and returned as xs:base64Binary type. If you need to decode binary data previously encoded as Base64 text, use the charset-decode function.
Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string-data</td>
<td>xs:string</td>
<td>The string data to be encoded.</td>
</tr>
<tr>
<td>encoding</td>
<td>xs:string</td>
<td>The character set used for encoding (for example, &quot;utf-8&quot;).</td>
</tr>
<tr>
<td>substitute</td>
<td>xs:string</td>
<td>Optional argument that specifies a replacement character when invalid characters are encountered. This argument is applicable if you use a non-Unicode encoding. For Unicode encodings, the replacement character is ².</td>
</tr>
</tbody>
</table>

Example

Let's suppose that you would like to encode the text "Lorem ipsum" as Base64 data, using the UTF-8 character set, and write it to a target XML file. The target XML file has a message element of xs:base64Binary type, as illustrated by the schema:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="message" type="xs:base64Binary"/>
</xs:schema>
```

A mapping that performs the Base64 encoding looks as follows:

This mapping produces XML output like the one below (the schema references and XML declaration were skipped):

```xml
<message>TG9yZW0gaXBzdW0=</message>
```

You can also encode text or XML files as Base64, with the help of a MapForce serialization component. For example, the mapping illustrated below serializes a source XML file to a string. The resulting string is then supplied as argument to the charset-encode function. Finally, the
function result is returned as mapping output, with the help of a simple output component, see Returning String Values from a Mapping. For more information about serialization, see Parsing and Serializing Strings.

8.9.18.4 count-substring

Result is the number of times that substr occurs in string.

8.9.18.5 empty

Result is true if the input string value is empty, otherwise false.

8.9.18.6 find-substring

Returns the position of the first occurrence of substr. within string, starting at position startindex. The first character has position 1. If the substring could not be found, then the result is 0.
8.9.18.7  **format-guid-string**

Result is a correctly formatted GUID string `formatted_guid`, using `unformatted_guid` as the input string, for use in database fields. See also the `create-guid` function in the **lang | generator functions** library.

```xml
<format-guid-string
  unformatted_guid="formatted_guid">
</format-guid-string>
```

8.9.18.8  **left**

Result is a string containing the first `number` characters of `string`.

```xml
<left
  string="This is a sentence"
  number="4">
  result="This"
</left>
```

E.g. `string="This is a sentence"` and `number=4`, result is "This".

8.9.18.9  **left-trim**

Result is the input string with all leading whitespace characters removed.

```xml
<left-trim
  string="This is a sentence">
  result="This is a sentence"
</left-trim>
```

8.9.18.10  **lowercase**

Result is the lowercase version of the input `string`. For Unicode characters the corresponding lower-case characters (defined by the Unicode consortium) are used.

```xml
<lowercase
  string="This is a sentence">
  result="this is a sentence"
</lowercase>
```

8.9.18.11  **match-pattern**

Result is true if the input `string` matches the regular expression defined by `pattern`, else false. The specific regular expression syntax depends on the target language (see also **Regular expressions**).
8.9.18.12 pad-string-left

Returns a string which is padded to the left by a single specific character, up to a required length. The desired string length and the padding character are supplied as arguments.

string | Specifies the input string.
desired-length | Defines the desired length of the string after padding.
padding-char | Defines the character to use as padding character.

8.9.18.13 pad-string-right

Returns a string which is padded to the right by a single specific character, up to a required length. The desired string length and the padding character are supplied as arguments.

string | Specifies the input string.
desired-length | Defines the desired length of the string after padding.
padding-char | Defines the character to use as padding character.

8.9.18.14 repeat-string

Repeats the string supplied as argument \( n \) times. The count argument specifies the number of times to repeat the string.
8.9.18.15  replace

Result is a new string where each instance of oldstring, in the input string value, is replaced by newstring.

For an example, see Replacing Special Characters.

8.9.18.16  reversefind-substring

Returns the position of the first occurrence of substr within string, starting at position endindex, i.e. from right to left. The first character has position 1. If the substring could not be found, then the result is 0.

8.9.18.17  right

Result is a string containing the last number characters of string.

E.g. string="This is a sentence" and number=5, result is "tence".
8.9.18.18  right-trim

Result is the input string with all trailing whitespace characters removed.

\[
\text{right-trim} \\
\text{string} \rightarrow \text{result}
\]

8.9.18.19  string-compare

Returns the result of a string comparison of \text{string1} with \text{string2} taking case into account. If \text{string1} = \text{string2} then result is 0.

\[
\text{string-compare} \\
\text{string1} \\
\text{string2} \rightarrow \text{result}
\]

If \text{string1} is smaller than \text{string2} then result is < 0.

If \text{string1} is larger than \text{string2} then result is > 0.

8.9.18.20  string-compare-ignore-case

Returns the result of a string comparison of \text{string1} with \text{string2} ignoring case. If \text{string1} = \text{string2} then result is 0.

\[
\text{string-compare-ignore-case} \\
\text{string1} \\
\text{string2} \rightarrow \text{result}
\]

If \text{string1} is smaller than \text{string2} then result is < 0.

If \text{string1} is larger than \text{string2} then result is > 0.

8.9.18.21  uppercase

Result is the string input converted into uppercase. For Unicode characters the corresponding upper-case characters (defined by the Unicode consortium) are used.

\[
\text{uppercase} \\
\text{string} \rightarrow \text{result}
\]
8.9.19 mime

The mime library contains functions that let MapForce interact with MIME (Multipurpose Internet Mail Extensions) content. You typically need mime functions to create Web service calls that send custom MIME content (such as binary data). With mime functions, you can also handle the response MIME entity returned by a Web service call.

8.9.19.1 content-encoding

The content-encoding function returns the character encoding of the MIME entity supplied as argument. In case of an HTTP message, this is the value of the charset parameter from the Content-Type header.

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>mf:MimeEntity</td>
<td>The MIME entity content.</td>
</tr>
</tbody>
</table>

Example

If a mapping calls a Web service that returns a Content-Type header as text/json; charset=utf-8, then the function output is utf-8.

8.9.19.2 content-type

The content-type function returns the media type (Content-Type) of the MIME entity supplied as argument.
The `decode-mime-entity` function returns a string that represents the MIME entity decoded using the `charset` parameter included in the `Content-Type` header of the entity. If no `charset` was specified, you can supply a default encoding, by connecting the desired string value to the `default-encoding` parameter.

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mime-entity</td>
<td>mf:MimeEntity</td>
<td>The MIME entity to be decoded.</td>
</tr>
<tr>
<td>default-encoding</td>
<td>xs:string</td>
<td>The default encoding to use when no encoding was specified in the <code>charset</code> parameter.</td>
</tr>
</tbody>
</table>
Example
The following mapping calls a Web service that returns JSON content.

```
GET http://localhost/apitest/api/product
```

When this mapping runs, the result of the Web service call is supplied as input to the `decode-mime-entity` function. The function decodes the "raw" body returned by the call according to the `charset` parameter of the `Content-Type` header. The result is as follows:

8.9.19.4  `mime-entity`

The `mime-entity` function returns a MIME entity. The MIME entity is created from the content supplied to the `content` argument, and has the media type supplied to the `type` argument.
## Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>mf:MediaType</td>
<td>The media type that describes this MIME entity, for example: image/png.</td>
</tr>
<tr>
<td>content</td>
<td>xs:base64Binary</td>
<td>The actual data encoded using the Base64 encoding scheme. You can encode such data by using the charset-encode function.</td>
</tr>
</tbody>
</table>

## Example

The following mapping calls a Web service that expects unstructured text submitted via POST requests.

![Mapping Diagram](image)

In the mapping above, the input text is supplied as parameter to the mapping, with the help of a simple input component, see Supplying Parameters to the Mapping. The string received as input is converted to binary form with the help of the charset-encode function, using the utf-8 encoding. The mime-entity function takes the binary content as argument and returns the body of the request (the MIME entity), which is mapped to the Body input of the Web service call.

### 8.9.20 xbrl

The XBRL library contains functions that convert data into the QName format necessary for XBRL instance/taxonomy files.

The xbrl:ID and xbrl:measure child elements of the xbrl:unit element are mandatory in an XBRL instance file, and must be mapped for the mapping to be valid.
8.9.20.1 \textit{xbrl-measure-currency}

Result is the QName for items of the monetaryItemType from the ISO 4217 currency code.

\[
\begin{array}{c|c}
\text{xbrl-measure-currency} & \text{ISO-code} \\
& \text{result} \\
\end{array}
\]

8.9.20.2 \textit{xbrl-measure-pure}

Result is the QName for items of rates, percentages or ratios.

\[
\begin{array}{c|c}
\text{xbrl-measure-pure} & \text{result} \\
\end{array}
\]

8.9.20.3 \textit{xbrl-measure-shares}

Result is the QName for items of sharesItemType.

\[
\begin{array}{c|c}
\text{xbrl-measure-shares} & \text{result} \\
\end{array}
\]

8.9.21 \textit{xlsx}

The XLSX library contains functions that convert data to/from Excel date/time formats.

8.9.21.1 \textit{columnname-to-index}

Returns the index of the column with the given name. The name of column 1 is "A".

\[
\begin{array}{c|c}
\text{columnname-to-index} & \text{name} \\
& \text{result} \\
\end{array}
\]
8.9.21.2  *date-to-xlsx*

Returns the Excel representation of *date / time / datetime* value extracted from the source.

```
<date-to-xlsx>
<date>result</date-to-xlsx>
```

8.9.21.3  *datetime-to-xlsx*

Returns the Excel representation of the *datetime* value extracted from the source.

```
<datetime-to-xlsx>
<datetime>result</datetime-to-xlsx>
```

8.9.21.4  *index-to-columnname*

Returns the name of column *n*. The name of column 1 is "A".

```
<index-to-columnname>
<n>result</n>
```

8.9.21.5  *time-to-xlsx*

Returns the Excel representation of the *time* value extracted from the source.

```
<time-to-xlsx>
<time>result</time-to-xlsx>
```

8.9.21.6  *xlsx-to-date*

Returns the *date* value extracted from the Excel representation.

```
<xlsx-to-date>
<number>result</number>
```
8.9.21.7  *xlsx-to-datetime*

Returns the *datetime* value extracted from the Excel representation.

```
xlsx-to-datetime
  number  result
```

8.9.21.8  *xlsx-to-time*

Returns the *time* value extracted from the Excel representation.

```
xlsx-to-time
  number  result
```

8.9.22  *xpath2* | accessors

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

8.9.22.1  *base-uri*

The *base-uri* function takes a node argument as input, and returns the URI of the XML resource containing the node. The output is of type *xs:string*. MapForce returns an error if no input node is supplied.

8.9.22.2  *node-name*

The *node-name* function takes a node as its input argument and returns its QName. When the QName is represented as a string, it takes the form of prefix:localname if the node has a prefix, or localname if the node has no prefix. To obtain the namespace URI of a node, use the *namespace-URI-from-QName* function (in the library of QName-related functions).

8.9.22.3  *string*

The *string* function works like the *xs:string* constructor: it converts its argument to *xs:string*.

When the input argument is a value of an atomic type (for example *xs:decimal*), this atomic value is converted to a value of *xs:string* type. If the input argument is a node, the string value of the node is extracted. (The string value of a node is a concatenation of the values of the node's descendant nodes.)
8.9.23  xpath2 | anyURI functions

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

8.9.23.1  resolve-uri

The `resolve-uri` function takes a URI as its first argument (datatype `xs:string`) and resolves it against the URI in the second argument (datatype `xs:string`).

The result (datatype `xs:string`) is a combined URI. In this way a relative URI (the first argument) can be converted to an absolute URI by resolving it against a base URI.

In the screenshot above, the first argument provides the relative URI, the second argument the base URI. The resolved URI will be a concatenation of base URI and relative URI, so `C:\PathToMyFile\MyFile.xml`.

Note: Both arguments are of datatype `xs:string` and the process of combining is done by treating both inputs as strings. So there is no way of checking whether the resources identified by these URIs actually exist. MapForce returns an error if the second argument is not supplied.

8.9.24  xpath2 | boolean functions

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

The Boolean functions `true` and `false` take no argument and return the boolean constant values, `true` and `false`, respectively. They can be used where a constant boolean value is required.

8.9.24.1  false

Returns the Boolean value "false".

8.9.24.2  true

Returns the Boolean value "true".
8.9.25 xpath2 | constructors

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

The functions in the Constructors part of the XPath 2.0 functions library construct specific datatypes from the input text. Typically, the lexical format of the input text must be that expected of the datatype to be constructed. Otherwise, the transformation will not be successful.

For example, if you wish to construct an \( \text{xs:date} \) datatype, use the \( \text{xs:date} \) constructor function. The input text must have the lexical format of the \( \text{xs:date} \) datatype, which is: YYYY-MM-DD (screenshot below).

In the screenshot above, a string constant (2009-08-22) has been used to provide the input argument of the function. The input could also have been obtained from a node in the source document.

The \( \text{xs:date} \) function returns the input text (2009-08-22), which is of \( \text{xs:string} \) datatype (specified in the Constant component), as output of \( \text{xs:date} \) datatype.

When you mouseover the input argument in a function box, the expected datatype of the argument is displayed in a popup.

8.9.26 xpath2 | context functions

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

The Context functions library contains functions that provide the current date and time, the default collation used by the processor, and the size of the current sequence and the position of the current node.

8.9.26.1 current-date

Returns the current date (\( \text{xs:date} \)) from the system clock.
8.9.26.2  current-dateTime

Returns the current date and time (xs:dateTime) from the system clock.

8.9.26.3  current-time

Returns the current time (xs:time) from the system clock.

8.9.26.4  default-collation

The default-collation function takes no argument and returns the default collation, that is, the collation that is used when no collation is specified for a function where one can be specified. The Altova XSLT 2.0 Engine supports the Unicode codepoint collation only. Comparisons, including for the fn:max and fn:min functions, are based on this collation.

8.9.26.5  implicit-timezone

Returns the value of the "implicit timezone" property from the evaluation context.

8.9.26.6  last

The last and position functions take no argument. The last function returns the position of the last node in the context nodeset. The position function returns the position of the current node in the nodeset being processed.

The context nodeset at the nodes where the functions are directed, is the nodeset to which the functions will apply. In the screenshot below, the nodeset of Language elements is the context nodeset for the last and position functions.

In the example above, the last function returns the position of the last node of the context nodeset (the nodeset of Language elements) as the value of the number attribute. This value is
also the size of the nodeset since it indicates the number of nodes in the nodeset.

The position function returns the position of the Language node being currently processed. For each Language element node, its position within the nodeset of Language elements is output to the language/@position attribute node.

We would advise you to use the position and count functions from the core library.

8.9.27 xpath2 | durations, date and time functions

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

The XPath 2 duration and date and time functions enable you to adjust dates and times for the timezone, extract particular components from date-time data, and subtract one date-time unit from another.

The 'Adjust-to-Timezone' functions
Each of these related functions takes a date, time, or dateTime as the first argument and adjusts the input by adding, removing, or modifying the timezone component depending on the value of the second argument.

The following situations are possible when the first argument contains no timezone (for example, the date 2009-01 or the time 14:00:00).

- Timezone argument (the second argument of the function) is present: The result will contain the timezone specified in the second argument. The timezone in the second argument is added.
- Timezone argument (the second argument of the function) is absent: The result will contain the implicit timezone, which is the system's timezone. The system's timezone is added.
- Timezone argument (the second argument of the function) is empty: The result will contain no timezone.

The following situations are possible when the first argument contains a timezone (for example, the date 2009-01-01+01:00 or the time 14:00:00+01:00).

- Timezone argument (the second argument of the function) is present: The result will contain the timezone specified in the second argument. The original timezone is replaced by the timezone in the second argument.
- Timezone argument (the second argument of the function) is absent: The result will contain the implicit timezone, which is the system's timezone. The original timezone is replaced by the system's timezone.
- Timezone argument (the second argument of the function) is empty: The result will contain no timezone.

The 'From' functions
Each of the 'From' functions extracts a particular component from: (i) date or time data, and (ii) duration data. The results are of the xs:decimal datatype.

As an example of extracting a component from date or time data, consider the day-from-date function (screenshot below).
The input argument is a date \((2009-01-01)\) of type \texttt{xs:date}. The \texttt{day-from-date} function extracts the day component of the date \((1)\) as an \texttt{xs:decimal} datatype.

Extraction of time components from durations requires that the duration be specified either as \texttt{xs:yearMonthDuration} (for extracting years and months) or \texttt{xs:dayTimeDuration} (for extracting days, hours, minutes, and seconds). The result will be of type \texttt{xs:decimal}. The screenshot below shows a \texttt{dayTimeDuration} of \texttt{P2DT0H} being input to the \texttt{days-from-duration} function. The result is the \texttt{xs:decimal} 2.

The 'Subtract' functions
Each of the three subtraction functions enables you to subtract one time value from another and return a duration value. The three subtraction functions are: \texttt{subtract-dates}, \texttt{subtract-times}, \texttt{subtract-dateTimes}.

The screenshot below shows how the \texttt{subtract-dates} function is used to subtract two dates \((2009-10-22 \text{ minus } 2009-09-22)\). The result is the \texttt{dayTimeDuration P30D}.
8.9.28  xpath2 | node functions

The following XPath 2 node functions are available:

lang

The `lang` function takes a string argument that identifies a language code (such as `en`). The function returns `true` or `false` depending on whether the context node has an `xml:lang` attribute with a value that matches the argument of the function.

In the screenshot above notice the following:

1. In the source schema, the `Language` element has an `xml:lang` attribute.
2. Language nodes are filtered so that only those `Language` nodes having an `xml:lang` value of `en` are processed (the filter test is specified in the `equal` function).
3. The `Language` node is the context node at the point where the `en` element is created in the output document.
4. The output of the `lang` function (true or false) is sent to the `en/@exists` attribute node of the output. The argument of the function is provided by the string constant `en`. The `lang` function then checks whether the context node at this point (the `Language` element) has an `xml:lang` attribute with a value of `en` (the argument of the function). If yes, then `true` is returned, otherwise `false`. 
**local-name, name, namespace-uri**

The `local-name`, `name`, and `namespace-uri` functions, return, respectively, the local-name, name, and namespace URI of the input node. For example, for the node `altova:Products`, the local-name is `Products`, the name is `altova:Products`, and the namespace URI is the URI of the namespace to which the `altova:` prefix is bound (say, `http://www.altova.com/mapforce`).

Each of these three functions has two variants:

- With no argument: the function is then applied to the context node (for an example of a context node, see the example given for the `lang` function above).
- An argument that must be a node: the function is applied to the submitted node.

The output of each of these six variants is a string.

**number**

Converts an input string into a number. Also converts a boolean input to a number.

The `number` function takes a node as input, atomizes the node (that is, extracts its contents), and converts the value to a decimal and returns the converted value. The only types that can be converted to numbers are booleans, strings, and other numeric types. Non-numeric input values (such as a non-numeric string) result in `NaN` (Not a Number).

There are two variants of the `number` function:

- With no argument: the function is then applied to the context node (for an example of a context node, see the example given for the `lang` function above).
- An argument that must be a node: the function is applied to the submitted node.

### 8.9.29 XPath 2 | numeric functions

The following XPath 2 numeric functions are available:

**abs**

The `abs` function takes a numeric value as input and returns its absolute value as a decimal. For example, if the input argument is `-2` or `+2`, the function returns `2`.

**round-half-to-even**

The `round-half-to-even` function rounds the supplied number (first argument) to the degree of precision (number of decimal places) supplied in the optional second argument. For example, if the first argument is `2.141567` and the second argument is `3`, then the first argument (the number) is rounded to three decimal places, so the result will be `2.141`. If no precision (second argument) is supplied, the number is rounded to zero decimal places, that is, to an integer.

The 'even' in the name of the function refers to the rounding to an even number when a digit in the supplied number is midway between two values. For example, `round-half-to-even(3.475, 2)` would return `3.48`. 
8.9.30  xpath2 | string functions

The following XPath 2 string functions are available:

**compare**

The `compare` function takes two strings as arguments and compares them for equality and alphabetically. If `String-1` is alphabetically less than `String-2` (for example the two string are: A and B), then the function returns -1. If the two strings are equal (for example, A and A), the function returns 0. If `String-1` is greater than `String-2` (for example, B and A), then the function returns +1.

A variant of this function allows you to choose what collation is to be used to compare the strings. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

**ends-with**

The `ends-with` function tests whether `String-1` ends with `String-2`. If yes, the function returns true, otherwise false.

A variant of this function allows you to choose what collation is to be used to compare the strings. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

**escape-uri**

The `escape-uri` function takes a URI as input for the first string argument and applies the URI escaping conventions of RFC 2396 to the string. The second boolean argument (escape-reserved) should be set to true() if characters with a reserved meaning in URIs are to be escaped (for example "+" or "/").

For example:

```
escape-uri("My A+B.doc", true()) would give My%20A%2B.doc
escape-uri("My A+B.doc", false()) would give My%20A%2B.doc
```

**lower-case**

The `lower-case` function takes a string as its argument and converts every upper-case character in the string to its corresponding lower-case character.

**matches**

The `matches` function tests whether a supplied string (the first argument) matches a regular expression (the second argument). The syntax of regular expressions must be that defined for the pattern facet of XML Schema. The function returns true if the string matches the regular expression, false otherwise.

The function takes an optional flags argument. Four flags are defined (i, m, s, x). Multiple flags can be used: for example, imx. If no flag is used, the default values of all four flags are used.
The meaning of the four flags are as follows:

- **i**  Use case-insensitive mode. The default is case-sensitive.
- **m**  Use multiline mode, in which the input string is considered to have multiple lines, each separated by a newline character (\x0a). The meta characters `^` and `$` indicate the beginning and end of each line. The default is string mode, in which the string starts and ends with the meta characters `^` and `$`.
- **s**  Use dot-all mode. The default is not-dot-all mode, in which the meta character `.` matches all characters except the newline character (\x0a). In dot-all mode, the dot also matches the newline character.
- **x**  Ignore whitespace. By default whitespace characters are not ignored.

### normalize-unicode

The `normalize-unicode` function normalizes the input string (the first argument) according to the rules of the normalization form specified (the second argument). The normalization forms NFC, NFD, NFKC, and NFKD are supported.

### replace

The `replace` function takes the string supplied in the first argument as input, looks for matches as specified in a regular expression (the second argument), and replaces the matches with the string in the third argument.

The rules for matching are as specified for the matches attribute above. The function also takes an optional `flags` argument. The flags are as described in the `matches` function above.

### starts-with

The `starts-with` function tests whether `String-1` starts with `String-2`. If yes, the function returns `true`, otherwise `false`.

A variant of this function allows you to choose what collation is to be used to compare the strings. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

### substring-after

The substring-after function returns that part of `String-1` (the first argument) that occurs after the test string, `String-2` (the second argument). An optional third argument specifies the collation to use for the string comparison. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

### substring-before

The substring-before function returns that part of `String-1` (the first argument) that occurs before the test string, `String-2` (the second argument). An optional third argument specifies the collation to use for the string comparison. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.
The `upper-case` function takes a string as its argument and converts every lower-case character in the string to its corresponding upper-case character.

### 8.9.31 `xslt | xpath functions`

The functions in the XPath Functions library are XPath 1.0 nodeset functions. Each of these functions takes a node or nodeset as its context and returns information about that node or nodeset. These function typically have:

- a context node (in the screenshot below, the context node for the `lang` function is the `Language` element of the source schema).
- an input argument (in the screenshot below, the input argument for the `lang` function is the string constant `en`). The `last` and `position` functions take no argument.

#### lang

The `lang` function takes a string argument that identifies a language code (such as `en`). The function returns `true` or `false` depending on whether the context node has an `xml:lang` attribute with a value that matches the argument of the function. In the screenshot above notice the following:

1. In the source schema, the `Language` element has an `xml:lang` attribute.
2. Language nodes are filtered so that only those `Language` nodes having an `xml:lang` value of `en` are processed (the filter test is specified in the `equal` function).
3. The `Language` node is the context node at the point where the `en` element is created in the output document.
4. The output of the `lang` function (`true` or `false`) is sent to the `en/@exists` attribute node of the output. The argument of the function is provided by the string constant `en`. The `lang` function then checks whether the context node at this point (the `Language` element) has an `xml:lang` attribute with a value of `en` (the argument of the function). If yes, then `true` is returned, otherwise `false`. 
last, position

The last and position functions take no argument. The last function returns the position of the last node in the context nodeset. The position function returns the position of the current node in the nodeset being processed.

The context nodeset at the nodes where the functions are directed is the nodeset to which the functions will apply. In the screenshot below, the nodeset of Language elements is the context nodeset for the last and position functions.

In the example above, the last function returns the position of the last node of the context nodeset (the nodeset of Language elements) as the value of the number attribute. This value is also the size of the nodeset since it indicates the number of nodes in the nodeset.

The position function returns the position of the Language node being currently processed. For each Language element node, its position within the nodeset of Language elements is output to the language/@position attribute node.

name, local-name, namespace-uri

These functions are all used the same way and return, respectively, the name, local-name, and namespace URI of the input node. The screenshot below shows how these functions are used. Notice that no context node is specified.

The name function returns the name of the Language node and outputs it to the language/@elementname attribute. If the argument of any of these functions is a nodeset instead of a single node, the name (or local-name or namespace URI) of the first node in the nodeset is returned.
The `name` function returns the QName of the node; the `local-name` function returns the local-name part of the node's QName. For example, if a node's QName is `altova:MyNode`, then `MyNode` is the local name.

The namespace URI is the URI of the namespace to which the node belongs. For example, the `xmlns:altova` prefix can be declared to map to a namespace URI in this way:

```
xmns:altova="http://www.altova.com/namespaces"
```

**Note:** Additional XPath 1.0 functions can be found in the Core function library.

### 8.9.32 xslt | xslt functions

The functions in the XSLT Functions library are XSLT 1.0 functions.

#### 8.9.32.1 current

The `current` function takes no argument and returns the current node.

#### 8.9.32.2 document

The `document` function addresses an external XML document (with the `uri` argument; see screenshot below). The optional nodeset argument specifies a node, the base URI of which is used to resolve the URI supplied as the first argument if this URI is relative. The result is output to a node in the output document.
Note that the `uri` argument is a string that must be an absolute file path.

### 8.9.32.3 element-available

The `element-available` function tests whether an element, entered as the only string argument of the function, is supported by the XSLT processor.

The argument string is evaluated as a QName. Therefore, XSLT elements must have an `xsl:` prefix and XML Schema elements must have an `xs:` prefix—since these are the prefixes declared for these namespaces in the underlying XSLT that will be generated for the mapping.

The function returns a boolean.

### 8.9.32.4 function-available

The `function-available` function is similar to the `element-available` function and tests whether the function name supplied as the function's argument is supported by the XSLT processor.

The input string is evaluated as a QName. The function returns a boolean.
8.9.32.5  *generate-id*

The *generate-id* function generates a unique string that identifies the first node in the nodeset identified by the optional input argument.

If no argument is supplied, the ID is generated on the context node. The result can be directed to any node in the output document.

8.9.32.6  *system-property*

The *system-property* function returns properties of the XSLT processor (the system). Three system properties, all in the XSLT namespace, are mandatory for XSLT processors. These are `xsl:version`, `xsl:vendor`, and `xsl:vendor-url`.

The input string is evaluated as a QName and so must have the `xsl:prefix`, since this is the prefix associated with the XSLT namespace in the underlying XSLT stylesheet.

8.9.32.7  *unparsed-entity-uri*

If you are using a DTD, you can declare an unparsed entity in it. This unparsed entity (for example an image) will have a URI that locates the unparsed entity.

The input string of the function must match the name of the unparsed entity that has been declared in the DTD. The function then returns the URI of the unparsed entity, which can then be directed to a node in the output document, for example, to an `href` node.
Chapter 9
Implementing SOAP Web Services
9 Implementing SOAP Web Services

With MapForce Enterprise edition you can generate Java or C# program code that implements SOAP Web services, from existing Web Services Description Language (WSDL) files. You can map data to WSDL operations as follows:

- From the input of the WSDL operation to any data sources supported by MapForce, including flat files, XML, XBRL, EDI, Microsoft Excel, and databases.
- From data sources supported by MapForce to the output of the WSDL operation.

MapForce supports WSDL 1.1 and WSDL 2.0 (for additional support information and limitations, see SOAP/WSDL Support Notes and Java SOAP Web Services Specifics).

Note: As an alternative to developing SOAP Web services from existing WSDL files, you can also deploy MapForce mappings to MapForce Server, where you can run them through an API or command line call. Additionally, with FlowForce Server, you can further expose the mapping as a generic (not SOAP) Web service on the machine where FlowForce Server runs. Any mapping designed for the BUILT-IN transformation language qualifies for deployment to MapForce Server or FlowForce Server. FlowForce Server is a dedicated server solution that is capable not only to convert MapForce mappings into generic Web services, but also run them as scheduled or on demand jobs. For more information, see Deploying Mappings to FlowForce Server.

Prerequisites
To create a Web service with MapForce, the WSDL file of the Web service is required. Note that you can design WSDL files and test SOAP requests with XMLSpy, for example. Additionally, you need platform-specific software required to build, deploy, and run a SOAP Web service, for example:

Java
- Java Development Kit 1.7 or later
- Apache Tomcat: https://tomcat.apache.org/
- Apache Axis2: https://axis.apache.org/axis2/java/core/, a SOAP framework running within Tomcat
- Apache Ant: https://ant.apache.org/

C#
- Microsoft Internet Information Services (IIS) version 5.0 or later.

How it works
The following diagram illustrates the entire process, starting from implementing a Web service from an existing WSDL file up to the stage when it can be called by SOAP client applications. Note that only the first two steps take place in MapForce. Steps 3 and 4 depend on the specifics of the third-party tools and environments used by your organization.
The process illustrated above works as follows:

<table>
<thead>
<tr>
<th>Step #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Design the Web Service from an existing WSDL file</td>
</tr>
<tr>
<td>2.</td>
<td>Generate Java or C# code</td>
</tr>
<tr>
<td>3.</td>
<td>Build and deploy the Web service to a server</td>
</tr>
<tr>
<td>4.</td>
<td>Host Web service</td>
</tr>
<tr>
<td>5.</td>
<td>Call Web service</td>
</tr>
</tbody>
</table>

Once you have the WSDL file, you can start a new MapForce Web Service Project. A MapForce Web Service Project has a predefined structure that enables you to quickly access a particular WSDL operation or Web service in it. When you start a MapForce Web Service Project, you can optionally instruct MapForce to generate mapping files automatically for all WSDL operations found in the WSDL file. As an alternative, you can select individually the WSDL Services and WSDL Endpoints to be included in the MapForce Web Service Project. In the graphical user interface of MapForce, WSDL operations correspond to mappings, and the operation's input and output parameters appear as components in the mapping.

While working with mappings that include input or output of WSDL operations, you can preview the execution result as you would do for any other mapping. To do this, you need a sample SOAP request file (XML format) that conforms to the WSDL grammar of the Web service (such instance files can be generated with XMLSpy).
2. Generate Java or C# Code

When your Web Service Project is ready in MapForce, you can generate the C# or Java code either for individual mappings (for testing purposes), or for the entire project. In the latter case, MapForce creates a complete Web service server in the language of choice (C# or Java).

3. Build and deploy the Web service to a server

In this step, you compile the generated C# or Java code (outside MapForce) and deploy it to your custom Web server. Your Web server must be capable to host SOAP services and respond to SOAP client calls, see the "Prerequisites" section above.

4. Host Web service

The compiled Web service can be hosted by the server of your choice, for example:
- Apache Tomcat
- Internet Information Services (IIS)

5. Call Web service

The clients of the Web service include any application that is capable of generating SOAP calls and processing the response, including XMLSpy or MapForce itself.

For more information about step 1, see Creating SOAP Web Service Projects. For all other steps, see Example: Generating SOAP Web Services (C#) and Example: Generating SOAP Web Services (Java)
# 9.1 SOAP/WSDL Support Notes

The following table summarizes the WSDL support details in MapForce.

| WSDL support | Version 1.1, W3C Note from [http://www.w3.org/TR/wsdl](http://www.w3.org/TR/wsdl)  
Version 2.0, W3C Recommendation from [http://www.w3.org/TR/wsdl20/](http://www.w3.org/TR/wsdl20/) |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WSDL type system</td>
<td>XML Schema 2001</td>
</tr>
</tbody>
</table>
Version 1.2: [http://www.w3.org/TR/soap12-part0/](http://www.w3.org/TR/soap12-part0/) |
| Protocols | SOAP over HTTP (HTTP POST, HTTP GET protocols are not supported).  |
| C# | The SOAPAction must be different for each operation in C#.  |
| Bindings | Multiple operations with same name are currently not supported (Section 2.5 of the WSDL 1.1 specification).  |
| style/use |  
- Document/literal: supported.  
- RPC/literal: supported in C#  
- RPC/encoded: limited support  
- One style/use per Web service (Java), or operation (C#) is currently supported.  |
| SOAP headers | Depends on underlying platform.  |
| SOAP encodingStyle | If use="encoded", encoding style "[http://schemas.xmlsoap.org/soap/encoding/](http://schemas.xmlsoap.org/soap/encoding/)/" for complete soap:Body is assumed. There is no support for other encoding styles.  
The encodingStyle attribute is ignored in messages (Section 4.1.1 of the SOAP 1.1 specification).  |
| References |  
- References to external resources are currently not supported (Section 5.4.1 of the SOAP 1.1 specification).  
- References to independent elements are supported.  |
| SOAP-ENC:Array | Linear access is supported. Partial arrays and sparse arrays are currently not supported.  |
| Custom SOAP enhancements | Not supported.  |
| Default or fixed values in schemas | Not supported.  |
| Non SOAP message validation | Not validated; passed on to underlying framework.  |
| Namespaces | Non namespace entries are invalid WSDL, and are therefore not supported (WSDL and XML 1.0).  |
WSDL 1.1

**portType**
A `<portType>` element defines a Web service interface, namely:
- the **operations** that can be performed.
- the **messages** involved in each operation as inputs and outputs.

**types**
The `<types>` element define the datatypes that are used by the Web service. MapForce supports XML Schemas in WSDL files, as this is the most common type system for WSDL files. MapForce displays these elements (datatypes) as items in a (message) component, allowing you to map them to other item/constructs directly.

**message**
The `<message>` element defines the **parts** of each message and the **data elements** of an operation's input and output parameters. These are the messages exchanged by the client and server. There are three types of messages: Input, Output and Fault. In MapForce, each **message** is a **component** from or to which you can map other items. Messages can consist of one or more message parts.

When using the document / literal combination in MapForce, it is necessary that the **message** / **part** element refer to a global element as opposed to a **type**. For example, in the following code, the **element** attribute refers to a global element defined in a schema (`ns2:Vendor`):

```xml
<message name="processRequest">
  <part name="inputData" element="ns2:Vendor"/>
</message>
```

Whereas the following code references a type in the schema:

```xml
<message name="processRequest">
  <part name="inputData" type="ns2:VendorType"/>
</message>
```

**operation**
Operations use messages as input and output parameters. An operation can have:
- one Input message
- zero or more Output messages
- zero or more Fault messages

Input messages can only be used as source components. Output and Fault messages can only be used as target components.

WSDL 2.0
WSDL 2.0 is substantially different from WSDL 1.1, the main differences being:
- PortTypes have been renamed to interfaces.
- Messages and parts are now defined using the XML Schema type system in the types element.
- Ports have been renamed to endpoints.
- WSDL 2.0 operation inputs and outputs are defined by the XML schema.

In MapForce, the Component Settings dialog box of a WSDL component displays "Endpoint" for both WSDL 1.1 Ports and WSDL 2.0 endpoints.
9.2 Creating SOAP Web Service Projects

This example shows you how to create a SOAP Web service in MapForce. The mapping project created in this example is also available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\Query Person database.mfp`. The Web service in this example has the following goals:

- Retrieve from a Microsoft Access database on a server a set of person records that satisfy some search criteria, through a SOAP Request. (The query is entered at runtime on the client, and is then sent to the server. The server then returns an XML response with the search results).
- Insert new records into the Microsoft Access database, also through SOAP requests. (The person details are provided in the SOAP Request. On success, the server returns a confirmation that the person details have been successfully added to the database.)

Note the following:

- The process of creating the Web service in MapForce does not depend on the target programming language; it is identical when generating Java, or C# Web services. The differences arise only when you compile and deploy the Web service on the web server, as further described in this documentation.
- In order to generate sample SOAP requests required to preview the mapping output, this example uses XMLSpy Enterprise Edition. The source WSDL file of this MapForce project was also created with XMLSpy (and can be validated with it). Designing Web services in MapForce is also possible without XMLSpy; however, in this case you will need a SOAP client capable to generate SOAP requests from WSDL files.

Creating a Web service project

As stated before, a WSDL file is required to create a SOAP Web service with MapForce. In this example, we will use a WSDL file that was created with XMLSpy Enterprise edition. The image below illustrates the WSDL structure as it is displayed in the WSDL view of XMLSpy. The important thing to notice is that it defines two operations (`getPerson` and `putPerson`) that correspond to the goals outlined above.

![WSDL Diagram](image-url)

**Note:** The WSDL file must be valid before you can create a MapForce Web Service project from it.

To create a Web service project from this web service, follow the steps below:

1. On the **File** menu, click **New**, and select "Web Service Project".
2. Browse for the `query.wsdl` file available in the folder `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\`. After you select the WSDL file, MapForce automatically fills in the remaining fields.

3. Click OK.
4. When prompted, enter the name of the new WSDL project, and click Save.

The Project window shows the project and WSDL name, as well as each of the operations defined in the WSDL file. The two operations are `getPerson` and `putPerson`. 
Designing the "getPerson" operation

Double-click the getPerson.mfd file in the Project window to load its contents in the main pane. The getPersonSoapIn component contains the query (item) which will be used to query the database through the Web service. The getPersonSoapOut component contains the Person structure defined in the WSDL file.

On the Insert menu, click Database, and add to the mapping area the following Access database: <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\people.mdb. The database contains only one table, Persons, whose fields match those in the getPersonSoapOut component. For more information about adding databases as mapping components, see Databases and MapForce.
Draw connections between components as shown below:

The mapping design above is not yet complete but clearly illustrates what will be happening at runtime. The connection between `part:Query` and `part:Result` represents the Web service call. The `part:Query` item of the `getPersonSoapIn` component is the query placeholder (it is where the query string is entered in the SOAP client). The connections between the database table and the `Person` item of the `getPersonSoapOutput` component extract values from the corresponding database fields and pass them to the corresponding target items. You can already quickly preview the output as follows:

1. Right-click the `getPersonSoapInput` component, and select **Properties** from the context menu.
2. Click **Browse** and select the following example request: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\getPersonRequest.xml`.
3. Click **OK**.
4. Click the **Output** tab.

At this stage, the mapping extracts all the records from the database table, which is not very useful. Therefore, we need to add a filter and extract only items that match the request string.

1. Add the `concat` function, by dragging it from the **Libraries window**. This function will be used to concatenate the `First` and `Last` names of each person.
2. On the **Insert** menu, click **Constant**, and add a constant which contains one space
character. The constant will supply the space character between the first and the last name extracted from the database.

3. Add the `contains` function, by dragging it from the Libraries window.

4. Right-click the connection between the source `Persons` table and `Person` target node, and select **Insert Filter: Nodes/Rows** from the context menu.

5. Draw connections between components as shown below.

As illustrated above, the `contains` function returns `true` when the query string matches the full or partial name of any person in the database (where "name" is provided by the `concat` function, and it consists of the first name, followed by a space, followed by the last name). When there is a match, the details of the matching person are included in the response message. Therefore, in this example, the search value "Ro" would return all the following records: "Martin Rope", "Ronald Superstring", "Robert Darkmatter", and "Roger Gravity".

**Designing the "putPerson" operation**

Double-click the `putPerson.mfd` file in the Project window to load its contents in the main pane. The `putPersonSoapInput` component supplies the request structure (items that will be inserted into the database through the Web service). The `putPersonSoapOutput` component represents the result of the Web service call. It contains a confirmation part to display the outcome of the call to the Web service caller.

Follow the same instructions as above to add the following to the mapping:

- The `people.mdb` Access database (this is where data supplied by the Web service
The `concat` function and five constants (these will be used to build the confirmation string shown to the caller of the Web service)

Draw the connections as shown below:

"putPerson" mapping

In the mapping above, the connection between `part: Put` and the database will cause the database to be updated at runtime. On the target side, the `part: Confirmation` response is obtained by concatenating various fields from the request, as well as some text supplied by constants. For more information about mappings which update databases, see Mapping Data to Databases.

To preview the output of the mapping:

1. Right-click the `getPersonSoapInput` component.
2. Click Browse and select the following example request: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\putPersonRequest.xml`.
3. Click the Output tab.

When you preview the mapping, MapForce will generate the INSERT query required to update the database and display it in the Output window, but will not perform the actual update. To run the query against the database, select the menu command Output | Run SQL-Script.

By contrast, when the compiled Web service is deployed to a server and called by a client, the
Implementing SOAP Web Services

Creating SOAP Web Service Projects

database will be updated with immediate effect. See also Executing Mappings Which Modify Databases.

Previewing the mapping output

Mappings created from WSDL files can be previewed in MapForce before they are deployed to an actual Web server. To preview such mappings, a sample SOAP request (XML file) is necessary which conforms to the WSDL file. You can create a sample request for each operation (mapping) inside your Web service project either with XMLSpy or another SOAP client of your choice. If you use XMLSpy, run the menu command SOAP | Create New SOAP Request to generate the request. If XMLSpy is installed alongside MapForce, you can generate a sample request directly from MapForce, for example:

1. Right-click the getPersonSoapInput component on the mapping, and select Properties from the context menu.

2. Click Create.

3. When prompted, save the XML file to a location on the disk.

The request file is generated with some dummy input values (for example, "String") instead of input parameters. Before using the request file, make sure to replace the dummy values with the actual request values that you want to use. For this project, two sample requests are available:

1. `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\getPersonRequest.xml`. Use this file to preview the output of the getPerson mapping. By default, this request file supplies the value "Ro" as input to the Web service. This will retrieve from the database all people records where the last or first name contains the text
To obtain a different output while previewing the two mappings, therefore, edit the files above so that they contain the input values you wish to test.

Next steps
You are now ready to generate code that implements the Web service (either Java or C#), compile it, and deploy it to a server, as further described in the following examples:

- Example: Generating SOAP Web Services (C#)
- Example: Generating SOAP Web Services (Java)
9.3 Defining SOAP Web Service Faults

A WSDL file can contain a fault element for an operation and a message attribute that contains the fault message. Accordingly, when designing a SOAP Web service in MapForce, you can add a “Fault” component to your mapping. This will cause the Web service to throw an exception if a certain condition is satisfied.

To insert a Fault component in a mapping, a fault element must be present in the WSDL file.

You can define the failure condition inside the mapping, and you can also set the text of the error message. When the condition is satisfied, the Web service will fail as follows:

- In a live environment (at execution time, after the Web service is deployed to a server), the Web service will fail and return the custom-defined message to the caller.
- In MapForce (at design time), the mapping execution stops, and you can view the message in the Message window.

To add a fault component to a MapForce mapping:

1. On the Insert menu, click Exception. (Alternatively, click the Exception toolbar button.)
2. Select Create WSDL-fault.

Note: If the Create WSDL-fault option is disabled and you see the message "No faults for this operation", this means that no faults exist in the WSDL file, and, consequently, none can be added in MapForce.

For an example of a mapping which contains a Web service fault, open the following sample MapForce project: <Documents>\Altova\MapForce2019\MapForceExamples\TimeService\TimeService.mfp. Double-click the mapping (operation) getTimeZoneTime.mfd.
In the mapping above, the exception should be triggered when `n0:timezone` is not equal to UTC.

- The `equal` function checks to see if timezone equals UTC, with the bool result being passed on to the filter.
- If the condition is false, i.e. something other than UTC, the `on-false` parameter of the filter activates the `Fault: errorSoapOut` exception and the mapping process is halted. (Note that you can also connect the exception to the on-true parameter, if that is what you need.)
- Two sets of error text are supplied by the SoapFault message.

It is very important to note the filter placement in the example:

- Both parameters of the filter component, `on-true` and `on-false`, must be mapped. One of them needs to be mapped to the fault component, and the other, to the target component that receives the filtered source data. Otherwise, the fault component will never be triggered.
- The exception and target components must be directly connected to the filter component. Functions or other components may not be placed between the filter and either the exception, or target components.
9.4 Java SOAP Web Services Specifics

MapForce generates all necessary code and scripts needed to create a SOAP Web service. However, for the Web service to be available to consumers, the generated code must be built and deployed to the Apache Tomcat server with Axis2 installed. Additionally, Apache Ant is required to build the Java code.

Generating and building Java code

1. Open the Web service project from which you want to generate code. See the previous section, Creating Web Service Projects from WSDL files, for an example of how to create such a project.
2. On the Project menu, click Generate code in | Java, and select the target directory. When code generation completes, several folders and files are created in the target directory, including a com directory which contains Altova generic classes, as well as the actual classes of the Web service project.
3. Build the generated Java code (by supplying to Apache Ant the build.xml file generated by MapForce). As a result, an Axis Archive File (*.aar) file is created, which you can then deploy to Axis2.

Note for Eclipse users: when you open the generated project with Eclipse, you may see an error like "The import org.apache cannot be resolved". In this case, make sure that the Axis2 libraries are added to the Java build path. To add the Axis2 libraries to the Java build path in Eclipse 4.4.2, do the following:

1. Right-click the project in the Package Explorer, and select Properties.
2. Click Java Build Path.
3. On the Libraries tab, click Add External JARs, and add the Axis2 libraries from the <AXIS2_HOME>\lib folder.

Deploying the Web service

To deploy the Web service, do one of the following:

- Open the "Upload Services" Web administration page of Axis2 and upload the .aar file created in the previous step
- Do a manual upload. For example, if your Tomcat server was installed to the folder <TOMCAT_HOME>, you can manually copy the .aar file to <TOMCAT_HOME>\webapps\axis2\WEB-INF\services.

For a step-by-step example, see Example: Generating SOAP Web Services (Java).

Undeployment

Delete the *.aar file from the <TOMCAT_HOME>\webapps\axis2\WEB-INF\services folder.

Axis2 limitations

Axis2 support for RPC/encoded is limited. MapForce can, however, generate RPC/encoded Web services (both SOAP 1.1 and SOAP 1.2). The limitation is that the original WSDL is not retrieved.
from the Web server.

This means that, for example, http://127.0.0.1/axis2/services/WS2DB?wsdl would not return a usable .wsdl file.

For **document/literal** Web services, the URL above will provide a usable and correct .wsdl file. It will differ from the original, however: comments will be stripped out, and namespaces will be changed. It will, however, still have the same semantics as the original .wsdl file with which the service was generated.

Although Axis2 does not support **RPC/encoded**, it is able to generate WSDL from deployed Java code (compiled code), and thus MapForce-generated code can process **RPC/encoded** messages; Axis2 is just used for transport.

**Known issue: namespaces in the SOAP response message**

The code generated by MapForce instantiates the `javax.xml.transform.TransformerFactory` class. When the class implementation is loaded, it might be read from the `javax.xml.transform.TransformerFactory` system property (for details, refer to the Java documentation of this class).

If the system property `javax.xml.transform.TransformerFactory` in your Java environment defines the implementation `org.apache.xalan.transformer.TransformerIdentityImpl`, the SOAP response message may be generated with incorrect namespaces.

To prevent this issue, it is recommended to use the `org.apache.xalan.xsltc.trax.TransformerFactoryImpl` implementation. You can do this by adding the following parameter to the Java Virtual Machine (JVM):

```
-Djavax.xml.transform.TransformerFactory=org.apache.xalan.xsltc.trax.TransformerFactoryImpl
```

The instructions for adding the parameter to the JVM depend on the operating system and server you are using. The following instructions are applicable to Apache Tomcat 7.0 configured to run as a service application on Windows 7:

1. Run `Tomcat7w.exe` (this file is located in the `bin` subfolder of the Tomcat distribution root folder, `CATALINA_HOME`).
2. Click the **Java** tab.
3. At the end of "Java Options" box, enter:
   ```
   -Djavax.xml.transform.TransformerFactory=org.apache.xalan.xsltc.trax.TransformerFactoryImpl
   ```
9.5 Example: Generating SOAP Web Services (C#)

This example illustrates how to generate a SOAP Web service with MapForce. You will generate C# program code from a sample MapForce project available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\Query Person database.mfp`. This project implements a SOAP Web service that exposes two operations: `getPerson` and `putPerson`. This Web service communicates in the background with a Microsoft Access database to retrieve or add person details, respectively, through the operations above. For an example of how to create such projects in MapForce, see [Creating Web Service Projects from WSDL Files](#).

This example uses Visual Studio 2015 to compile the generated C# code. The solution will be deployed to Internet Information Services (IIS) 10 with ASP.NET enabled.

For simplicity, this example uses a minimum Web server configuration and should not be taken as prescriptive. In a production environment, you may need to use a different approach as demanded by the specifics of your Web server, the security policies in your organization, and other factors.

To call the SOAP Web service, this example makes use of XMLSpy Enterprise Edition. If you do not have XMLSpy, you will need a SOAP client to test the Web service, or, alternatively, write program code that can call the Web service.

**Note:** If you are using Visual Studio 2008, and IIS 7.x, you may first need to install the Windows feature "IIS Metabase and IIS 6 configuration compatibility".

**Step 1: Prepare the query.wsdl file**

This example Web service originates from the following WSDL file: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\query.wsdl`. By default, the `query.wsdl` file defines the service details for both C# and Java in two separate sections, one of which is commented out. Therefore, before generating C# code, locate the following section in the `query.wsdl` file and make sure that it is not commented out:

```xml
<service name="WS2DB">
  <port name="WS2DBSoapPort" binding="tns:WS2DBSoapBinding">
    <soap:address location="http://localhost/WS2DB/WS2DB.asmx"/>
  </port>
</service>
```

Also, make sure that the following section (only applicable to Java) is commented out:

```xml
<!--<service name="WS2DB">
  <port name="WS2DBSoapPort" binding="tns:WS2DBSoapBinding">
    <soap:address location="http://localhost:8080/axis/services/WS2DB"/>
  </port>
</service>-->
Step 2: Generate C# code and build it

Run MapForce and open the "Query Person database.mfp" project from the folder `<Documents>`\Altova\MapForce2019\MapForceExamples\Tutorial\. Right-click the project in the Project window, and select Generate code in | C#. MapForce generates program code and displays the outcome in the Messages window:

By default, code will be generated in a subdirectory called "output", relative to the project's directory. For the scope of this tutorial, we will leave the default settings as is. However, if you want to change the output directory, see Setting the Code Generation Settings.

Run Visual Studio and open the generated solution. Be patient while all projects in the solution finished loading.

You may need to run Visual Studio as administrator. Also, this example solution must be compiled as a 32-bit application, as shown below.

You can build the solution as follows:

- On the Build menu, click Build Solution (Ctrl+Shift+B).

Note that this example queries a Microsoft Access database using the ADO provider for Access, which only works with 32-bit applications. Therefore, the solution must be compiled as a 32-bit application, as follows:

- On the Build menu, click Configuration Manager, create a new solution platform for x86, and build again.
Step 3: Deploy the Web service to IIS

In the Solution Explorer window of Visual Studio, right-click the project `services-Site/WS2DB` and select “Publish Web App” from the content menu.

When prompted to select a publish target, select “Custom”.
When prompted to enter a profile name, enter a value that will be easy to recognize later, for example "iis".

Select "File System" as publish method and choose a destination folder (in this example, "C:\WS2DB").
Click **Next**. Leave the default settings as is.
Click **Next**, and then click **Publish**.
Step 4: Configure the application in IIS Manager

You can add the new application to IIS as follows:

1. In IIS, right-click **Sites** | **Default Web Site**, and select **Add Application** from the context menu.
2. Under "Alias name", enter a name to identify your web application ("services", in this example).
3. Under "Physical path", enter the path where the application was previously published ("C:\WS2DB", in this example).
Additionally, for this particular example, it is necessary that 32-bit applications are enabled for the pool where the app is running.

1. In IIS, right-click the application pool where the app was deployed, and select Advanced Settings from the context menu.
2. Under Enable 32-bit Applications, select the option True.
Finally, since this example reads data from a Microsoft Access database, the IIS application pool must have permissions to access the database file. Unless you modified the MapForce project, the database is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\people.mdb`.

1. In Windows Explorer, right-click the `people.mdb` file, and select Properties.
3. Click Add, and enter the object name as follows: `IIS AppPool\DefaultAppPool`. Change "DefaultAppPool" to a different value if you deployed the app to a pool other than the default one.
4. Grant the required permissions, and click Apply. (In this example, permission to read and write the database file is required.)
To test if the Web service is up and running, access the following URL: http://localhost/WS2DB.

**WS2DBWebService**

The following operations are supported. For a formal definition, please review the Service Description.

- `getPerson`
- `putPerson`

If the Web service details load in the browser, you can proceed to calling the Web service from a client application, as shown below.

**Calling the Web service**

If you followed the steps above literally, then the WSDL of the Web service is available at http://localhost/WS2DB/query.wsdl. To call the Web service we just deployed, we will use XMLSpy, although you can use, of course, other SOAP clients as well.

![Screenshot of XMLSpy window showing WSDL file selection](image)

Click OK. When prompted to select a SOAP Operation, select `getPerson`.

![Screenshot of XMLSpy window showing SOAP operation selection](image)

XMLSpy generates a sample request like the one below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <getPerson SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <Query xsi:type="xsd:string">String</Query>
    </getPerson>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Locate the `<Query>` element of the request, and replace "String" with the value you want to supply as input to the Web service. As stated before, this Web service retrieves people details from an Access database. For the scope of this example, in order for the call to actually retrieve some data, “String” must be replaced with “Ro” (in other words, only people whose first or last name contains "Ro" would be retrieved). The new request now looks as follows:
<?xml version="1.0" encoding="UTF-8"?>
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <getPerson SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <Query xsi:type="xsd:string">Ro</Query>
    </getPerson>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

You can now send the request above to the server. On the SOAP menu, click Send Request to Server. The SOAP Request Settings dialog box opens.

Click OK. XMLSpy initiates the call, and returns the response in the editor. The image below illustrates a successful response.
Using the same approach as described above, you can also call the **putPerson** operation exposed by this Web service. For step-by-step instructions, see "Calling the Web Service" section in the Java tutorial. Only the WSDL URL is different in the Java tutorial; all other instructions for calling the Web service are identical.

**Troubleshooting**

The following table lists common problems that you might encounter when building or invoking the Web service, and their solution.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calling the Web service returns the following error:</td>
<td>1. Build the C# solution in Visual Studio as a 32-bit application.</td>
</tr>
<tr>
<td></td>
<td>2. Deploy to IIS.</td>
</tr>
<tr>
<td></td>
<td>3. Right-click the IIS application pool where you deployed the application, and select <strong>Advanced Settings</strong>.</td>
</tr>
<tr>
<td></td>
<td>4. Under <strong>Enable 32-bit applications</strong>, select <strong>True</strong>.</td>
</tr>
<tr>
<td></td>
<td>• Grant to the IIS application pool permission to read the Access database file, see &quot;Step 4: Configure the application in IIS Manager&quot; above.</td>
</tr>
<tr>
<td>Calling the Web service returns the following error:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Problem

\<user>\Documents\Altova\MapForce2019\MapForceExamples\Tutorial\people.mdb'. It is already opened exclusively by another user, or you need permission to view its data.

### Solution

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Calling the Web service returns the following error:</strong></td>
<td>• Grant to the IIS application pool permission to write the Access database file, see &quot;Step 4: Configure the application in IIS Manager&quot; above.</td>
</tr>
</tbody>
</table>
9.6 Example: Generating SOAP Web Services (Java)

This example illustrates how to generate a SOAP Web service with MapForce. You will generate Java program code from a sample MapForce project available at the following path: <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\Query Person database.mfp. This project implements a SOAP Web service that exposes two operations: `getPerson` and `putPerson`. This Web service communicates in the background with a Microsoft Access database to retrieve or add person details, respectively, through the operations above. For an example of how to create such projects in MapForce, see Creating Web Service Projects from WSDL Files.

This example requires Java 7 and Apache Ant to compile the generated Java code. The .aar file will be deployed to a running Apache Tomcat server with Axis2 installed.

For simplicity, this example uses a minimum Web server configuration and should not be taken as prescriptive. In a production environment, you may need to use a different approach as demanded by the specifics of your Web server, the security policies in your organization, and other factors.

To call the SOAP Web service, this example makes use of XMLSpy Enterprise Edition. If you do not have XMLSpy, you will need a SOAP client to test the Web service, or, alternatively, write program code that can call the Web service.

**Note:** As stated before, this Web service example accesses a Microsoft Access database. With Java 7, connectivity to Access databases is possible through the ODBC-JDBC bridge included in Java. However, the ODBC-JDBC bridge has been removed in Java 8, and Oracle recommends using JDBC drivers provided by the vendor, see http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/bridge.html. Therefore, to run this example in a Java 8 or newer environment, you will need to find and install third-party JDBC drivers that can connect to a Microsoft Access database.

### Step 1: Prepare the query.wsdl file

This example Web service originates from the following WSDL file: <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\query.wsdl. By default, the `query.wsdl` file defines the service details for C# and Java in two separate sections, one of which is commented out. Therefore, before generating Java code, locate the following section in the `query.wsdl` file and make sure that it is commented out:

```xml
<!---service name="WS2DB"

<port name="WS2DBSoapPort" binding="tns:WS2DBSoapBinding">
  <soap:address location="http://localhost/WS2DB/WS2DB.asmx"/>
</port>
</service-->
```

Also, make sure that the following section (only applicable to Java) is NOT commented out:

```xml
<!--service name="WS2DB"

<port name="WS2DBSoapPort" binding="tns:WS2DBSoapBinding">
  <soap:address location="http://localhost/WS2DB/WS2DB.asmx"/>
</port>
</service-->
```
Step 2: Generate Java code and build it

Run MapForce and open the "Query Person database.mfp" project from the folder `<Documents>` \Altova\MapForce2019\MapForceExamples\Tutorial\. Right-click the project in the Project window, and select Generate code in | Java. MapForce generates program code and displays the outcome in the Messages window:

By default, code will be generated in a subdirectory called "output", relative to the project's directory. For the scope of this tutorial, we will leave the default settings as is. However, if you want to change the output directory, see Setting the Code Generation Settings.

To build the Java code, run Ant at the command line in the same directory as the build.xml file.

As a result, an Axis Archive (.aar) file called Query_Person_database.aar is generated in the output directory.
Before you can call Ant from any directory as illustrated above, it must be installed and configured correctly. Here is a quick summary (for more details, check the Ant manual from http://ant.apache.org):

- The `PATH` environment variable must include the path to the "bin" subdirectory of Ant root directory.
- The `ANT_HOME` environment variable must be set to the directory where Ant is installed.
- The `JAVA_HOME` environment variable must be set to the directory where Java JDK is installed.
- Check that Ant is running by typing "ant" at the command line. The message "Buildfile: build.xml does not exist!" means that Ant is configured and is looking for a build file.

**Step 3: Deploy the Web service to Axis 2**

In this example, Axis2 v1.7.6 runs on Apache Tomcat 8.5.0 installed on a local machine. In brief, Axis2 can be installed and deployed to a running Tomcat server as follows (for more details, refer to Axis2 documentation):

2. Unpack the downloaded package to a local directory.
3. Copy the `axis2.war` file to Tomcat's directory `%CATALINA_HOME%`/webapps.
4. Check that Axis2 is running, by opening: http://localhost:8080/axis2

Open the Axis2 Web administration page and upload the .aar file generated in the previous step.
A new Web service called "WS2DB" should now be available and active in the Axis2 administration page.

**Available services**

**WS2DB**

Service Description : WS2DB

Service EPR : http://localhost:8080/axis2/services/WS2DB

Service Status : Active

**Available Operations**

- getPerson
- putPerson

**Calling the Web service**

If you followed the steps above literally, then the WSDL of the Web service is available at http://
localhost:8080/axis2/services/WS2DB?wsdl. To call the Web service we just deployed, we will use XMLSpy, although you can use, of course, other SOAP clients as well.


Click OK. When prompted to select a SOAP Operation, select putPerson.

XMLSpy generates a sample request like the one below.
<?xml version="1.0" encoding="UTF-8"?>
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:m0="http://www.altova.com/WS2DB.xsd">
  <SOAP-ENV:Body>
    <putPerson SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <Put xsi:type="m0:Person">
        <First xsi:type="xsd:string">String</First>
        <Last xsi:type="xsd:string">String</Last>
        <Title xsi:type="xsd:string">String</Title>
        <PhoneEXT xsi:type="xsd:string">String</PhoneEXT>
        <Email xsi:type="xsd:string">String</Email>
      </Put>
    </putPerson>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Replace "String" with the person details you want to supply as input to the Web service. For the scope of this example, you could use a request like the one below:

<?xml version="1.0" encoding="UTF-8"?>
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:m0="http://www.altova.com/WS2DB.xsd">
  <SOAP-ENV:Body>
    <putPerson SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <Put xsi:type="m0:Person">
        <First xsi:type="xsd:string">John</First>
        <Last xsi:type="xsd:string">Doe</Last>
        <Title xsi:type="xsd:string">Mr</Title>
        <PhoneEXT xsi:type="xsd:string">123</PhoneEXT>
        <Email xsi:type="xsd:string">john.doe@example.org</Email>
      </Put>
    </putPerson>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

You can now send the request above to the server. On the SOAP menu, click **Send Request to Server**. The SOAP Request Settings dialog box opens.
Click OK. XMLSpy initiates the call, and returns the response in the editor. The image below illustrates a successful response.
Using the same approach as described above, you can also call the `getPerson` operation exposed by this Web service. For step-by-step instructions, see “Calling the Web Service” section in the C# tutorial. Only the WSDL URL is different in the C# tutorial; all other instructions for calling the Web service are identical.
Chapter 10

Calling Web Services
10 Calling Web Services

You can design a mapping so that it calls a Web service as part of its execution. Namely, you can insert a Web service call into a mapping similar to a function, connect input and output components to it, and consume the result as required (for example, preview it in the MapForce output window, pass it to another component, or write it to a file). This effectively turns MapForce into a powerful Web service client which is easily configurable from a graphical user interface. A Web service call created with MapForce may be executed either by MapForce itself, or on a different machine or even platform, by MapForce Server. The latter can run mappings through the command-line interface, as an API call, or as a FlowForce Server job, see Automation with MapForce Server.

MapForce supports calling both WSDL-style and REST-style Web services. Therefore, when you add a Web service to the mapping, you can use one of the following approaches:

- For WSDL-style Web services, provide to MapForce the WSDL (Web Service Definition Language) file of the Web service to be called. MapForce uses the .wsdl file to communicate with the Web server. If the .wsdl file implements multiple services, endpoints, and operations, you can select or update them directly in MapForce;
- For non-WSDL Web services, manually enter into MapForce the Web service details. This includes the URL, the request method (for example, GET, POST, PUT), request and response structure (such as XML, JSON, or custom MIME types), and parameters. Optionally, if you have the WADL (Web Application Definition Language) file of the Web service to be called, you can import the definition from the WADL file. Also, if you have a sample URL of the Web service, you can instruct MapForce to parse the URL and automatically extract any query, template, or matrix parameters from it, so that you don't have to define them manually.

The following table summarizes the support details of REST-style and WSDL-style Web services in MapForce.

<table>
<thead>
<tr>
<th>Feature</th>
<th>REST-Style</th>
<th>WSDL-Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping language</td>
<td>BUILT-IN</td>
<td>BUILT-IN, C#, Java</td>
</tr>
<tr>
<td>Automation with MapForce Server</td>
<td>Yes</td>
<td>Yes, when language is &quot;BUILT-IN&quot;</td>
</tr>
<tr>
<td>Protocols</td>
<td>HTTP (GET, POST, PUT, DELETE, custom verbs)</td>
<td>• SOAP 1.1, 1.2 over HTTP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Non-SOAP services over HTTP</td>
</tr>
<tr>
<td>Request/response structures</td>
<td>• XML</td>
<td>SOAP message</td>
</tr>
<tr>
<td></td>
<td>• JSON</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Protocol Buffers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Custom MIME types</td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td>• HTTP/HTTPS</td>
<td>• HTTP/HTTPS</td>
</tr>
<tr>
<td></td>
<td>• Server certificates</td>
<td>• Server certificates</td>
</tr>
<tr>
<td></td>
<td>• Client certificates</td>
<td>• Client certificates*</td>
</tr>
<tr>
<td>Feature</td>
<td>REST-Style</td>
<td>WSDL-Style</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>• HTTP authentication</td>
<td>• HTTP authentication</td>
</tr>
<tr>
<td></td>
<td>• Preemptive authentication</td>
<td>• Preemptive authentication*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• WS-Security*</td>
</tr>
<tr>
<td>Import service definition</td>
<td>Optionally, from WADL file</td>
<td>From WSDL file (required)</td>
</tr>
<tr>
<td>Import parameters from URL</td>
<td>Yes</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Dynamic URL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Timeout</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Dynamic authentication</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Remarks:

- "Dynamic URL" means that you can provide the URL of the Web service from the mapping (or as a parameter to the mapping) as an alternative to entering it in a dialog box and saving it together with the mapping.
- "Timeout" means that you can configure in MapForce the interval after which the call should time out if the Web server is not responding.
- "Dynamic authentication" means that you can provide the authentication details (username, password) from the mapping (or as a parameter to the mapping) as an alternative to entering them in a dialog box and saving them together with the mapping.

In the case of WSDL-style Web services, you can use the following protocols:

- SOAP 1.1, 1.2 over HTTP
  - Both the RPC/Encoded and Document/Literal styles are supported. If the Web server returns a WSDL fault, the mapping execution stops. For such cases, you can optionally insert an exception component on the mapping area to handle the error. If the Web server returns a non-WSDL error, the mapping execution stops, and an error message is returned (or displayed on the screen, if you are previewing the mapping in MapForce).

- Non-SOAP over HTTP
  - This refers to slightly less common, non-SOAP HTTP services. When using HTTP GET, the url-encoded style is supported. When using HTTP POST, the url-encoded and text/xml styles are supported.
10.1 Adding a Web Service Call (REST-Style)

Generic (not WSDL-style) Web services include a large category of Web services that follow or partially follow an architectural style referred to as "REST" (and are typically called "RESTful", or REST-style Web services). Generic HTTP Web services typically carry custom request or response structures in the message body part. MapForce supports the following kinds of request or response bodies: JSON, XML, Protocol Buffers, and unstructured bodies carrying custom MIME types. Therefore, from a MapForce perspective, the prerequisites to call a Web service depend on the type of the request:

- In case of XML or JSON requests or responses, MapForce needs the request or response structure as a JSON, XML, or DTD schema. MapForce will also accept as structure an XML file with a valid schema reference. The structure of the Web service could be published by the service provider as XML or JSON schema, or through a formal language such as WADL, or even be a human-readable specification. If you already have the XML or JSON schema of the request/response, you will need the least amount of effort to create the Web service call. If you have the Web service definition as a WADL file, you can import it from the WADL file, and make any potential adjustments manually. Note, however, that WADL provides no standard way to define JSON structures, only XML structures. Finally, if you have a sample XML or JSON instance file but don't have a schema file, you can either create or generate the schema with X MLSpy (https://www.altova.com/xmlspy.html). If necessary, XMLSpy can also convert your instance file from XML to JSON, or vice versa.

- In case of Protocol Buffers requests or responses, you need the .proto file that describes the Protocol Buffers binary file. In this scenario, the body of the Web service can be mapped to or from a Protocol Buffers component. For more information, see Example: Read Data from Protocol Buffers and Example: Write Data to Protocol Buffers.

- You can also call Web services where the request or response structure is flexible and not tied to a particular schema. For such cases, you can use MapForce built-in mime functions to either create the raw message body sent to the Web service (the MIME entity) or process from the mapping the MIME entity returned by the Web service.

Adding a call to a generic Web service

1. On the Insert menu, click Web Service Function. (Alternatively, click the Insert Web service function toolbar button).
3. Optionally, if you have the WADL file describing the service, click **Import from WADL file** and select the file, see also [Importing Web Service Information from WADL](#).

4. Select the HTTP request method that MapForce should use to call the Web service. You can either select a value from the existing list, or type the name of the request method. The HTTP method names are case-sensitive.

5. Enter the URL of the Web service. If the URL to the Web service uses parameters, note the following:
   a. If you are calling a Web service with "template" or "matrix" style parameters, enclose the parameters within curly braces, for example: `http://example.org/api/products/{id}`. Then define the actual settings of each parameter in the "Parameters" table. At runtime, MapForce processes the parameter names in curly braces and produces the final URL which includes actual values.
   b. If you are calling a Web service with "query" URL parameters (for example, `http://example.org/api/products?sort=asc&category=1&page=1`), do not enter the query part in the URL text box. Instead, define the parameters only in the "Parameters" table.

   For examples, see [Defining Web Service Parameters](#).

6. Optionally, under **Timeout**, enter a period in seconds after which the connection should time out if the server is not responding.

7. If the HTTP method requires or returns a body part (such as XML, JSON, and others), click the **Edit** button under **Structure** and browse for the schema of the body part. For
more information, see Defining the Request Structure and Defining the Response Structure.

8. Under Parameters, define the parameters of the Web service. Optionally, click Import from URL to import the parameters from a sample URL of the Web service and populate the "Parameters" table automatically, as shown in Importing Web Service Parameters from URL. After importing parameters from a URL, you can modify the contents of the "Parameters" table if necessary.

Note: To specify custom request headers, add a parameter with style "Header", where the parameter name corresponds to the header name, and the parameter value corresponds to the header value. Furthermore, if you need to provide the value of the request header from the mapping itself, set the parameter type to "Mappable". For further information, see Defining Web Service Parameters.

9. If the Web service uses HTTP authentication or certificate-based security, click the Edit button under HTTP Security Settings and fill in the required fields, see also Setting HTTP Security.

After you click OK, a new Web service component is added to the mapping area. For example, the mapping below calls a Web service in order to retrieve a product by its ID using a GET request. In this example, the ID supplied in the HTTP request has the constant value "2". However, it could also be a parameter to the mapping, or be supplied by any component supported by MapForce. In addition to the id parameter, the request contains the header Accept: text/json. You can set the request headers by means of "header" parameters, as shown in Defining Web Service Parameters.

The mapping illustrated above has been configured to have three outputs:

1. HTTP status code returned by the Web service call
2. The response headers as comma-separated values
3. The response JSON structure

This is only for demo purposes; you could also design a mapping so that it has only one output, for example, the HTTP status code. When a mapping has multiple outputs like the one above, you can click the Preview button of each component to preview that particular output before running the mapping.
Notice that the Web service component consists of two parts: Request and Response. The Request part enables you to supply data from the mapping to the Web service, while the Response part enables you to access the data returned by the Web service and map it to other formats. The structure of the request and response depends on the parameters, as well as the request or response structure you have defined by clicking the button. In the Web service call above, the structure of the request body is not set, while the structure of the response body is set as JSON—which makes it possible to map the result of the Web service to an actual JSON file.

To call the Web service with request parameters (if applicable), draw mapping connections between any component supported by MapForce (for example, an XML or JSON file) and the Request part. Likewise, to map data returned by the Web service to another format, draw mapping connections between the Response part and any other component type supported by MapForce. If you are new to MapForce and need instructions about drawing mapping connections, see Working with Connections.

The response headers returned by the Web service are also mappable, if they are additional headers (the ones which do not begin with "Content"). The header values are available on the Web service component through an item called "Headers" which contains two child items: "Name" and "Value". This structure acts as sequence, and enables you to map data from any number of headers returned by the Web service. To map the data from the response headers to any other format supported by MapForce, connect the "Headers" structure node to a target sequence in the mapping. For example, if you connect the Headers sequence (and its children) to a Rows sequence (and its children) of a CSV component, one header would correspond to one row in the CSV file.

The Body represents the entity body of the HTTP message. Data at this level is binary encoded, so interacting with it directly requires MapForce mime functions. Note that you don't typically need to interact with the Body at all if the Web service expects or returns structured data such as XML or JSON. Mapping data directly to or from the Body is only necessary when calling Web services that expect or return unstructured content.

By default, the Body item is configured to parse the result if the HTTP status code ranges from 200 through 299. This means that the mapping will return an error for status codes greater than 299. Also, an error will be returned if the response cannot be parsed or the Web service cannot be called due to connection failure or DNS resolution problems.

In some cases, you might want the mapping not to throw an error even if the HTTP status code is greater than 299. To do this, click the button next to the response Body and change the HTTP status code range. Alternatively, you can create multiple response Body items. Creating multiple Body items is useful when you want to handle the mapping conditionally depending on the HTTP status code returned by the Web service. To do this, right-click the Body of the response part, and select Add Body Node Before/After from the context menu. For example, you could create two Body items:

1. A Body for all status codes that are in the success range (200 to 299). You can configure the status codes range by clicking the button. For this scenario, you will typically want to map the Body structure to some "success" output, like a JSON or XML file.
2. A Body for all other status codes, in the range between 300 and 599. Some Web services may provide additional details about error status codes in the body. To find out the reason of the error, you can therefore map the "erroneous" Body to an output other
than the "success" one (for example, a string).

When multiple **Body** items exist, MapForce evaluates them from top to bottom. To change the priority, right-click the **Body** item, and select **Move Up** or **Move Down** from the context menu. For more information about handling the HTTP response conditionally, see [Handling the HTTP Response Conditionally](#).

Mappings containing generic HTTP Web service calls can be executed like most other mappings, namely:

- Manually, with MapForce, by clicking the **Output** tab. In this case, the result of the mapping call is immediately available in the **Output** pane. If the mapping has multiple output components like in the example above, press the **Preview** button on the desired component before running the mapping.
- Through command line or API calls, with MapForce Server ([https://www.altova.com/mapforce-server](https://www.altova.com/mapforce-server)). This requires compiling the mapping to a mapping execution file first (see [Compiling Mappings to MapForce Server Execution Files](#)).
- As a recurring job, with MapForce Server running under FlowForce Server control ([https://www.altova.com/flowforceserver](https://www.altova.com/flowforceserver)). See also [Deploying Mappings to FlowForce Server](#).

For step-by-step examples of how to create a generic Web service call, see:

- [Example: Calling a REST-Style Web Service](#)
- [Example: Mapping Data from an RSS Feed](#)

### 10.1.1 Defining Web Service Parameters

When you need to call a Web service with URL parameters, the parameters must be explicitly defined on the Web Service Call Settings dialog box. This dialog box also lets you configure the Web service request with custom headers.

**To define Web service parameters:**

1. Double-click the title bar of the Web service component on the mapping, or right-click it and select **Properties**. The Web Service Call Settings dialog box opens.
2. Under **Service Definition**, select **Manual**. As illustrated above, you can enter the parameters manually, or, as an alternative, you can import them from an existing URL or from a WADL file. The imported parameters become available in the **Parameters** table, where you can further modify them if required.

3. Use the **Add Parameter** (➕) and **Delete Parameter** (➖) buttons, respectively.

The columns in the **Parameters** table have the following meaning:

<table>
<thead>
<tr>
<th>Name</th>
<th>Specifies the name of the URL parameter. The parameter name must be unique and may consist of letters, digits, periods (.), hyphens (-) and underscores (_). No spaces are allowed in the parameter name.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Style</td>
<td>Specifies the syntax (style) of the URL parameter.</td>
</tr>
<tr>
<td></td>
<td>Use the &quot;Header&quot; style to add a parameter to the HTTP header when calling the Web service. For example, adding a parameter <strong>Accept</strong> with value <strong>text/xml</strong> is equivalent to specifying the <strong>Accept: text/xml</strong> header, which informs the Web server that MapForce expects the response to be in XML format. For more information about HTTP headers, see <a href="https://www.iana.org/assignments/message-headers/message-headers.xhtml">https://www.iana.org/assignments/message-headers/message-headers.xhtml</a>.</td>
</tr>
<tr>
<td></td>
<td>Use the &quot;Query&quot; style for URL parameters that define key-value pairs using the format: <strong>key=value&amp;key=value</strong></td>
</tr>
<tr>
<td></td>
<td>For example: <a href="http://example.org/api/products?sort=asc&amp;category=1&amp;page=1">http://example.org/api/products?sort=asc&amp;category=1&amp;page=1</a></td>
</tr>
<tr>
<td></td>
<td>Use the &quot;Template&quot; style for URL parameters enclosed within curly brackets, for example: <strong>{id}</strong></td>
</tr>
</tbody>
</table>
### Types of Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Specifies the data type of the parameter (string, integer, date, etc). This can be any XML schema type. Note that any value that is not a string is converted to a string when the Web service call takes place. Nevertheless, setting a type is meaningful if you want MapForce to show conversion error messages when you attempt to call a Web service with wrong values.</td>
</tr>
<tr>
<td><strong>Mappable</strong></td>
<td>Select this check box if you want to pass values to this parameter from the mapping. This option is mutually exclusive with the &quot;Fixed Value&quot; option.</td>
</tr>
<tr>
<td><strong>Fixed Value</strong></td>
<td>Specifies the value of the parameter. Applicable only if the parameter has a constant value. Not applicable if the parameter is mappable (see previous option).</td>
</tr>
<tr>
<td><strong>Required</strong></td>
<td>Select this check box if the parameter is required by the Web service. For parameters that are required and also mappable, MapForce enforces validation checks (that is, an error message is displayed if the parameter does not have a value).</td>
</tr>
<tr>
<td><strong>Repeating</strong></td>
<td>Specifies whether the parameter is single-valued or may have multiple values. This setting is applicable only for mappable parameters. It enables you to pass multiple values in the same Web service call, by means of a single parameter. When you select the check box, you can connect a sequence of values to the parameter structure node on the mapping, instead of a single value. MapForce will then handle the sequence of values depending on the style of the parameter, as follows:</td>
</tr>
<tr>
<td></td>
<td>• For &quot;Template&quot; parameters, the values will be supplied to the Web service as comma-separated, for example: <a href="http://example.org/api/products/1,2,3">http://example.org/api/products/1,2,3</a></td>
</tr>
<tr>
<td></td>
<td>• For &quot;Query&quot; parameters, the parameter name will be repeated for each value, for example: <a href="http://example.org/api/products?color=red&amp;color=green&amp;color=blue">http://example.org/api/products?color=red&amp;color=green&amp;color=blue</a></td>
</tr>
<tr>
<td></td>
<td>• If the style is &quot;Matrix&quot;, multiple values will be separated by comma, for example: <a href="http://example.org/api/products;color=red;color=green;color=blue;size=big;size=small">http://example.org/api/products;color=red;color=green;color=blue;size=big;size=small</a></td>
</tr>
<tr>
<td></td>
<td>• If the style is &quot;Header&quot;, the HTTP header will be repeated for each value.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Specifies the optional description of the parameter. If the parameter is</td>
</tr>
</tbody>
</table>

---

**Example:**

```
http://example.org/api/products/\{id\} . For such parameters, MapForce escapes the values according to the RFC 6570 rules (https://tools.ietf.org/html/rfc6570).
```

Use the "Matrix" style for URL parameters that define key-value pairs in the format: `key=value;key=value;` For example:

```
http://example.org/api/products;sort=asc;category=1;page=1;
```

To use Boolean matrix parameters, set the style to "Matrix" and the type to "boolean" (see also next option).
mappable, the description entered here appears on the mapping component as an annotation next to the mapping item.

Example 1: Header and template parameters
The Web service illustrated below retrieves a product by its identifier (id) using the HTTP GET method. The URL of the Web service specifies the id parameter in the curly brackets. Notice that the id parameter exists in the Parameters table and has the style “Template”. It is also mappable: this causes the parameter to appear on the mapping as a structure node to which you can connect the actual value of id (which could be, for example, taken from a database, a file, or a constant). At mapping execution runtime, this parameter would be replaced with the actual value; so, if the value is “1”, the URL becomes http://example.org/api/products/1.

To supply a constant id value instead of a value from the mapping, clear the check box “Mappable” and enter the value in the “Fixed value” column.

The parameters Accept and Accept-Charset have the “Header” style. These parameters are used to call the Web service with custom request headers. There are two ways to supply the header value:

- Leave the option Mappable checked and supply the custom header value from the mapping, or
- Uncheck the option Mappable and enter the value directly in the “Fixed value” column.
Example 2: Matrix parameters

The Web service illustrated below retrieves a list of products that match the color and size supplied as arguments. The style of the parameters is "Matrix", so they are defined both as placeholders inside the URL and in the mapping table. Notice that the parameters are mappable and the "Repeating" option is checked. This means that their value will be read from some sequence of values on the mapping (for example, a list of rows inside a text file, an XML node, or a database column) and supplied to the Web service at runtime. Thus, a URL such as the one below would become http://example.org/api/products/;color=red;color=blue;size=big;size=small, provided that the mapping supplies red and blue as colors, and big and small as size.

![Connection Settings](Image)

If you need the URL to be http://example.org/api/products;color=red,blue;size=big,small, do the following:

1. Enter the URL as http://example.org/api/products;color={color};size={size}
2. In the "Parameters" table, change the parameter style to "Template".

If you need the URL to be http://example.org/api/products?color=red,blue&size=big,small, do the following:

1. Enter the URL as http://example.org/api/products?color={color}&size={size}
2. In the "Parameters" table, change the parameter style to "Template".

Example 3: Query parameters

The Web service illustrated below also retrieves a list of products that match the color and size supplied as arguments, this time using the style "Query". For this style, it is not necessary to define the parameters as placeholders in the URL, so they are defined only in the "Parameters" table. In this example, let's assume that the parameter values are fixed and must not be supplied from the mapping. To achieve this, the parameter values were entered under "Fixed Values", and the "Mappable" option is unchecked. Thus, at mapping runtime, the URL below would become http://example.org/api/products?color=red&size=big.
10.1.2 Importing Web Service Parameters from URL

If the URL to the Web service uses parameters, you can instruct MapForce to parse the URL and import any parameters automatically. When you use this option, any template, matrix, or query parameters extracted from the URL become available in the Parameters table of the Web Service Call Settings dialog box, where you can further manipulate them as necessary.

Example of a URL with template parameters: http://example.org/api/products/{id}
Example of a URL with matrix parameters: http://example.org/api/products;sort=asc;category=1;page=1
Example of a URL with query parameters: http://example.org/api/products?category=1&page=2&sort=asc

Note: A URL must begin with either "http://" or "https://" to be parseable.

To import Web service parameters from a URL:

1. Open the Web Service Call Settings dialog box (see Web Service Call Settings).
3. Click Import from URL.
4. Enter or paste the URL to the Web service in the text box, and click OK.
10.1.3 Importing Web Service Information from WADL

The Web Application Description Language (WADL) is one of the ways to represent the contents of the responses and requests used in a web application, including RESTful Web services. MapForce supports importing Web service information from WADL version 2009 (https://www.w3.org/Submission/2009/SUBM-wadl-20090831/).

If the Web service has a WADL file, you can import the Web service definition from the WADL file, rather than entering it manually. When you import information from a WADL file, the parameters and the XML message body structure are populated automatically on the Web Service Call Settings dialog box. After the import, you can modify the parameters and the response/request structure manually, if necessary. This will not affect the underlying WADL file.

To import Web service information from a WADL file:

1. Open the Web Service Call Settings dialog box (see Web Service Call Settings).
3. Click Import from WADL file and browse for the WADL file.
4. On the dialog box, select the resource and method, and click OK.
10.1.4 Defining the Request Structure

When calling Web services that expect content in the body part of the request message, you can set the request structure to be one of the following:

- No structure
- XML
- JSON
- Protocol Buffers files, see Protocol Buffers

The structure selected here determines the mappable items that appear on the mapping as children of the Body item.

To define the request structure:

1. Open the Web Service Call Settings dialog box (see Web Service Call Settings).
4. Select one of the following options:
   a) If the Web service expects no content in the body part, or if you would like to provide the raw body (MIME entity) directly from the mapping, select No structure.
   b) For XML, obtain the XML or DTD schema of the request from the provider of the Web service. If you select an XML instance file, it must have a valid schema reference. Some XML schemas define elements with global declaration (that is, elements whose parent is the schema element). For such schemas, you can choose what element in the schema should be the root element of the mapping structure in MapForce. To do this, click...
Choose, and then, in the dialog box that appears, select the desired root element.

c) For JSON, obtain the JSON schema of the request from the provider of the Web service, or generate it from a sample JSON file with a tool such as XMLSpy.

Tip: If you have a sample XML or JSON request file but don’t have a schema file, you can use XMLSpy to generate the schema file. If necessary, XMLSpy can also convert your instance file from XML to JSON, or vice versa.

b) For Protocol Buffers files, obtain from the Web service provider the .proto file describing the binary file, and then click Browse to select the file. The Root Message Type field is applicable if the Protocol Buffers file contains multiple message types. Click Choose to select the message type that should be at the structure’s root.

5. Optionally, select the Override content type check box if you need to override the Content-Type header set implicitly by MapForce. You can also enter parameters after the content type, for example: \texttt{Content-Type: text/html; charset=utf-8}.

When you save the mapping, any absolute paths on this dialog box will become relative to the mapping design file (.mfd) if the check box Save all file paths relative to mfd is selected in component settings, see Web Service Call Settings.

### 10.1.5 Defining the Response Structure

When calling Web services that return content the body part of the response message, you can set the response structure to be one of the following:

- No structure
- XML
- JSON
- Protocol Buffers files, see Protocol Buffers

The structure selected here determines the mappable items that appear on the mapping as children of the Body item.

To define the response structure:

1. Open the Web Service Call Settings dialog box (see Web Service Call Settings).
Tip: As a shortcut to display this dialog box, click the button in the request part of the Web service component on the mapping.

4. Select one of the following options:

a) If the Web service returns no content in the body part, or if you would like to process the raw body (MIME entity) directly from the mapping, select No structure.

b) For XML, obtain the XML or DTD schema of the request from the provider of the Web service. If you select an XML instance file, it must have a valid schema reference. Some XML schemas define elements with global declaration (that is, elements whose parent is the schema element). For such schemas, you can choose what element in the schema should be the root element of the mapping structure in MapForce. To do this, click Choose, and then, in the dialog box that appears, select the desired root element.

c) For JSON, obtain the JSON schema of the request from the provider of the Web service, or generate it from a sample JSON file with a tool such as XMLSpy.

Tip: If you have a sample XML or JSON request file but don't have a schema file, you can use XMLSpy to generate the schema file. If necessary, XMLSpy can also convert your instance file from XML to JSON, or vice versa.

b) For Protocol Buffers files, obtain from the Web service provider the .proto file describing the binary file, and then click Browse to select the file. The Root Message Type field is applicable if the Protocol Buffers file contains multiple message types. Click Choose to select the message type that should be at the structure's root.

Optionally, fill the Content type and HTTP status text boxes if you need to filter a response MIME entity by media (content) type and HTTP status code. This is useful if you need conditional handling depending on the HTTP status code and the content type returned by the Web service call. For an example, see Handling the HTTP Response Conditionally.

By default, the Content type filter is set to */*, which matches all media types. The default
HTTP status codes are 200 through 299. If the Web service call returns some other status code that isn't handled explicitly, the mapping execution will fail with an error.

When you save the mapping, any absolute paths on this dialog box will become relative to the mapping design file (.mfd) if the check box Save all file paths relative to mfd is selected in component settings, see Web Service Call Settings.

### 10.1.6 Handling the HTTP Response Conditionally

Let's assume that a Web service returns either JSON or XML content type in the response body, depending on the value of the Accept HTTP request header. Additionally, in case of an error (status code > 299), the Web service may return an undefined content type. This is a case where it might be useful to handle the response conditionally based on the HTTP status code and content type.

To handle both the JSON and the XML response from the Web service in the same mapping, you could create multiple body items, by right-clicking the body item on the component and selecting Add Body Node After from the context menu.

On the mapping, a Web service component with multiple body items configured for the scenario above could look as follows:

The mapping illustrated above runs takes two input parameters. The first parameter provides the expected content type and its value can be application/json, or application/xml. The second parameter provides the numeric ID of the product to be retrieved.

If you provide the value application/json as input parameter, click the Preview button of the JSON target component. Otherwise, if the input parameter value is application/xml, click the Preview button of the XML target component. The respective output file (JSON or XML) will be generated if the Web service call is successful.

If the Web service call returns a status code greater than 300 (error), the HTTP response reaches the third body item. Since we don't know the exact content type, we choose to decode it with the help of the decode-mime-entity function. The Web service does not supply the character set value in the response header, which is why the constant value utf-8 is connected to the decoding function.

Each of the three response body items on the component above are configurable by clicking the
button next to each body:

1. The first body handles content returned as application/json if the status code is between 200 and 299. Also, the schema of the expected JSON response structure is set.

![Response Structure](image1.png)

2. The second body handles content returned as application/xml if the status code is between 200 and 299. Also, the schema of the expected XML response structure is set.

![Response Structure](image2.png)

3. The third body handles content returned as */* if the Web service returns an erroneous status code (between 300 and 599).
### 10.2 Adding a Web Service Call (SOAP)

This topic describes how to add to a mapping a call to a WSDL-style Web service. For instructions on how to add a generic Web service to the mapping, see [Adding a Web Service Call (REST-Style)](#).

Before adding a WSDL-style Web service call to the mapping area, make sure that you have the Web Service Definition Language (.wsdl) that describes the Web service to which you are attempting to connect.

**To add a call to a WSDL-style Web service:**

1. On the **Insert** menu, click **Web Service Function**. (Alternatively, click the **Insert Web service function** toolbar button).
2. Under **Service Definition**, click **WSDL** if this option is not already selected.
3. Under **WSDL Settings**, click **Browse** and select the Web Service Definition Language (.wsdl) file. Two sample .wsdl files are available in the MapForce Example folder, one for each version of WSDL (1.0 and 2.0). The corresponding paths are as follows: `<Documents>\Altova\MapForce2019\MapForceExamples\TimeService\TimeService.wsdl` and `<Documents>\Altova\MapForce2019\MapForceExamples\TimeServiceWsdl2\TimeService20.wsdl`.
4. If the .wsdl file defines multiple services, operations, and endpoints, click **Choose** and select the required options.
5. If necessary, define other settings as required (see [Web Service Call Settings](#)).
6. Click **OK**.

See also [Example: Calling a SOAP Web Service](#).
10.3 Web Service Call Settings

You can change the settings applicable to Web services from the Web Service Call Settings dialog box. To open this dialog box, do one of the following:

- Click the Web service component on the mapping, and, then, on the Component menu, click Properties.
- Double-click the component header.
- Right-click the component header, and then click Properties.

The list of available settings depends on whether the Service definition option at the top of the dialog box is set to WSDL or Manual. The following image illustrates the settings applicable for a generic HTTP Web service:

The following image illustrates the settings applicable for a WSDL-style Web service:
The available settings are described below.

**Service Definition**
If the settings you define apply to a WSDL-style Web service, select **WSDL**. Otherwise, select **Manual**.

When **Manual** is selected, you must enter the Web service settings manually into the dialog box. Optionally, if you have the WADL file of the Web service, you can import the Web service settings by clicking **Import from WADL** (see Importing Web Service Information from WADL). Also, if you want to extract the Web service parameters from a URL, click the **Import from URL** button (see Importing Web Service Parameters from URL).

**WSDL Settings**
This group of settings is applicable only for WSDL-style Web services.

| **WSDL Definitions** | Specifies the Web Service Definition Language (WSDL) file of the web service to be called. |
### Calling Web Services

#### Web Service Call Settings

<table>
<thead>
<tr>
<th><strong>Service</strong></th>
<th>Specifies the name of the Web service to be called. If the WSDL file defines multiple web services, click <strong>Choose</strong> to select the required one.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Endpoint</strong></td>
<td>Specifies the endpoint (or port) of the Web service to be called. If the selected Web service defines multiple web service endpoints, click <strong>Choose</strong> to select the required one.</td>
</tr>
<tr>
<td><strong>Operation</strong></td>
<td>Specifies the operation of the Web service to be called. If the selected endpoint defines multiple web service operations, click <strong>Choose</strong> to select the required one.</td>
</tr>
</tbody>
</table>

#### Request Method
This setting is applicable only for generic (not WSDL) Web services. It defines the HTTP method (verb) used by the service (for example, GET, POST, PUT, and so on).

#### Connection Settings
URL specifies the address (URL) of the Web service. Timeout defines the time interval after which the Web service call will time out if there is no response from the server. Select Infinite if the call should wait for a response for an indefinite amount of time.

If you would like to supply the URL of the Web service from the mapping, select the **Dynamic URL (supplied by mapping)** check box. This creates an additional input connector on the Web service component on the mapping, to which you can connect some input which provides the URL of the Web service.

#### Parameters
This group of settings is applicable only for generic (not WSDL) Web services. The **Parameters** table specifies the URL parameters with which MapForce will call the Web service (see Defining Web Service Parameters).

#### Structures
This group of settings is applicable only for generic (not WSDL) Web services. For more information, see Defining the Request Structure and Defining the Response Structure.

#### HTTP Security Settings
If the Web service requires authentication (either HTTP or HTTPS), click **Edit** to specify the required authentication settings (see Setting HTTP Security).

#### WS-Security Settings
These settings are applicable only for WSDL-style Web services. If the Web service uses SOAP security, click **Edit** to specify further settings (see Setting WS-Security).

#### Save all file paths relative to MFD file
When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. See also Using Relative and Absolute Paths.
10.4 Setting HTTP Security

The HTTP security settings must be configured in the following cases:

- You are calling a Web service through HTTPS and the Web service requires a client certificate
- The service uses an incorrect server certificate
- The Web service requires HTTP authentication.

To set HTTP security:

1. Open the Web Service Call Settings dialog box (see Web Service Call Settings).
2. Click the Edit button next to HTTP Security Settings.

The "HTTPS (TLS)" group of options applies for Web services called through HTTPS.

| Check Server Certificate | This check box is selected by default, meaning the MapForce is configured to check the certificate of the server before proceeding with the request. When this option is enabled, the Web service request (and the mapping) will fail if the server is not trusted, or if your operating system is not configured to trust the Web server.
|                         | It is not recommended to switch this option off unless you |

HTTP Security Settings dialog box

The "HTTPS (TLS)" group of options applies for Web services called through HTTPS.
### Allow host name mismatch between certificate and request

Sometimes a server certificate issued for a particular host name (for example, `www.example.com`) is installed on a different host name (for example, `example.com`).

Select this check box to proceed with authentication even if the host name of the certificate does not match the host name called by the Web service.

### Client certificate

Click **Select** to choose a client certificate from the *Current User\Personal* certificate store. This assumes the client certificate already exists in the *Current User\Personal* certificate store; otherwise, you can import it using the Certificates snap-in (see [Accessing the Certificate Stores on Windows](#)).

If the mapping will be deployed for execution to another operating system, the same certificate must be installed on the target operating system as well. For further information, see [Digital Certificate Management](#).

---

The "HTTP Authentication" group of options applies if the Web service requires HTTP authentication.

<table>
<thead>
<tr>
<th>Username</th>
<th>Specifies the username required to access the Web service.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
<td>Specifies the password required to access the Web service.</td>
</tr>
<tr>
<td><strong>Dynamic authentication</strong></td>
<td>Select this check box if you would like to supply the username and password from the mapping instead of entering them into this dialog box.</td>
</tr>
<tr>
<td><strong>Preemptive authentication</strong></td>
<td>Select this check box if the Web service is configured to expect authentication data in the first call. Otherwise, MapForce attempts access without username and password and will use them if the server requires authorization (HTTP status 401).</td>
</tr>
</tbody>
</table>
10.5 Setting WS-Security

The WS-Security settings must be configured if the Web service is protected by WS-Security and requires that you provide the **UsernameToken** security token.

**Conventions**

The following abbreviations for the namespaces applicable to Web services are used in this topic:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsse</td>
<td><a href="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssext-1.0.xsd">http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssext-1.0.xsd</a></td>
</tr>
<tr>
<td>wsu</td>
<td><a href="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssext-utility-1.0.xsd">http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssext-utility-1.0.xsd</a></td>
</tr>
</tbody>
</table>

**To set WS-Security:**

1. Open the Web Service Call Settings dialog box (see [Web Service Call Settings](#)).
2. Click the **Edit** button next to WS-Security Settings.

![WS-Security Settings dialog box](#)

| Add UsernameToken A UsernameToken is an optional WS-security element |
|--------------------------------------------------------|---------------------------------------------------------|
present in the header of the SOAP message. The UsernameToken is used by the Web server to authenticate the caller of the Web service.

In the `wsse` specification (see [Conventions](#)), this field corresponds to:

```
/wsse:UsernameToken
```

### Username
Enter the username included in the UsernameToken.

In the `wsse` specification (see [Conventions](#)), this field corresponds to:

```
/wsse:UsernameToken/wsse:Username
```

### Password
Enter the text of the password included in the UsernameToken.

In the `wsse` specification (see [Conventions](#)), this field corresponds to:

```
/wsse:UsernameToken/wsse:Password
```

### Password type
Select the type of password included in the UsernameToken. Select **Digest** if the Web server expects the password in this mode; otherwise select **Text**.

In the `wsse` specification (see [Conventions](#)), this field corresponds to:

```
/wsse:UsernameToken/wsse:Password/@Type
```

### Dynamic authentication
Select this check box if you would like to supply the username and password from the mapping instead of entering them into this dialog box.

### Add nonce
Select this check box if you want to add a nonce to the Username token. A nonce is a random value which uniquely identifies each UsernameToken to provide additional security. If you enable this option, it is recommended to enable the **Add "Created" timestamp** option as well.

In the `wsse` specification (see [Conventions](#)), this field corresponds to:

```
/wsse:UsernameToken/wsse:Nonce
```

### Add "Created" timestamp
Select this option to add a timestamp to each nonce.

In the `wsse` specification (see [Conventions](#)), this field corresponds to:

```
/wsse:UsernameToken/wsu:Created
```
### Add Timestamp

Select this check box if you want to enable the time-to-live (TTL) value for the SOAP message (see the next option).

In the **wsu** specification (see [Conventions](#)), this field corresponds to:

```xml
/wsuh:Timestamp
```

### Time to live

Enter the time-to-live (TTL) for the SOAP message to diminish the chance of someone intercepting the message and replaying it.
10.6 Example: Calling a REST-Style Web Service

This example shows you how to call a generic HTTP Web service from MapForce. The description of the Web service called in this example can be found at https://app.swaggerhub.com/apis/FAA/ASWS/1.1.0. This Web service returns the current status of any major US airport, as an XML or JSON structure, accepting the three-letter airport code as argument (for example, "SFO", "IAD", "ABE", "DFW", etc). The example is accompanied by a mapping design file, which is available at the following path: <Documents>\Altova\MapForce2019\MapForceExamples\AirportStatus.mfd.

In this example, the response of the Web service is in JSON format, and is mapped to a JSON file. Therefore, the JSON schema of the Web service response structure will be required. For convenience, the JSON schema was generated with XMLSpy from the a sample response supplied by the Web service provider. It is available at the following path: <Documents>\Altova\MapForce2019\MapForceExamples\AirportStatus.schema.json.

For the request part, we are going to call the Web service with a parameter which will supply the value "SFO" in the request. For the response part, we will map the response data to a JSON file. Therefore, the mapping will consist of three main components: the Web service call, the input parameter, and the JSON output file.

Step 1: Add the Web service component

1. Make sure the transformation language of the mapping is BUILT-IN (see Selecting a Transformation Language).
4. Set the request method to GET and the URL to https://soa.smext.faa.gov/asws/api/airport/status/{airportCode}). The value within curly braces is a template parameter which will be replaced with the actual airport code at runtime (see Adding a Web Service Call (REST-Style).
5. Click the Add Parameter ( ) button and add a new parameter to the "Parameters" table. Notice the name of the parameter must be the same as that of the parameter specified within curly braces in the URL. Set the style to "Template", type to "String", check the "Mappable" and the "Required" option. The "Description" field is optional.
6. Add a header to tell the Web server that the client (that is, MapForce) expects JSON in the response. To do this, click again the Add Parameter ( ) button, name the parameter "Accept", set the style to "Header" and enter application/json as fixed value.
7. Under **Response**, click the **Edit** button and browse for the schema of the Web service response. The schema can be found at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\AirportStatus.schema.json`. 
8. Click **OK**. The Web service is now available on the mapping area.

Step 2: Add the input parameter

1. On the **Insert** menu, click **Insert Input**.
2. Enter "airportCode" as parameter name and click to clear the **Input is required** option (since the input will be supplied by a constant, as shown in the next step).
3. Add a constant (using the menu command **Insert | Constant**) with value "SFO" and connect it to the parameter input.

4. Connect the output of the airportCode parameter to the input of the request, as shown below.

At this stage, the request part of the Web service is ready. At mapping runtime, the Web service will be called with the value provided by the constant (in this case, "SFO").

**Step 3: Add the JSON output**

1. On the **Insert** menu, click **JSON Schema/File**.
2. Browse for the `<Documents>\Altova\MapForce2019\MapForceExamples \AirportStatus.schema.json`, and click **Open**.
3. When prompted to supply a sample JSON file, click **Skip** (there is no need for a sample JSON file since it will be generated).
4. On the **Connection** menu, make sure that the **Auto Connect Matching Children** menu item is enabled. Enabling this option saves you time with the next step.
5. Connect the **object** node of the response body of the Web service to the **object** node of
the JSON component, as shown below. Since the **Auto Connect Matching Children**
was enabled in the previous step, all descendent nodes are connected automatically, so
you don't need to draw individual connections for each.

![Diagram of JSON component](image)

**Step 4: Execute the mapping**

You are now ready to call the Web service. Click the **Output** button to execute the mapping and
preview the generated output. If the Web service call is executed successfully, the **Output** tab
displays the returned JSON structure. If the call was not successful, MapForce returns the error
accordingly. As stated in the description of the Web service, the HTTP code 500 may be returned
by the Web service in case of error response. You may also get the HTTP code 404 if the delays
information is unavailable.

If you have MapForce Server ([https://www.altova.com/mapforce-server](https://www.altova.com/mapforce-server)), you can also compile the
mapping to a mapping execution file (".mfx") and execute it from the command-line or from the
MapForce Server API on the server machine where MapForce Server runs (see Compiling
Mappings to MapForce Server Execution Files).

You have now finished creating a generic HTTP Web service call that uses a GET method to
retrieve airport status data in real time. For more information about working with generic Web
services, see also Adding a Web Service Call (REST-Style).
10.7 Example: Mapping Data from an RSS Feed

This example shows you how to call a generic HTTP Web service from MapForce in order to map data from an RSS (Rich Site Summary) feed to Microsoft Excel. The example is accompanied by a mapping design file, which is available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\RssReader.mfd`.

In this example, the response of the Web service is mapped from the RSS feed of the Altova blog (https://blog.altova.com/feed). The schema of the Web service response structure will be required, so that MapForce can create the structure of the data returned by the feed. For convenience, the required schema is available at the following path relative to your "(My) Documents" folder: `<Documents>\Altova\MapForce2019\MapForceExamples\rss-2-0.xsd`.

Calling the RSS feed does not require any request parameter, so the request part of the Web service call will be empty. As for the response, it will be mapped to a Microsoft Excel file. Thirdly, in order to make the publication date of each RSS entry easily readable, it will be formatted as YYYY-MM-DD in the Excel file. To achieve this goal, date processing functions will be used.

Therefore, the mapping will consist of three main parts: the Web service call, the target Excel component, and the date processing functions.

Step 1: Add the Web service component

1. Make sure the transformation language of the mapping is BUILT-IN (see Selecting a Transformation Language).
4. Set the request method to GET and the URL to https://blog.altova.com/feed.
5. Under Response, click the Edit button and browse for the schema of the Web service response. The schema can be found at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\rss-2-0.xsd`.

6. Click OK.
At this stage, the Web service is available on the mapping area and you are ready to map data from the response part.

### Step 2: Add the target Excel component

1. On the **Insert** menu, click **Excel 2007+ File**.
2. When prompted to supply a sample file, click **Skip**. The Excel component is now available on the mapping area.
3. Click the button next to **Sheet 1**, and click to clear the **Show Worksheets by name** option.
4. Click the button next to **Row 1**, and change the options as shown below.
5. Connect the following items of the response body to the following items of the Excel component:
Calling Web Services Example: Mapping Data from an RSS Feed

<table>
<thead>
<tr>
<th>Response body</th>
<th>Excel file</th>
</tr>
</thead>
<tbody>
<tr>
<td>channel</td>
<td>Worksheets</td>
</tr>
<tr>
<td>title</td>
<td>Worksheet Name</td>
</tr>
<tr>
<td>item</td>
<td>Rows</td>
</tr>
<tr>
<td>item/title</td>
<td>Row B (Title)</td>
</tr>
<tr>
<td>item/link</td>
<td>Row C (Link)</td>
</tr>
</tbody>
</table>

At this stage, the **Row A (Date)** is still missing an input value. This value will be added in the next step.

**Step 3: Add the date processing functions**

The date in the response of the RSS feed appears formatted in RFC-822 format, for example: `Wed, 20 Jan 2016 14:49:35 +0000`. The goal is, however, to see the date formatted in the Excel file as `YYYY-MM-DD`. To achieve this, add the `substring-after` and `parse-date` functions of the MapForce core library to the mapping (for instructions about adding functions to the mapping, see [Working with Functions](#)).

As shown above, the `substring-after` function takes the value from the `pubDate` node and returns only the text after `, `. The resulting value is then passed to another MapForce core function, `parse-date`, which parses it using the mask `[D] [Mn] [Y]`, and returns the value as `xs:date`. The format mask means "numeric day of the month, followed by a space, followed by the month name in title case, followed by a space, followed by the four-digit year" (for more information, see the `format-dateTime` function).

Finally, the parsed value is supplied to the **Date** row of the Excel component.

**Step 4: Executing the mapping**

You are now ready to call the Web service. Click the **Output** button to execute the mapping and preview the generated output. If the Web service call is executed successfully, the **Output** tab displays the returned data structure. If the call was not successful, MapForce returns the error.
accordingly.

If you have MapForce Server (https://www.altova.com/mapforce-server), you can also compile the mapping to a mapping execution file (*.mfx) and execute it from the command-line or from the MapForce Server API on the server machine where MapForce Server runs (see Compiling Mappings to MapForce Server Execution Files).

You have now finished creating a generic HTTP Web service call that uses a GET method to read data from an RSS feed. For general information about working with generic Web services, see also Adding a Web Service Call (REST-Style).
10.8 Example: Calling a SOAP Web Service

This example shows you how to query a Web time service using a constant as an input. The Web service itself was implemented using MapForce. This is for demonstration purposes—you can implement the Web service with any other technology that supports a compatible protocol.

The mapping shown below is part of the TimeService.mfp mapping project, available in the <Documents>\Altova\MapForce2019\MapForceExamples\TimeService folder. The TimeService2.mfp project file available in the <Documents>\Altova\MapForce2019 \MapForceExamples\TimeserviceWsdl2 folder supports WSDL 2.0.

To view how the Web service is implemented:

1. Select File | Open and select the TimeService.mfp file in the <Documents>\Altova \MapForce2019\MapForceExamples\Timeservice folder. The files associated with the project are loaded in the Project window.

2. Double-click the getCityTime.mfd entry in the project window.
The `getCityTime.mfd` mapping accepts a city name as input and returns the current time in the output. If the city is not "Boston", a WSDL fault is returned. The mapping takes the input data from the `getCityTimeRequest.xml` file available in the `<Documents>\Altova\MapForce2019\MapForceExamples\TimeService` directory. If you double-click the `getCityTimeSoapIn` component, you can see that this file is set as data source.

For the scope of this example, we will now assume that this particular Web service project has already been compiled and deployed to an actual Web server which you are going to call in the following steps of this example. A demo online service is available for that purpose at [http://nanonull.com/TimeService/TimeService.asmx](http://nanonull.com/TimeService/TimeService.asmx). For examples of how to create such Web services with MapForce, see Example: Generating SOAP Web Services (C#) and Example: Generating SOAP Web Services (Java).

To add the Web service to the mapping:

1. Select **File | New**, click the Mapping icon and confirm with OK.
2. Select the menu option **Insert | Web service function...** or click the toolbar button.
3. Click Browse to select the WSDL definition file; select `TimeService.wsdl` from the `TimeService` directory, then click the Open button.
4. When prompted to choose a Web service endpoint, click `TimeServiceSoap`.
5. When prompted to choose a Web service operation, click `getCityTime`.
6. Leave the WSDL Call Settings unchanged, and click OK. For more information, see Web Service Call Settings.

The getCityTime Web service function is inserted as a single component. Note that it actually represents all eight components that make up the getCityTime.mfd file as saved in the WSDL project.

The left section of the component defines the data input (SoapIn), while the right side defines the data output (SoapOut), which may also include a fault section, if one has been defined in the .wsdl file.

To call the Web service:

1. Insert the component that is to supply the input data, e.g. a constant, text, or schema component. For the scope of this example, insert a constant component, and enter “Boston” as the input string.
2. Connect the constant to the \texttt{n0:city} item.
3. Insert a simple output component (on the \texttt{Function} menu, click \texttt{Insert Output}).
4. Connect the \texttt{n0:getCityTimeResult} to the output component.

5. Right-click the “getCityTime” component on the mapping, and select \texttt{Properties}. The Web Service Call Settings dialog box opens. Enter \texttt{http://nanonull.com/TimeService/TimeService.asmx} in the URL box (this is a demo online Web service created with MapForce, see Implementing SOAP Web Services).
6. Click the Output tab. As illustrated below, the current time in Boston is displayed in the Output pane.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{example.png}
\caption{Example: Calling a SOAP Web Service}
\end{figure}

\textbf{Note:} The input value of the Web service function takes precedence over the data source of the original mapping. For example, the constant “Boston” takes precedence over the \texttt{getCityTimeRequest.xml} data source file in the original mapping.

\textbf{To map Web service faults:}

1. Select \texttt{Insert | Exception}, or click the Exception toolbar button \texttt{\textcolor{red}{\textbullet}}.
2. Map the \texttt{Fault:} item to the \texttt{throw} item of the exception component.
3. Map the \texttt{n0:errorText} item to the \texttt{error-text} item of the exception component.
Digital Certificate Management

Digital certificate management is an integral part of secure data exchange between a client computer and a Web server. Since mappings can be executed not only on Windows by MapForce, but also on a Windows, Linux or macOS server by MapForce Server (either standalone or in FlowForce Server execution), this section deals with managing HTTPS certificates on various platforms.

In the context of secure HyperText Transport Protocol (HTTPS), it is important to distinguish between server and client certificates.

Server certificates

A server certificate is what identifies a server as a trusted entity to a client application such as MapForce. The server certificate may be digitally signed by a commercial Certificate Authority, or it may be self-signed by your organization. In either case, while designing the mapping in MapForce, you can specify the following settings:

- Whether the server certificate must be checked.
- Whether the request must proceed if a mismatch has been detected between the name certificate and the name of the host.

These settings are available on the HTTP Security Settings dialog box of MapForce (see Setting HTTP Security). When you enable server certificate checks, consider the following:

- If you are calling a Web server whose certificate is signed by a trusted Certificate Authority, your operating system will likely be already configured to trust the server certificate, and no additional configuration is necessary.
- If you are calling a Web server which provides a self-signed certificate (for example, a local network server within your organization), you will need to configure your operating system as well to trust that certificate.

In most cases, you can check the level of trust between your operating system and the Web server by typing the URL of the Web service in the browser's address bar. If the server is not trusted, or if your operating system is not configured to trust the server, your browser will display a message such as "This connection is untrusted", or "There is a problem with this website's certificate". Note that you cannot use the browser to check the level of trust with a Web server if the browser uses a certificate database other than that of the operating system (for example, Firefox 35.0.1 on Ubuntu 14.04).

On Windows, you can establish trust with the server by following the browser's instructions and importing or installing the required certificates into your system's Trusted Root Authorities store (see Trusting Server Certificates on Windows). On macOS, you can do the equivalent operation in Keychain Access (see Trusting Server Certificates on macOS). For instructions applicable to Linux, see Trusting Server Certificates on Linux.

Client certificates

While server certificates are used to identify a server as a trusted entity, client certificates are primarily used to authenticate the caller against the Web server. If you intend to call a Web server which requires client certificates, you may need to contact the administrator of the Web server for
Calling Web Services

The client configuration instructions. Taking IIS (Internet Information Services) as an example, the Web server may be configured to handle HTTPS and client certificates in one of the following ways:

- Require HTTPS and ignore client certificate
- Require HTTPS and accept client certificate
- Require HTTPS and require client certificate

The success or failure of the Web service request depends both on the configuration of the Web server and the client application. For example, if the Web server is configured to require a client certificate, then, for the call to be successful, the calling application must present a valid client certificate.

From a MapForce perspective, the same is true for mappings which include Web service calls through HTTPS. In particular, to run such mappings successfully, it is assumed that the Web server has been configured to accept or require the client certificate, and that the operating system where the mapping runs provides the correct client certificate to the Web server.

The diagram below illustrates a scenario where a client certificate used in MapForce is transferred to a Linux server running MapForce Server. Once the certificate has been transferred to the target operating system, MapForce Server can use it to authenticate itself against the Web server and execute the mapping successfully.

For HTTPS authentication in Web service calls, MapForce is capable of using Transport Layer Security (TLS) on top of HTTP, which is the successor of Secure Sockets Layer (SSL) protocol. Note that fallback to SSL may occur if either the client implementation or the server does not support TLS.

To support Web calls with client certificate authentication on multiple platforms, MapForce (and MapForce Server) relies on the certificate management implementation of each platform, thus...
ensuring that certificate management is always in the scope of the underlying operating system. Each operating system provides different support for certificate management, as shown in the table below.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Certificate management and implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>On Windows, you can manage certificates using the Certificate snap-in (see Accessing the Certificate Stores on Windows). TLS support is available through the Secure Channel (also known as SChannel) library.</td>
</tr>
<tr>
<td>Linux</td>
<td>On Linux, you can manage certificates using the OpenSSL (openssl) command line tool and library. If OpenSSL support is not already available on the Linux machine where MapForce Server is installed, you will need to download and install it before you can manage certificates. TLS support is available through the OpenSSL library (<a href="https://www.openssl.org/">https://www.openssl.org/</a>).</td>
</tr>
<tr>
<td>macOS</td>
<td>On macOS, you can manage certificates using the Keychain Access Manager, located under Finder &gt; Applications &gt; Utilities. TLS support is provided by the Secure Transport library native to the operating system.</td>
</tr>
</tbody>
</table>

If you execute the mapping on a Windows operating system where you can already successfully consume the same Web service that you intend to call from MapForce, no additional certificate configuration is normally required (for the conditions to run the mapping successfully on Windows, see Client Certificates on Windows). However, if you design mappings with MapForce on a Windows computer, and then deploy them to another computer (which may run a different operating system), the client certificate is not stored or copied together with the deployed package. For the Web service call (and the mapping) to execute successfully, the client certificate must exist on the target operating system as well.

To transfer a certificate from a Windows system to another Windows-based computer, export the required certificate (with private key) from the source system (see Exporting Certificates from Windows). Then import the same certificate to the Current User\Personal store on the target operation system (see Client Certificates on Windows).

For instructions on how to transfer client certificates to the Linux and macOS platforms, see Client Certificates on Linux and Client Certificates on macOS, respectively.

10.9.1 Trusting Server Certificates on Linux

On Linux, you can import a trusted certificate into the system's certificate store as shown below.

Perform the following steps only if you are sure of the authenticity of the certificate you want to trust.
On Debian and Ubuntu, follow the steps below:

1. Copy the certificate file of the Web server to the following directory.
   
   ```
   sudo cp /home/downloads/server_cert.crt /usr/local/share/ca-certificates/
   ```

2. Update the certificate store as follows:
   
   ```
   sudo update-ca-certificates
   ```

On CentOS, follow the steps below:

1. Install the `ca-certificates` package:
   
   ```
   yum install ca-certificates
   ```

2. Enable the dynamic certificate authority configuration feature:
   
   ```
   update-ca-trust enable
   ```

3. Copy the server certificate to the following directory:
   
   ```
   cp server_cert.crt /etc/pki/ca-trust/source/anchors/
   ```

4. Use the command:
   
   ```
   update-ca-trust extract
   ```

For cases where you need to access the server only through the browser, it is sufficient to import the certificate into the browser certificate store. The exact instructions will vary for each browser. For example, in Firefox 59.0.2, you can do this as follows:

1. Under **Options | Privacy & Security**, click **View Certificates**.
2. On **Authorities** tab, click **Import** and browse for the root certificate file created previously.
3. When prompted, select **Trust this CA to identify websites**.
10.9.2  Trusting Server Certificates on macOS

On macOS, you can import a trusted certificate into Keychain Access as follows.

1. Run Keychain Access.
2. Click System, and then click Certificates.
3. On the File menu, click Import Items.
4. Browse for the trusted certificate, and click Open.
5. Enter the Keychain Access password when prompted, and then click Modify Keychain.
6. Double-click the certificate, expand the Trust section, and select Always Trust.
10.9.3 Trusting Server Certificates on Windows

On Windows, you can import a trusted certificate into the system certificates store as follows:

1. Open the Windows certificate store for the computer account, see Accessing Windows Certificate Store.

   Perform the following steps only if you are sure of the authenticity of the Web server certificate.

2. Under "Trusted Root Certification Authorities", right-click Certificates, and select All Tasks | Import, and follow the certificate import wizard.
10.9.4 Accessing the Certificate Stores on Windows

On Windows, you can manage certificates from the Microsoft Management Console (MMC) snap-in, either for your user account, or for the computer account.

To open the Certificates snap-in (for the current Windows user):

- Run `certmgr.msc` at the command line.

To open the Certificates snap-in (for the computer account):

1. Run `mmc` at the command line.
2. On the **File** menu, click **Add/Remove Snap-in**.
3. Click **Certificates**, and then click **Add**.

4. Click **Computer account**, and click **Next**.

5. Click **Local computer**, and then click **Finish**.
10.9.5 Exporting Certificates from Windows

For mappings that call Web services through HTTPS and are deployed to a macOS or Linux server running MapForce Server or FlowForce Server, the same client certificate must be available on the non-Windows operating system as the one used on Windows to design and test the mapping. To execute such mappings on a non-Windows operating system with MapForce Server, export the required certificate with private key from Windows and then import it into the target operating system.

To export a certificate with private key from Windows:

1. On Windows, open the Certificates snap-in (see Accessing the Certificate Stores on Windows).
2. Right-click the certificate that you want to export, point to All Tasks, and then click Export.
3. Click Next.
4. Choose to export from Windows the certificate together with its private key, and then click **Next**.
5. Choose the *Personal Information Exchange - PKCS #12 (.pfx)* file format, and then click **Next**.
**Note:** Make sure not to select the option *Delete the private key if the export is successful*, otherwise you will not be able to make use of the certificate after it is exported.

6. Enter a password, and then click **Next**. You will need this password after you copy the certificate to the target operating system.
7. Browse for the location of the file to export, and then click Next.
8. Click **Finish**.
If your mappings include Web service authentication through HTTPS by means of client certificates, follow these steps to deploy such mappings to a Linux machine running MapForce Server:

1. Open the mapping which calls the Web service.
2. Double-click the header of the Web Service component. The Component Settings dialog box appears.
3. Click **Edit** next to HTTP Security Settings.
4. In the HTTP Security Settings dialog box, click **Client Certificate**, and then select the required certificate from the **Current User\Personal** store on Windows (see **Setting HTTP Security**).
5. Save the mapping and compile it to a mapping execution file or deploy it to FlowForce Server (see **Compiling Mappings to Server Execution Files** and **Deploying Mappings to FlowForce Server**).
6. Transfer the client certificate required by the Web service call to the target operating system. Make sure that the certificate has a private key, and that the **Enhanced Key Usage** property of the certificate includes “Client authentication” as purpose.
To transfer the client certificate to Linux:

1. Export the client certificate with private key from Windows, in the Personal Information Exchange - PKCS #12 (.pfx) file format (see Exporting Certificates from Windows).
2. Copy the certificate file to the Linux machine.
3. Convert the .pfx file to .pem format using the command:

   ```bash
   openssl pkcs12 -in cert.pfx -out "John Doe.pem" -nodes
   ```

   This command parses the .pfx file and outputs a .pem file, without encrypting the private key. Certificates with an encrypted private key prompt for password and are not supported in server execution.

Executing the mapping

To instruct MapForce Server to use the .pem file as client certificate, set the --certificatespath parameter when running the mapping. The --certificatespath parameter defines the path of the directory where all certificates required by the current mapping are stored. For example, if the certificate file path is /home/John/John Doe.pem, then --certificatespath must be set to /home/John.

By default, if the --certificatespath parameter is not provided, MapForce Server looks for certificates in the directory $HOME/.config/altova/certificates of the current user.

For the mapping to execute successfully, the certificate file is expected to have the .pem extension and the file name must match the Common Name (CN) of the certificate, including spaces (for example, John Doe.pem). If the CN contains a forward slash (/), it must be replaced with an underscore (_) character.

If you intend to execute the mapping as a FlowForce Server job, copy the certificate file to the $HOME/.config/altova/certificates directory. When running the job, FlowForce Server will use this directory to look for any certificate files required by the mapping.

For security considerations, make sure that certificate files are not readable by other users, since they contain sensitive information.

10.9.7 Client Certificates on macOS

If your mappings include Web service authentication through HTTPS client certificates, follow these steps to deploy such mappings to a macOS running MapForce Server:

1. Open the mapping which calls the Web service.
2. Double-click the header of the Web Service component. The Component Settings dialog box appears.
3. Click Edit next to HTTP Security Settings.
4. In the HTTP Security Settings dialog box, click Client Certificate, and then select the required certificate (see Setting HTTP Security).
5. If the certificate name does not match exactly the host name of the server, select **Allow name mismatch between certificate and request**.

6. Save and deploy the mapping to the target operating system (see [Deploying a MapForce mapping](#)).

7. Transfer the client certificate required by the Web service call to the target operating system. Make sure that the certificate has a private key, and that the **Enhanced Key Usage** property of the certificate includes “Client authentication” as purpose.

**To transfer the client certificate to macOS:**

1. Export the client certificate with private key from Windows, in the **Personal Information Exchange - PKCS #12 (.pfx)** file format (see [Exporting Certificates from Windows](#)) and copy the .pfx file to the macOS.

2. If this hasn’t been done already, make sure that the operating system trusts the server certificate (see [Trusting Server Certificates on Mac OS](#)).

3. Run Keychain Access from **Finder > Applications > Utilities**.

4. On the **File** menu, click **Import Items**.

5. Browse for the client certificate exported from Windows in step 1 and select a destination keychain.

6. Click **Open** and enter the password with which the certificate was encrypted.

**Executing the mapping**

You are now ready to run the mapping using the MapForce Server **run** command. Note the following:

- If you execute the mapping remotely through SSH, first unlock the keychain with the `security unlock-keychain` command.
- If you execute the mapping through the macOS graphical user interface, when prompted to allow MapForce Server access to the keychain, click **Allow**.

**10.9.8 Client Certificates on Windows**

When you run on Windows a mapping which requires client certificates, the conditions to run the mapping successfully are as follows:

- The client certificate must exist in the **Current User\Personal** certificate store (also
referred to as the My store). For the certificate to exist in this store, it must be imported through the Certificate Import Wizard. For instructions, see https://technet.microsoft.com/en-us/library/cc754489(v=ws.11).aspx.

- The certificate must have a private key.
- The Enhanced Key Usage property of the certificate must include "Client authentication" as purpose.

In the current version of MapForce, due to a limitation of the library used by MapForce, Windows will select the required certificate automatically from the certificate store when you run the mapping. The mapping will execute successfully if, after filtering the Current User\Personal certificate store, the server finds a suitable certificate. Note that the HTTPS authentication (and the certificate selection operation) is managed by Windows and is not controlled by MapForce or MapForce Server. In some cases, if multiple certificates exist in the Current User\Personal store, an unsuitable certificate may be selected automatically by the operating system, which causes the mapping execution to fail. This situation can be avoided by limiting the number of certificates available in the Current User\Personal store.
Chapter 11

Automating Mappings and MapForce
11 Automating Mappings and MapForce

Mappings designed with MapForce can be executed in a server environment (including Linux and macOS servers), and with server-level performance, by the following Altova transformation engines (licensed separately):

- **RaptorXML Server.** Running a mapping with this engine is suitable if the transformation language of the mapping is XSLT 1.0, XSLT 2.0, or XQuery. See Automation with RaptorXML Server.

- **MapForce Server (or MapForce Server Advanced Edition).** This engine is suitable for any mapping where the transformation language is BUILT-IN*. The BUILT-IN language supports the most mapping features in MapForce, while MapForce Server (and, in particular, MapForce Server Advanced Edition) provides best performance for running a mapping. See Automation with MapForce Server.

* The BUILT-IN transformation language requires MapForce Professional or Enterprise Edition.

In addition to this, MapForce provides the ability to automate generation of XSLT, XQuery, C#, C++, and Java code from the command line interface. This includes the ability to compile server execution files (.mfx) intended for MapForce Server execution. For more information, see MapForce Command Line Interface.
11.1 Automation with RaptorXML Server

RaptorXML Server (hereafter also called RaptorXML for short) is Altova's third-generation, super-fast XML and XBRL processor. It has been built to be optimized for the latest standards and parallel computing environments. Designed to be highly cross-platform capable, the engine takes advantage of today's ubiquitous multi-core computers to deliver lightning fast processing of XML and XBRL data.

RaptorXML is available in two editions which can be downloaded from the Altova download page (https://www.altova.com/download-trial-server.html):

- RaptorXML Server is a very fast XML processing engine with support for XML, XML Schema, XSLT, XPath, XQuery, and more.
- RaptorXML+XBRL Server supports all the features of RaptorXML Server with the additional capability of processing and validating the XBRL family of standards.

If you generate code in XSLT 1.0 or 2.0, or in XQuery, MapForce creates a batch file called DoTransform.bat which is placed in the output folder that you choose upon generation. Executing the batch file calls RaptorXML Server and executes the XSLT (or XQuery) transformation on the server.

If you intend to execute or automate MapForce mappings for other outputs on a server, see Automation with MapForce Server.

Note: You can also preview the XSLT and XQuery code using the built-in engine.
### 11.2 Automation with MapForce Server

MapForce Server is an enterprise server software solution for Windows, Linux and macOS operating systems. The role of MapForce Server is to execute mappings in a server environment (including on non-Windows platforms) and with server-level performance. Any MapForce mapping where the target execution language is BUILT-IN qualifies for server execution (see also Selecting a Transformation Language). MapForce Server can operate either standalone (invoked from command line or API), or under the management of FlowForce Server.

If MapForce Server is used as a standalone product then the MapForce mapping has to be compiled and copied to the machine where MapForce Server runs. The mapping is then run using the MapForce Server command line command `run`. You can also run the mapping by invoking the `run` method of the MapForce Server API. For further information, see Compiling Mappings to MapForce Server Execution Files.

If MapForce Server runs under FlowForce Server management, the mapping can be deployed to a target machine through an HTTP (or SSL/HTTPS) connection directly from MapForce. On the server, the mapping can then be executed as a triggered or scheduled job, or through a Web service call defined from the the FlowForce Server administration interface. For further information, see Deploying Mappings to FlowForce Server.

There are two editions of MapForce Server:

- MapForce Server
- MapForce Server Advanced Edition (this edition is part of the FlowForce Server installation package)

MapForce Server Advanced Edition provides the same features as MapForce Server, and additionally includes optimization features for mappings which qualify for optimization. This is the case of mappings which join or filter large amounts of data, and where it is possible to apply join optimization so as to increase the execution speed. Unlike MapForce Server, MapForce Server Advanced Edition can execute mappings where node functions are present, see Defaults and Node Functions.

Limitations:

- XML digital signatures are not supported
- ADO, ADO.NET, and ODBC database connections are supported only on Windows (for other operating systems, see Preparing Mappings for Server Execution).

For more information about MapForce Server, refer to its accompanying documentation (https://www.altova.com/documentation).
11.3 Preparing Mappings for Server Execution

A mapping designed and previewed with MapForce may refer to resources which are outside of the current machine and operating system (such as databases). In addition to this, in MapForce, all mapping paths follow Windows-style conventions by default. Thirdly, the machine where MapForce Server runs might not support the same database connections as the machine where the mapping was designed. For this reason, running mappings in a server environment typically requires some preparation, especially if the target machine is not the same as the source machine.

**Note:** The term "source machine" refers to the computer where the MapForce is installed and the term "target machine" refers to the computer where MapForce Server or FlowForce Server is installed. In the most simple scenario, this is the same computer. In a more advanced scenario, MapForce runs on a Windows machine whereas MapForce Server or FlowForce Server runs on a Linux or macOS machine.

As best practice, always make sure that the mapping validates successfully in MapForce before deploying it to FlowForce Server or compiling it to a MapForce Server execution file (see Validating Mappings).

If MapForce Server runs standalone (without FlowForce Server), the required licenses are as follows:

- On the source machine, MapForce Enterprise or Professional edition is required to design the mapping and compile it to a server execution file (.mfx), see Compiling Mappings to MapForce Server Execution Files.
- On the target machine, MapForce Server or MapForce Server Advanced Edition is required to run the mapping.

If MapForce Server runs under FlowForce Server management, the following requirements apply:

- On the source machine, MapForce Enterprise or Professional edition is required to design the mapping and deploy it to a target machine, see Deploying Mappings to FlowForce Server.
- Both MapForce Server and FlowForce Server must be licensed on the target machine. The role of MapForce Server is to run the mapping; the role of FlowForce is to make the mapping available as a job which benefits from features such as scheduled or on demand execution, execution as a Web service, error handling, conditional processing, email notifications, and others.
- FlowForce Server must be up and running at the configured network address and port. Namely, the "FlowForce Web Server" service must be started and configured to accept connections from HTTP clients (or HTTPS if configured) and must not be blocked by the firewall. The "FlowForce Server" service must also be started and running at the designated address and port.
- You have a FlowForce Server user account with permissions to one of the containers (by default, the /public container is accessible to any authenticated user).

**General considerations**

- If you intend to run the mapping on a target machine with standalone MapForce Server,
all input and output files and schemas referenced by the mapping must be copied to the target machine as well. If MapForce Server runs under FlowForce Server management, there is no need to copy files manually. In this case, the instance and schema files are included in the package deployed to the target machine, see Deploying Mappings to FlowForce Server.

- If the mapping includes database components which require specific database drivers, such drivers must be installed on the target machine as well. For example, if your mapping reads data from a Microsoft Access database, then Microsoft Access or Microsoft Access Runtime (https://www.microsoft.com/en-us/download/details.aspx?id=50040) must be installed on the target machine as well.

- When you deploy a mapping to non-Windows platforms, ADO, ADO.NET and ODBC database connections are automatically changed to JDBC. Native SQLite and native PostgreSQL connections are preserved as such and require no additional configuration. See also “Database connections” below.

- If the mapping contains custom function calls (for example, to .dll or .class files), such dependencies are not deployed together with the mapping, since they are not known before runtime. In this case, copy them manually to the target machine. The path of the .dll or .class file on the server must be the same as on the “Options” dialog box in MapForce, for example:

  ![Options dialog box](image)

Some mappings read multiple input files using a wildcard path (see Processing Multiple Input or Output Files Dynamically). In this case, the input file names are not known before runtime and so they are not deployed. For the mapping to execute successfully, the input files must exist on the target machine.

- If the mapping output path includes directories, those directories must exist on the target machine. Otherwise, an error will be generated when you execute the mapping. This behavior is unlike MapForce, where non-existing directories are generated automatically if the option Generate output to temporary files is enabled (see Changing the MapForce Options).

- If the mapping calls a Web service that requires HTTPS authentication with a client
Preparing Mappings for Server Execution

Making paths portable

If you intend to run the mapping on a server, ensure that the mapping follows the applicable path conventions and uses a supported database connection.

To make paths portable to non-Windows operating systems, use relative instead of absolute paths when designing the mapping in MapForce, see Using Relative and Absolute Paths. For example, you can copy all input or output files required by the mapping into the same directory as the mapping, and then reference them just by file name. Importantly, both MapForce Server and FlowForce Server support a so-called "working directory" against which all relative paths will be resolved, see also Paths in Various Execution Environments. The working directory is specified at mapping runtime, as follows:

- In FlowForce Server, by editing the "Working-directory" parameter of any job.
- In MapForce Server API, through the WorkingDirectory property of the COM and .NET API, or through the setWorkingDirectory method of the Java API.
- In MapForce Server command line, the working directory is the current directory of the command shell.

Database connections

Be aware that ADO, ADO.NET, and ODBC connections are not supported on Linux and macOS machines. Therefore, if the target machine is Linux or macOS, such connections are converted to JDBC when you deploy the mapping to FlowForce or when you compile the mapping to a MapForce Server execution file. In this case, you have the following options before deploying the mapping or compiling it to a server execution file:

- In MapForce, create a JDBC connection to the database (see Setting up a JDBC Connection)
- In MapForce, fill the JDBC database connection details in the "JDBC-specific Settings" section of the database component (see Database Component Settings).

If the mapping uses a native connection to a PostgreSQL or SQLite database, the native connection is preserved and no JDBC conversion takes place, see Database mappings in various execution environments. If the mapping connects to a file-based database, such as Microsoft Access and SQLite, additional configuration is required, see "File-based databases" below.

Running mappings with JDBC connections requires that the Java Runtime Environment or Java Development Kit be installed on the server machine. This may be either Oracle JDK or an open source build such as Oracle OpenJDK.

- The JAVA_HOME environment variable must point to the JDK installation directory.
- On Windows, a Java Virtual Machine path found in the Windows registry will take priority over the JAVA_HOME variable.
To set up a JDBC connection on Linux or macOS:

1. Download the JDBC driver supplied by the database vendor and install it on the operating system. Make sure to select the 32-bit version if your operating system runs on 32-bit, and the 64-bit version if your operating system runs on 64-bit.
2. Set the environment variables to the location where the JDBC driver is installed. Typically, you will need to set the CLASSPATH variable, and possibly a few others. To find out which specific environment variables must be configured, check the documentation supplied with the JDBC driver.

**Note:** On macOS, the system expects any installed JDBC libraries to be in the `/Library/Java/Extensions` directory. Therefore, it is recommended that you unpack the JDBC driver to this location; otherwise, you will need to configure the system to look for the JDBC library at the path where you installed the JDBC driver.

**Oracle Instant Client connections on macOS**

These instructions are applicable if you connect to an Oracle database through the Oracle Database Instant Client, on macOS. Prerequisites:

- Java 8.0 or later must be installed. If the Mac machine runs a Java version prior to Java 8, you can also connect through the JDBC Thin for All Platforms library, and disregard the instructions below.
- Oracle Instant Client must be installed. You can download the Oracle Instant Client from the Oracle official download page. Note that there are several Instant Client packages available on the Oracle download page. Make sure to select a package with Oracle Call Interface (OCI) support, (for example, Instant Client Basic). Also, make sure to select the 32-bit version if your operating system runs on 32-bit, and the 64-bit version if your operating system runs on 64-bit.

Once you have downloaded and unpacked the Oracle Instant Client, edit the property list (.plist) file shipped with the installer so that the following environment variables point to the location of the corresponding driver paths, for example:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASSPATH</td>
<td>/opt/oracle/instantclient_11_2/ojdbc6.jar:/opt/oracle/instantclient_11_2/ojdbc5.jar</td>
</tr>
<tr>
<td>TNS_ADMIN</td>
<td>/opt/oracle/NETWORK_ADMIN</td>
</tr>
<tr>
<td>ORACLE_HOME</td>
<td>/opt/oracle/instantclient_11_2</td>
</tr>
<tr>
<td>DYLD_LIBRARY_PATH</td>
<td>/opt/oracle/instantclient_11_2</td>
</tr>
<tr>
<td>PATH</td>
<td>$PATH:/opt/oracle/instantclient_11_2</td>
</tr>
</tbody>
</table>

**Note:** Edit the sample values above to fit the paths where Oracle Instant Client files are installed.
on your operating system.

**File-based databases**

File-based databases such as Microsoft Access and SQLite are not included in the package deployed to FlowForce Server or in the compiled MapForce Server execution file. Therefore, if the source and target machine are not the same, take the following steps:

1. In MapForce, right-click the mapping and clear the check box **Make paths absolute in generated code** (see Changing the Mapping Settings).
2. Right-click the database component on the mapping and add a connection to the database file using a relative path, see Setting the Path to File-Based Databases. A simple way to avoid path-related issues is to save the mapping design (.mfd file) in the same directory as the database file and to refer to the latter from the mapping just by file name (thus using a relative path).
3. Copy the database file to a directory on the target machine (let’s call it “working directory”). Keep this directory in mind since it will be required to run the mapping on the server, as shown below.

To run such mappings on the server, do one of the following:

- If the mapping will be run by MapForce Server under FlowForce Server control, configure the FlowForce Server job to point to the working directory created previously. The database file must reside in the working directory. For an example, see “Exposing a Job as a Web Service” (https://www.altova.com/manual/FlowForceServer/FlowForceServerAdvanced/index.html?fs_example_web_service.htm).
- If the mapping will be run by standalone MapForce Server at the command line, change the current directory to the working directory (for example, `cd path\to\working\directory`) before calling the `run` command of MapForce Server.
- If the mapping will be run by the MapForce Server API, set the working directory programmatically before running the mapping. To facilitate this, the property `WorkingDirectory` is available for the MapForce Server object in the COM and .NET API. In the Java API, the method `setWorkingDirectory` is available.

If both the source and the target machines are Windows machines running on the local network, an alternative approach is to configure the mapping to read the database file from a common shared directory, as follows:

1. Store the database file in a common shared directory which is accessible by both the source and the target machine.
2. Right-click the database component on the mapping and add a connection to the database file using an absolute path (see Setting the Path to File-Based Databases).

**Global Resources**

If a mapping includes references to Global Resources instead of direct paths or database connections, such references are preserved when you compile the mapping to a server execution file (.mfx), or when you deploy the mapping to FlowForce Server, see Global Resources in Various Execution Environments.

**Note:** FlowForce Server does not currently support Global Resources. Do not use Global Resources if you intend to execute the mapping with MapForce Server running under FlowForce Server management.
XBRL Taxonomy Packages

When you deploy a mapping that references XBRL Taxonomy Packages (see XBRL Taxonomy Packages) to FlowForce Server, MapForce collects all external references from the mapping and then resolves them using the current configuration and currently installed taxonomy packages. If there are resolved external references that point to a taxonomy package, then the taxonomy package is deployed together with the mapping. FlowForce Server will use that package—as it was during deployment—to execute the mapping. To refresh the taxonomy package used by FlowForce Server, you will need to change it in MapForce and redeploy the mapping.

Note that the root catalog of MapForce Server influences the way taxonomies are resolved on the target machine. The root catalog is found at the following path relative to the MapForce Server installation directory: `etc/RootCatalog.xml`.

Taxonomy packages that were deployed with a mapping will be used if the root catalog of MapForce Server does not already contain such package or does not contain a package that is defined for the same URL prefix. The root catalog of MapForce Server has priority over the deployed taxonomy.

If MapForce Server runs standalone (without FlowForce Server), it is possible to specify the root catalog that should be used by the mapping as follows:

- At the command line, this is possible by adding the option `-catalog` to the `run` command.
- In the MapForce Server API, call the method `SetOption`, and supply the string "catalog" as first argument, and the path to the root catalog as second argument.

If a mapping uses XBRL components with table linkbases, the taxonomy package or the taxonomy package configuration file must be supplied to the mapping at runtime, as follows:

- At the MapForce Server command line, add the option `--taxonomy-package` or `--taxonomy-packages-config-file` to the run command.
- In the MapForce Server API, call the method `SetOption`. The first argument must be either "taxonomy-package" or "taxonomy-packages-config-file". The second argument must be the actual path to the taxonomy package (or taxonomy package configuration) file.
11.4 Compiling Mappings to MapForce Server Execution Files

When the target language of a mapping created in MapForce is set to BUILT-IN, it can be executed not only by MapForce, but also by MapForce Server (see About MapForce Server). There are two ways to execute a mapping with MapForce Server:

- If MapForce Server runs in standalone mode (that is, no FlowForce Server is installed), the mapping must be compiled to a server execution file (.mfx), as shown below. You can then run the .mfx file at the command line, using the command run. You can also run the mapping by invoking the run method of the MapForce Server API. For further information, see the MapForce Server documentation (https://www.altova.com/documentation).
- If MapForce Server runs under FlowForce Server management, you can deploy the mapping to a machine where both MapForce Server and FlowForce Server run. For more information about this scenario, see Deploying Mappings to FlowForce Server.

Prerequisites
See Preparing Mappings for Server Execution.

To compile a mapping to a MapForce Server Execution (.mfx) file:

1. Open a mapping in MapForce (for example, myMapping.mfd).
3. Select the folder you want to place the .mfx file in and change the file name if necessary.
4. Click Save. The MapForce Server Execution file myMapping.mfx is generated in the selected folder.

To compile a mapping to a MapForce Server Execution (.mfx) file, using the command line:

- Run MapForce at the command line, and specify the mapping file and the /COMPILE command line option.

For example, the following command compiles the mapping C:\Users\altova\Documents\Altova\MapForce2019\MapForceExamples\SimpleTotal.mfd to a MapForce Server execution file that will be created in the target output directory C:\Users\altova\Desktop.

```
"C:\Program Files (x86)\Altova\MapForce2019\MapForce.exe" "C:\Users\altova\Documents\Altova\MapForce2019\MapForceExamples\SimpleTotal.mfd" /COMPILE "C:\Users\altova\Desktop"
```

See also the MapForce Command Line Interface.

Compiling mappings for a specific MapForce Server version
If your MapForce Server has an older version than MapForce, the former might not be able to execute .mfx files created with a newer version of MapForce, since new features will likely have been added in the meanwhile. In such cases, you can compile the .mfx file for a specific version of MapForce Server, as follows:
1. On the **Tools** menu, click **Options**, and then click **Generation**.
2. Under **Server Execution File**, next to **Generate for MapForce Server version**, select the required MapForce Server version from the drop-down list.

Once you have a newer MapForce Server version, remember to change this option accordingly. If you have no particular reason to compile for a specific version of MapForce Server, select the "most current" option (this is the default option). When this option is selected, the .mfx file is compiled for the most recent version of MapForce Server and could benefit from latest features and improvements which might otherwise not be available in previous versions.

To specify a target MapForce Server version at the command line, run the `/COMPILE` command with the `/MFXVERSION` switch, for example:

```
"C:\Program Files (x86)\Altova\MapForce2019\MapForce.exe" /COMPILE /MFXVERSION:2019
```

See also the **MapForce Command Line Interface**.

**Other options**
Compilation of MapForce Server Execution Files is also affected by the following options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Convert all ADO and ODBC Database Connections to JDBC</strong></td>
<td>If the option is enabled, ADO, ADO.NET, and ODBC database connections are transformed to JDBC using the JDBC driver and the database URL defined in the Database Component Settings dialog box (see <strong>Database Component Settings</strong>). The JDBC connection will be used implicitly if the target machine is a Linux or macOS server.</td>
</tr>
<tr>
<td><strong>Ignore Digital Signatures</strong></td>
<td>This option is applicable only to MapForce Enterprise. It is enabled by default. If the mapping uses XML digital signatures, it skips the digital signature information, since MapForce Server does not support XML digital signatures.</td>
</tr>
</tbody>
</table>

To view or change these options:

- On the **Tools** menu, click **Options**, and then click **Generation**.

These options are also available from the command line interface. See also the **MapForce Command Line Interface**.
11.5 Deploying Mappings to FlowForce Server

Deploying a mapping to FlowForce Server means that MapForce organizes the resources used by the specific mapping into an object and passes it through HTTP (or HTTPS if configured) to the machine where FlowForce Server runs. MapForce mappings are typically deployed to FlowForce Server in order to automate their execution by means of FlowForce Server jobs. Once a mapping is deployed, you can create a full-featured FlowForce Server job from it, and benefit from all job-specific functionality (for example, define custom triggering conditions for the job, expose it as a Web service, and so on).

Note: The term "source machine" refers to the computer where the MapForce is installed and the term "target machine" refers to the computer where FlowForce Server is installed. In the most simple scenario, this is the same computer. In a more advanced scenario, MapForce runs on a Windows machine whereas FlowForce Server runs on a Linux or macOS machine.

The package deployed to FlowForce includes the following:

- The mapping itself. After deployment, the mapping becomes available in the FlowForce Server administration interface as a mapping function (.mapping), at the path you specify. Any source components become input arguments, and any target components become output arguments of this function.

- All kinds of input instance files (XML, CSV, Text, EDI, Excel, JSON, XBRL) that are used by the mapping.
- If the mapping uses EDI and FlexText configurations, these are deployed in the state in which they exist on the source machine at the moment of deployment. If you change the EDI or FlexText configurations after deployment, the mapping must be re-deployed to FlowForce Server.
- If the mapping uses XBRL taxonomies or XBRL taxonomy packages, these are deployed in the state in which they exist on the source machine at the moment of deployment (see also "XBRL Taxonomy Packages" below).

Prerequisites

See Preparing Mappings for Server Execution.
Deploying the mapping to FlowForce Server

1. Ensure that the transformation language is set to BUILT-IN (see Selecting a Transformation Language).

3. Enter your deployment settings (as described below), and click OK. If you selected the Open web browser to create new job check box, the FlowForce Server administration interface opens in the browser, and you can start creating a FlowForce Server job immediately.

The following table lists the mapping deployment settings available on the Deploy Mapping dialog box.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server and Port</td>
<td>Enter the server host name (or IP address) and port of FlowForce Server. These could be localhost and 8082 if FlowForce Server is running on the same machine at the default port. When in doubt, log on to FlowForce Server Web administration interface and check the I.P. address and port displayed in the Web browser's address bar. If you encounter connectivity errors, ensure that the machine on which FlowForce Server runs is configured to allow incoming connections on the designated address and port.</td>
</tr>
</tbody>
</table>
To deploy the mapping through a SSL-encrypted connection, select the **Use SSL** check box. This assumes that FlowForce Server is already configured to accept SSL connections. For more information, refer to FlowForce Server documentation.

### User and Password

The user name and password to be entered depends on the value of the Login drop-down list (see next option). If the Login drop-down list is set to `<Default>` or **Directly**, enter your FlowForce Server user name and password. Otherwise, enter your Windows user name and password, and select the Windows domain name from the Login drop-down list.

### Login

If Windows Active Directory integration is enabled in FlowForce Server, select the Windows domain name from this drop-down list, and enter your Windows credentials in the User and Password fields (see previous option).

### Path

Click **Browse**, and select the path where the mapping function should be saved in FlowForce Server container hierarchy. By default, the path is set to the `/public` container of FlowForce Server.

From the Choose Deployment Name dialog box, you can also create new containers or delete existing containers and mappings, provided that you have the required FlowForce Server permissions and privileges.
Save mapping before deploying | This option is available if you are deploying an unsaved mapping. Select this check box to save the mapping before deployment.

Open browser to create new job | If you select this check box, the FlowForce Server Web administration interface opens in the browser after deployment, and you can start creating a FlowForce Server job immediately.

If the server where you deploy the mapping has multiple versions of MapForce Server running under FlowForce Server management (applicable to Windows servers only), then you are additionally prompted to specify the version of MapForce Server with which you want this mapping to be executed.

Note: The dialog box appears when the FlowForce Server installation directory contains .tool files for each MapForce Server version which runs under FlowForce Server management. By default, a MapForce Server .tool file is added automatically to this directory when you install MapForce Server as part of FlowForce Server installation. The path where the .tool files are stored in FlowForce is: C:\Program Files\Altova\FlowForceServer2019\tools. If you have additional versions of MapForce Server which you want to run under FlowForce Server management, their .tool files may need to be copied manually to the directory above. The .tool file of MapForce Server can be found at: C:\Program Files\Altova\MapForceServer2019\etc.
AS2 Integration

AS2 (Applicability Statement 2) is a specification that enables exchanging files securely over the Internet. AS2 is used by businesses to exchange primarily EDIINT (EDI over Internet) and XML files through either HTTP or HTTPS.

FlowForce Server Advanced Edition provides the functionality required to send AS2 messages to trading partners, or receive AS2 from trading partners. In addition, FlowForce Server is capable of processing AS2 data and storing it locally, with the help of its built-in set of functions. For even more advanced needs, if you need to prepare AS2 data from some existing source (for example, a database), or convert it to other formats, or send it to some Web service, you can also include MapForce and MapForce Server into the AS2 process.

There are two main scenarios where MapForce and MapForce Server are necessary:

1. To map or generate data in any format supported by MapForce (such as XML, XBRL, Excel, databases, Web services), before sending it to AS2 partners.
2. To transform data received from AS2 partners in a variety of ways (for example, convert it to Excel, convert it to a different XML schema, store it in a database, send it to a Web service, and so on).

Generating and sending AS2 data

In a scenario where you need to prepare or generate AS2 data with MapForce before sending it to partners, the high-level process looks as follows:
Generating and sending AS data

In the diagram above, both MapForce Server and FlowForce Server must be installed on the same machine (it can be a Windows, Linux, or macOS operating system). MapForce may run on the same machine as MapForce Server and FlowForce Server (provided that it's a Windows machine), or on a different machine that can connect to FlowForce Server via HTTP or HTTPS. The AS2 partner is a remote server with which FlowForce Server communicates through HTTP(S).

The AS2 process illustrated above works as follows:

<table>
<thead>
<tr>
<th>Step #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Design and test the EDI/XML data mapping</td>
</tr>
<tr>
<td></td>
<td>With MapForce, you can design a data mapping transformation that takes as input data in various formats</td>
</tr>
</tbody>
</table>
### Step # | Description
--- | ---

1. **Design the mapping**

   Designing a mapping for EDI purposes is not different to other mappings, and various such examples are included in MapForce documentation, see the EDI chapter. While you design the mapping, you can validate and preview the mapping output directly in MapForce, by clicking the Output tab. To ensure that the mapping is suitable for execution in a server environment, you will need to design and test it for the BUILT-IN transformation language, see also Selecting a Transformation Language.

2. **Deploy mapping to FlowForce Server**

   FlowForce Server automates various tasks by means of on demand or scheduled jobs that can be defined from a Web interface. FlowForce Server can also automate the execution of a mapping designed with MapForce, provided that MapForce Server runs under FlowForce Server management. Once the MapForce mapping produces the required output, you are ready to automate its execution, by deploying it to FlowForce Server. For instructions, see Deploying Mappings to FlowForce Server.

3. **Configure AS2 partner and certificates**

   To set up the communication with AS2 partners, you will need to obtain their AS2 connectivity details (such as URI and AS2 name), and exchange certificates. The certificates must be imported (and partner details must be entered) into FlowForce Server.

4. **Create a job**

   A FlowForce job must be created in order to (a) run the mapping and produce the required output, and (b) send the AS2 message (see also step 7). These two actions may be either execution steps of the same job, or two different jobs altogether.

5. **Run job**

   The FlowForce Server job created in the previous step may be configured to run in various ways, depending on your business needs. For example, it can run as a Web service call, or whenever a file changes on the file system, or it could be scheduled to occur at a specific time and date. This step is fully automated.

6. **Run data mapping**

   This step also takes place automatically and is executed by MapForce Server. If a job is configured to execute a data mapping (be it scheduled or on demand), an internal call to MapForce Server takes place. As a result, MapForce Server runs the mapping and returns the output to FlowForce Server.

7. **Pick output and send AS2 message**

   In order to send the AS2 message, your job (or execution step within a job) must call the FlowForce Server built-in function `/system/as2/send`. This function takes a number of
### Receiving and processing AS2 data

If your organization receives AS2 data from trading partners, you can additionally configure a data receiving workflow. In this scenario, your organization would be able to not only receive and store AS2 data, but also transform it to other formats, save it to a database, or send it to another Web service. For example, you could receive files in EDI or XML format from AS2 trading partners and then supply them as input to some mapping that runs as a recurrent FlowForce Server job. In this scenario, an example AS2 process looks as follows:

<table>
<thead>
<tr>
<th>Step #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>parameters required to send the AS2 message, including the partner object configured in step 3, the partner's URI, and the AS2 message content that you want to send. Your job may also need to call various FlowForce Server AS2 expression functions in order to convert the mapping output to the required form (for example, from a file to a stream).</td>
</tr>
<tr>
<td>8.</td>
<td>Partner replies with synchronous MDN</td>
</tr>
</tbody>
</table>
Receiving and processing AS2 data

The example AS2 process illustrated above works as follows:

<table>
<thead>
<tr>
<th>Step #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 2, 3</td>
<td>These are the same steps as in the previous table. The only difference is that this time the mapping is expected to take as input some file that your organization expects to receive from an AS2 trading partner (for example, an EDI or XML file).</td>
</tr>
<tr>
<td>4. Create a job (AS2 service)</td>
<td>This is a one-time step. In this step, you create a FlowForce Server job that exposes an AS2 service. The AS2 service listens for requests from your AS2 partners at a configured HTTP(S) address and port.</td>
</tr>
<tr>
<td>Step #</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>5.</td>
<td>Send AS2 data</td>
</tr>
<tr>
<td></td>
<td>In this step, a trading partner submits AS2 messages to the AS2 service. For communication to be successful, the partner's AS2 name and certificates must already be defined in FlowForce Server.</td>
</tr>
<tr>
<td>6.</td>
<td>Reply with synchronous MDN</td>
</tr>
<tr>
<td></td>
<td>FlowForce Server replies to the AS2 partner with a synchronous MDN that indicates the outcome of the operation (success or error).</td>
</tr>
<tr>
<td>7.</td>
<td>Process and save data</td>
</tr>
<tr>
<td></td>
<td>As soon as there is an incoming message, a FlowForce Server job converts the received data to a string or a file, and then stores it in some directory, or passes it to another job as argument. The exact processing logic is configurable with the help of FlowForce Server built-in and expression functions.</td>
</tr>
<tr>
<td>8.</td>
<td>Run data mapping</td>
</tr>
<tr>
<td></td>
<td>The FlowForce Server job that receives AS2 data may optionally invoke the data mapping job that was created in the first step. The mapping job takes as input the AS2 data received from the partner and then processes it in any of the ways supported by MapForce: for example, transforms it to another format, saves it to a database, sends it to another Web service, and so on.</td>
</tr>
</tbody>
</table>

11.7 MapForce Command Line Interface

The general syntax of a MapForce command at the command line is:

```
MapForce.exe <filename> [/{target} [[<outputdir>] [/options]]]
```

Legend

The following notation is used to indicate command line syntax:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text without brackets or braces</td>
<td>Items you must type as shown</td>
</tr>
<tr>
<td>&lt;Text inside angle brackets&gt;</td>
<td>Placeholder for which you must supply a value</td>
</tr>
<tr>
<td>[Text inside square brackets]</td>
<td>Optional items</td>
</tr>
<tr>
<td>{Text inside braces}</td>
<td>Set of required items; choose one</td>
</tr>
<tr>
<td>Vertical bar (</td>
<td>)</td>
</tr>
<tr>
<td>Ellipsis (...)</td>
<td>Items that can be repeated</td>
</tr>
</tbody>
</table>

`<filename>`

The mapping design (.mfd) or mapping project (.mfp) file from which code is to be generated. To generate code for the whole project, set the target `/GENERATE` (see description below) and enter the project path as `<filename>`, for example, `MapForceExamples.mfp`.

`{/target}`

Specifies the target language or environment for which code is to be generated. The following code generation targets are supported.

<table>
<thead>
<tr>
<th>Target</th>
<th>Description</th>
</tr>
</thead>
</table>
| `/COMPILE[:compileoptions]`   | Compiles a mapping to a MapForce Server execution file (.mfx). Optionally, the following options can be supplied, delimited by a comma:  

    - `JDBC` Transforms all database connections to JDBC using the JDBC driver and the database URL defined in the Database Component Settings dialog box, see also Database Component Settings.  
    - `NOXMLSIGNATURES` Suppresses the generation of digital signatures in the MapForce Server Execution file (note that digital signatures are optional in MapForce Enterprise Edition). |
<table>
<thead>
<tr>
<th>Target</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/GENERATE</td>
<td>Generates project code for all mappings in the project file using the current folder settings, see Managing Project Folders. If you select this target, make sure to supply a MapForce project (.mfp file) as <code>&lt;filename&gt;</code>.</td>
</tr>
<tr>
<td>/XSLT</td>
<td>Generates XSLT 1.0 code.</td>
</tr>
<tr>
<td>/XSLT2</td>
<td>Generates XSLT 2.0 code.</td>
</tr>
<tr>
<td>/XQuery</td>
<td>Generates XQuery code.</td>
</tr>
<tr>
<td>/JAVA</td>
<td>Generates Java code.</td>
</tr>
<tr>
<td>/CS</td>
<td>Generates C# code. This command also optionally allows setting specific options for code generation, namely:</td>
</tr>
<tr>
<td></td>
<td>`/CS[:(VS2008</td>
</tr>
<tr>
<td>VS2008</td>
<td>Visual Studio 2008</td>
</tr>
<tr>
<td>VS2010</td>
<td>Visual Studio 2010</td>
</tr>
<tr>
<td>VS2013</td>
<td>Visual Studio 2013</td>
</tr>
<tr>
<td>VS2015</td>
<td>Visual Studio 2015</td>
</tr>
<tr>
<td>VS2017</td>
<td>Visual Studio 2017</td>
</tr>
<tr>
<td>If no Visual Studio version is specified, code will be generated using the Visual Studio version defined in the code generation options, see Code Generator Options.</td>
<td></td>
</tr>
<tr>
<td>/CPP</td>
<td>Generates C++ code. This command also optionally allows setting specific code generation options, namely:</td>
</tr>
<tr>
<td></td>
<td>`/CPP[:(VS2008</td>
</tr>
<tr>
<td>The first option group set the target Visual Studio version. Valid values:</td>
<td></td>
</tr>
<tr>
<td>VS2008</td>
<td>Visual Studio 2008</td>
</tr>
<tr>
<td>VS2010</td>
<td>Visual Studio 2010</td>
</tr>
<tr>
<td>VS2013</td>
<td>Visual Studio 2013</td>
</tr>
<tr>
<td>VS2015</td>
<td>Visual Studio 2015</td>
</tr>
<tr>
<td>Target</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>VS2017</td>
<td>Visual Studio 2017</td>
</tr>
</tbody>
</table>

The second option group specifies the XML library targeted by the generated code. Valid values:

- **MSXML**: Generate code for MSXML 6.0
- **XERCES3**: Generate code for Xerces 3

The third option group specifies whether static as opposed to dynamic libraries should be generated. Valid values:

- **LIB**: Generate static LIB libraries
- **DLL**: Generate DLL libraries

The fourth option group specifies whether code should be generated with or without MFC support. Valid values:

- **MFC**: Enable MFC support
- **NoMFC**: Disable MFC support

If the options above are not specified, code will be generated using the Visual Studio version defined in the code generation options, see Code Generator Options.

<outputdir>
Optional parameter which specifies the output directory. If an output path is not supplied, the current working directory will be used. Note that any relative file paths are relative to the current working directory.

When target is `/GENERATE` and the `<outputdir>` parameter is not set, the code generation language, as well as the output path of each mapping, are supplied by the settings defined for each folder inside the project, see Managing Project Folders.

When target is `/GENERATE` and the `<outputdir>` parameter is set, the `<outputdir>` value supplied at the command line takes precedence over the output directory defined at the root project level. It does not take precedence, however, over the code generation settings defined at each folder inside the project.

/options
The /options are not mutually exclusive. One or more of the following options can be set.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![MFXVERSION][&lt;version&gt;]</td>
<td>This option is applicable if the target is !COMPILE!. It compiles the MapForce Server Execution (.mfx) file for a particular version of MapForce Server. You can supply as value any version of MapForce Server, starting with 2013r2 onwards, up to the current MapForce version. See also Compiling mappings for a specific MapForce Server version.</td>
</tr>
<tr>
<td>/GLOBALRESOURCEFILE</td>
<td>This option is applicable if the mapping uses Global Resources to resolve input or output file or folder paths, or databases. For more information, see Altova Global Resources.</td>
</tr>
<tr>
<td>&lt;filename&gt;</td>
<td>The option /GLOBALRESOURCEFILE specifies the path to a Global Resource .xml file. Note that, if /GLOBALRESOURCEFILE is set, then /GLOBALRESOURCECONFIG must also be set.</td>
</tr>
<tr>
<td>/GLOBALRESOURCECONFIG</td>
<td>This option specifies the name of the Global Resource configuration (see also the previous option). Note that, if /GLOBALRESOURCEFILE is set, then /GLOBALRESOURCECONFIG must also be set.</td>
</tr>
<tr>
<td>&lt;config&gt;</td>
<td></td>
</tr>
<tr>
<td>/LIBRARY &lt;libname&gt; (...)</td>
<td>Use together with a code generation target language to specify additional function libraries. This option can be specified more than once to load multiple libraries. These libraries are temporarily (for this one run) added to the libraries from Tools</td>
</tr>
<tr>
<td>/LOG &lt;logfilename&gt;</td>
<td>Generates a log file at the specified path. &lt;logfilename&gt; can be a full path name, for example, it can include both a directory and a file name. However, if a full path is supplied, the directory must exist for the log file to be generated. If you specify only the file name, then the file will be placed in the current directory of the Windows command prompt.</td>
</tr>
</tbody>
</table>

Remarks

- Relative paths are relative to the working directory, which is the current directory of the application calling MapForce. This applies to the path of the .mfd filename, .mfp filename, output directory, log filename, and global resource filename.
- Do not use the end backslash and closing quote at the command line (for example, "C: \My directory\"). These two characters are interpreted by the command line parser as a literal double quotation mark. Use the double backslash \ \ if spaces occur in the command line and you need the quotes ("c:\\My Directory\"), or try to avoid using spaces and therefore quotes at all.

Examples

1) To start MapForce and open the mapping <filename>.mfd, use:
MapForce.exe <filename>.mfd

2) To generate XSLT 2.0 code and also create a log file with the name <logfilename>, use:

MapForce.exe <filename>.mfd /XSLT2 <outputdir> /LOG <logfilename>

3) To generate XSLT 2.0 code taking into account the global resource configuration <grconfigname> from the global resource file <grfilename>, use:

Mapforce.exe <filename>.mfd /XSLT2 <outputdir> /GLOBALRESOURCEFILE <grfilename> /GLOBALRESOURCECONFIG <grconfigname>

4) To generate a C# application for Visual Studio 2015 and output a log file, use:

MapForce.exe <filename>.mfd /CS:VS2015 <outputdir> /LOG <logfilename>

5) To generate a C++ application using the code generation settings defined in Tools | Options, and output a log file, use:

MapForce.exe <filename>.mfd /CPP <outputdir> /LOG <logfilename>

6) To generate a C++ application for Visual Studio 2015, MSXML, with static libraries, MFC support, and no log file, use:

MapForce.exe <filename>.mfd /CPP:VS2015,MSXML,LIB,MFC

7) To generate a C++ application for Visual Studio 2015, Xerces, with dynamic libraries, no MFC support, and a log file, use:

MapForce.exe <filename>.mfd /CPP:VS2015,XERCES,DLL,NoMFC <outputdir> /LOG <logfilename>

8) To generate a Java application and also output a log file, use:

MapForce.exe <filename>.mfd /JAVA <outputdir> /LOG <logfilename>

9) To generate code for all mappings in the project, using the language and output directory defined in the folder settings (of each folder inside the project), use:

MapForce.exe <filename>.mfp /GENERATE /LOG <logfilename>

10) To generate Java code for all mappings in the project file, use:

MapForce.exe <filename>.mfp /JAVA /LOG <logfilename>

Note that the code generation language defined in the folder settings are ignored, and Java is used
for all mappings.

11) To supply input and output files at the command line for a previously compiled Java mapping, use:

```
java -jar <mappingfile>.jar /InputFileName <inputfilename> /OutputFileName <outputfilename>
```

The /InputFileName and /OutputFileName parameters are the names of special input components in the MapForce mapping that allow you to use parameters in command line execution (see Supplying Parameters to the Mapping).

12) To compile a mapping to a MapForce Server execution file, for MapForce Server version 2019, and suppress XML signatures:

```
MapForce.exe <filename>.mfd /COMPILE:NOXMLSIGNATURES <outputdir> /MFXVERSION:2019 /LOG <logfilename>
```
Chapter 12

Customizing MapForce
12 Customizing MapForce

This section provides information about working with Altova Global Resources, customizing the mapping output, generating and customizing mapping documentation, and working with catalog files.
12.1 Changing the MapForce Options

You can change the general and other preferences in MapForce as follows:

- On the **Tools** menu, click **Options**.

The available options are grouped as shown below.

**Libraries**
From this page, you can add or delete custom function libraries to MapForce. For more information, see [Importing Custom XSLT 1.0 or 2.0 Functions](#), [Importing Custom XQuery 1.0 Functions](#), [Importing Custom Java and .NET Libraries](#).

**General**
The settings available in this page are as follows:

<table>
<thead>
<tr>
<th>Show logo</th>
<th>Show on start</th>
<th>Shows or hides an image (splash screen) while MapForce starts.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show gradient background</td>
<td>Enables or disables the gradient background in the Mapping pane.</td>
<td></td>
</tr>
<tr>
<td>Limit annotation display to N lines</td>
<td>This option applies to components which support annotations (for example, XML schema, EDI). If the annotation text contains multiple lines, then enabling this option shows only the first N lines on the component, where N is the value you specify. This setting also applies to SELECT statements visible in a component.</td>
<td></td>
</tr>
<tr>
<td>Encoding name</td>
<td>Sets the default character encoding for new components. This setting can also be changed individually for each component, see Changing the Component Settings.</td>
<td></td>
</tr>
<tr>
<td>Use execution timeout</td>
<td>Sets an execution timeout when previewing the mapping result in the <strong>Output</strong> pane.</td>
<td></td>
</tr>
<tr>
<td>Generate output to temporary files</td>
<td>When this option is set, the output generated when you preview the mapping result will be written to temporary files (this is the default option). If the output file path contains folders that do not exist yet, MapForce will create these folders. <strong>Warning:</strong> If you intend to deploy the mapping to a server for execution, any directories in the path must exist on the server; otherwise, an execution error will occur. See also Preparing Mappings for Server Execution.</td>
<td></td>
</tr>
<tr>
<td>Write directly to final output files</td>
<td>When this option is set, the output generated when you preview the mapping result will be written to actual files. If the output file path contains folders that do not exist yet, then a mapping error occurs. <strong>Warning:</strong> This option overwrites any existing output files.</td>
<td></td>
</tr>
</tbody>
</table>
Customizing MapForce

Changing the MapForce Options

<table>
<thead>
<tr>
<th>Show logo</th>
<th>Show on start</th>
<th>Shows or hides an image (splash screen) while MapForce starts. without requesting further confirmation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display text in steps of N million characters</td>
<td>Specifies the maximum size of the text displayed in the Output pane when you preview mappings that generate large XML and text files. If the output text exceeds this value, you will need click a Load more button to load the next chunk. For more information, see Previewing the Output.</td>
<td></td>
</tr>
</tbody>
</table>

**Editing**

The settings available in this page are as follows:

<table>
<thead>
<tr>
<th>Align components on mouse dragging</th>
<th>Specify whether components or functions should be aligned with other components, while you drag them with the mouse, see Aligning Components.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smart component deletion</td>
<td>When enabled, this option &quot;remembers&quot; connections of deleted components, see Keeping Connections After Deleting Components.</td>
</tr>
</tbody>
</table>

**Messages**

From this page, you can re-enable message notifications that were previously disabled using the "Do not show this message again" option.

**Generation**

From this page, you can define settings applicable when you generate program code and MapForce Server Execution files. For more information, see Code Generator Options and Compiling Mappings to MapForce Server Execution Files, respectively.

**XBRL**

See General XBRL Options.

**Debugger**

See Debugger Settings.

**Database**

From this page, you can define settings applicable when querying databases in the DB Query tab (see Database Query Settings).

**Network proxy**

See Network Proxy Settings.
12.1.1 Java Settings

On the **Java** tab, you can optionally enter the path to a Java VM (Virtual Machine) on your file system. Note that adding a custom Java VM path is not always necessary. By default, MapForce attempts to detect the Java VM path automatically by reading (in this order) the Windows registry and the JAVA_HOME environment variable. The custom path added on this dialog box will take priority over any other Java VM path detected automatically.

You may need to add a custom Java VM path, for example, if you are using a Java virtual machine which does not have an installer and does not create registry entries (for example, Oracle's OpenJDK). You might also want to set this path if you need to override, for whatever reason, any Java VM path detected automatically by MapForce.

Note the following:

- The Java VM path is shared between Altova desktop (not server) applications. Consequently, if you change it in one application, it will automatically apply to all other Altova applications.
- The path must point to the `jvm.dll` file from the `\bin\server` or `\bin\client` directory, relative to the directory where the JDK was installed.
- The MapForce platform (32-bit, 64-bit) must be the same as that of the JDK.
- After changing the Java VM path, you may need to restart MapForce for the new settings to take effect.

Changing the Java VM path affects the following areas:

- JDBC connectivity
- Java extension functions for XSLT/XPath

This setting does not affect Java code generation.
12.1.2 Network Proxy Settings

The **Network Proxy** section enables you to configure custom proxy settings. These settings affect how the application connects to the Internet (for XML validation purposes, for example). By default, the application uses the system's proxy settings, so you should not need to change the proxy settings in most cases. If necessary, however, you can set an alternative network proxy using the options below.

**Note:** The network proxy settings are shared between all Altova MissionKit applications. Consequently, if you change the settings in one application, they will automatically affect all other applications.

**Use system proxy settings**
Uses the Internet Explorer (IE) settings configurable via the system proxy settings. It also queries the settings configured with `netsh.exe winhttp`.

**Automatic proxy configuration**

The following options are provided:

- **Auto-detect settings**: Looks up a WPAD script (`http://wpad.LOCALDOMAIN/wpad.dat`) via DHCP or DNS, and uses this script for proxy setup.
- **Script URL**: Specify an HTTP URL to a proxy-auto-configuration (.pac) script that is to be used for proxy setup.
- **Reload:** Resets and reloads the current auto-proxy-configuration. This action requires Windows 8 or newer, and may need up to 30s to take effect.

**Manual proxy configuration**

Manually specify the fully qualified host name and port for the proxies of the respective protocols. A supported scheme may be included in the host name (for example: `http://hostname`). It is not required that the scheme is the same as the respective protocol if the proxy supports the scheme.

The following options are provided:

- **Use this proxy for all protocols:** Uses the host name and port of the HTTP Proxy for all protocols.
- **No Proxy for:** A semi-colon (`;`) separated list of fully qualified host names, domain names, or IP addresses for hosts that should be used without a proxy. IP addresses may not be truncated and IPv6 addresses have to be enclosed by square brackets (for example: `[2606:2800:220:1:248:1893:25c8:1946]`). Domain names must start with a leading dot (for example: `example.com`).
- **Do not use the proxy server for local addresses:** If checked, adds `<local>` to the No Proxy for list. If this option is selected, then the following will not use the proxy: (i) `127.0.0.1`, (ii) `::1`, (iii) all host names not containing a dot character `.`

**Note:** If a proxy server has been set and you want to deploy a mapping to Altova FlowForce Server, you must select the option *Do not use the proxy server for local addresses.*

**Current proxy settings**

Provides a verbose log of the proxy detection. It can be refreshed with the **Refresh** button to the right of the **Test URL** field (for example, when changing the test URL, or when the proxy settings have been changed).

- **Test URL:** A test URL can be used to see which proxy is used for that specific URL. No I/O is done with this URL. This field must not be empty if proxy-auto-configuration is used (either through **Use system proxy settings** or **Authomatic proxy configuration**).
12.2 Altova Global Resources

Altova Global Resources represent a way to refer to files, folders, or databases so as to make these resources reusable, configurable and available across multiple Altova applications. For example, let's assume that several MapForce mappings routinely read data from the same XML file which is critical for your business workflow. If this file has been renamed on the disk for whatever reason, this would cause "file not found" errors in multiple contexts, and break the workflow. To prevent such issues, it is possible to create a so-called "file alias" (in other words, a Global Resource), and change all mappings to refer to this Global Resource instead of the actual file on disk. This way, if the file name ever changes, you would only need to change the file alias, in one place.

Global Resources can be defined and shared between the following Altova desktop applications: Authentic, DatabaseSpy, MobileTogether Designer, MapForce, StyleVision, and XMLSpy. On the server side, Global Resources can be consumed by the following Altova server applications: MapForce Server, MapForce Server Advanced Edition, RaptorXML Server, RaptorXML+XBRL Server.

Global Resources (be they file, folder, or database references) can be used in MapForce for various scenarios, for example:

- To supply a configurable file path as mapping input, see Example: Run Mappings with Variable Input Files.
- To redirect the mapping output to a configurable path. For more information, see Example: Generate Mapping Output to Variable Folders.
- To supply a configurable path to a StyleVision Power Stylesheet (.sps) file, if one is used by the mapping. For example, instead of referencing a plain StyleVision .sps file from the MapForce component settings, you could refer to an .sps file previously defined as a Global Resource, which has two possible configurations (let's say, "Website.sps" and "Print.sps"). See also Styling Mapping Output with StyleVision.
- To reuse a database connection. If a database has already been defined as a Global Resource (in any Altova application), you can connect to it without going through all the set-up steps again, see Using a Connection from Global Resources.
- To easily switch the database from which the mapping reads data, or the one to which the mapping writes data (provided that two or more databases have the same structure but different data, see Example: Switch Databases).

Note:
- FlowForce Server does not support Global Resources. MapForce Server can consume Global Resources either at the command line or at API level.
- MapForce Basic Edition does not support consuming database connections defined as Global Resources.

12.2.1 Creating Global Resources

A Global Resource alias is a reusable reference which represents a file or folder path, or a database connection. Aliases are defined only once and can be reused as many times as necessary in contexts which support them, including across multiple Altova applications. Taking databases as example, if you frequently work with a specific database in more than one Altova
application, then it is a good idea to add the database connection as a Global Resource. This way, you wouldn't need to go through all the Database Connection Wizard steps each time when you need to connect to the same database from another Altova application.

File, folder, and database aliases are configurable in their turn, by means of so-called "configurations". Configurations make it possible to easily switch between files, folders and databases that are consumed or produced by Altova applications, which is particularly useful for testing scenarios. For example, you could create a database alias that consists of three separate connections to the same database, each with a different driver kind: (a) ODBC, the default connection kind, (b) JDBC, and (c) ADO.NET. This way, to connect to the database with a specific driver, you would just select the corresponding configuration from the Global Resources drop-down list before running the mapping.

Global Resources drop-down list

Configurations can also help you generate mapping output to variable folders, with a click of a button. For example, you could create a folder alias with two configurations: (a) "Testing", which points to directory C:\Testing and (b) "Production", which points to directory C:\Production. It is then possible to configure a mapping to generate output to either C:\Testing or C:\Production folders, just by selecting the required configuration from the Global Resources drop-down list before running the mapping. This example is discussed in more detail in Example: Generate Output to Variable Folders.

How to create a Global Resource alias

1. On the Tools menu, click Global Resources. (Alternatively, click the Global Resource toolbar button.)
2. Click Add and select the resource type you wish to create (file, folder, database).
3. Enter a descriptive name for this alias in the Resource alias text box (for example, "MappingInputFile", "MappingOutputFolder", "DatabaseConnection").
4. Set up the "Default" configuration:
   a) If it's a file or folder, browse for the file or folder to which this resource should point by default.
   b) If it's a database connection, click Choose Database and follow the Database Connection Wizard to connect to the database (see Connecting to a Database). This database connection will be used by default when the mapping runs (unless a different configuration is explicitly selected from the Global Resources drop-down list or supplied as a command line parameter in server execution).
5. Optionally, if the resource should have an additional configuration (for example, a driver kind in case of databases, or an alternative path in case of files or folders), click the Add configuration button, enter a descriptive name (for example "ProductionFolder" or "JDBC_Alternative"), and set it up as follows:
   a) If it's a file or folder, browse for the file or folder to which this resource should point as an alternative to the default configuration defined in previous step.
   b) If it's a database connection, follow the Database Connection Wizard to connect to the database. This database connection will be used as an alternative to the default one.
In some cases, it might be more convenient to create a configuration as a copy of the default configuration, and then edit it. In this case, click the **Add configuration as a copy of the currently selected configuration** button.

6. Repeat the previous step for each additional configuration required.

### 12.2.2 Databases as Global Resources

When you add a database connection as a Global Resource, the database connection parameters are automatically populated on the Global Resource dialog box.

![Global Resource dialog box](image)

On the Global Resource dialog box, it is possible to edit some of the database connection parameters. As illustrated above, the parameters are grouped into two categories:

<table>
<thead>
<tr>
<th>Database</th>
<th>MapForce-specific execution parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>These parameters are shared between Altova applications. In MapForce, they are used at design time, that is, when the mapping is loaded, or when you click the <strong>Output</strong> tab in MapForce to preview the mapping.</td>
<td>These parameters are applicable when you generate program code or compile a mapping to MapForce Server</td>
</tr>
</tbody>
</table>
execution file (.mfx). They are used at mapping runtime, as follows:

- In generated C++, C#, or Java program code.
- If you compiled the mapping to a MapForce Server execution file, and automatic JDBC conversion took place. For more information about automatic JDBC conversion, see Database mappings in various execution environments.

If a mapping uses a Global Resource to connect to a database, then the database connection details visible in the Global Resource dialog box take precedence over those defined on the mapping. In the Component Settings dialog box, illustrated below, notice that the database connection settings become grayed out. The dialog box also informs you that the connectivity parameters are defined as a Global Resource.

To change the database component to connect to the database directly (without using Global Resources), click Change, and follow the wizard steps to reconnect to the database.

### 12.2.3 MapForce and StyleVision Transformation Result as Global Resource

It is also possible to create Global Resources which, instead of pointing to a static file, read a specific file produced by either a MapForce mapping or StyleVision transformation. In this case, the Altova application which consumes the Global Resource will first call either MapForce or
StyleVision, run the corresponding mapping or transformation, and finally fetch the resulting file. This makes it possible to define data workflows between Altova applications (for example, pass the result of a MapForce mapping or StyleVision transformation as input to another mapping or transformation). For an example which illustrates how XMLSpy consumes the result (output) of a MapForce mapping using Global Resources, see Example: Create Application Workflow.

**Note:**
- In order to make a mapping result (output) available as a Global Resource, either the transformation language of the mapping must be set to BUILT-IN, or the mapping must contain only components which are supported by the BUILT-IN language (for example, some XSLT functions are not supported by the BUILT-IN language.)
- MapForce Basic Edition does not support providing mapping transformation results as Global Resources.

### 12.2.4 The Global Resources XML File

By default, all Global Resources, regardless of the Altova application where they were created, are stored at the following path: `C:\Users\<username>\Documents\Altova\GlobalResources.xml`. This makes them transparent, easy to backup, as well as portable to other workstations where Altova products are installed. It is also possible to rename or duplicate the `GlobalResources.xml` file and thus create multiple Global Resource files. However, only one Global Resource file can be active at a time in an Altova application.

**To set up the active Global Resource file:**

1. On the **Tools** menu, click **Global Resources**. (Alternatively, click the **Global Resource** toolbar button.)
2. Click **Browse** and select the required Global Resource XML file.

If you are using multiple Global Resource files, make sure that the currently active Global Resource file contains all Global Resources required to run the mapping. For example, if a mapping was configured to read data from a path using a Global Resource, then the currently active Global Resource file must contain that specific Global Resource. Otherwise, error messages like “Errors resolving global resource” will occur in the **Messages** window.

### 12.2.5 Global Resources in Various Execution Environments

A mapping using Global Resources will behave differently in each environment where it is run, as shown below.

**Global Resources in XSLT, XSLT2, XQuery**

When you generate XSLT or XSLT2 code and the mapping uses Global Resources, this does not affect the generated XSLT stylesheet in any way. With or without Global Resources, the input and output files are not a permanent assignment and can be specified flexibly anyway when you run the XSLT stylesheet in your XSLT processor. The same applies for generated XQuery code.
An exception to this is the **DoTransform.bat** file generated for RaptorXML execution. Any Global Resources used by the mapping will be resolved to actual paths in **DoTransform.bat**, taking into account the value (configuration) which is currently selected from the Global Resource drop-down list.

For information about supplying Global Resources to RaptorXML, refer to the RaptorXML documentation (see [https://www.altova.com/documentation.html](https://www.altova.com/documentation.html)).

**Global Resources in C#, C++, Java**

When you generate C#, C++, or Java program code, any Global Resources used by the mapping will be resolved. For example, a file or folder alias defined as Global Resource will be converted to the actual file or folder path. If a particular Global Resource configuration is selected from the Global Resources drop-down list, then the code will be generated for the selected configuration.

The **Messages** window provides information as to how exactly a Global Resource was resolved, for example:

To generate code for a particular Global Resource configuration, select it from the Global Resource drop-down list before generating code. Alternatively, if you generate code from the command line, supply the **GLOBALRESOURCEFILE** and **GLOBALRESOURCECONFIG** parameters at the command line (see also [MapForce Command Line Interface](https://www.altova.com/documentation.html)).

It is not possible to switch or refer to Global Resources from generated code (instead, you can modify the code to change the input or output file path).

**Note:** In C# or Java, you can change not only the path but also the data type of input or output, see [Changing the data type of the mapping input/output (C#, Java)](https://www.altova.com/documentation.html).

**Global Resources in MapForce Server**

When you compile a mapping to a MapForce Server execution file (.mfx), any Global Resources references used by the mapping are preserved as such. In MapForce Server, the following is required to run an .mfx file compiled from a mapping which uses Global Resources:

1. The path to the Global Resource XML file (that is, the file where Global Resources are defined, see [The Global Resources XML File](https://www.altova.com/documentation.html)).
2. The Global Resource configuration name. The name of the default configuration is "Default". If you created additional configurations, as explained in [Creating Global Resources](https://www.altova.com/documentation.html), then the desired configuration must be called by its corresponding name.

The Global Resource file path and the name of the configuration can be specified as follows:
If you run the mapping through the command line interface, set the options `--globalresourceconfig` and `--globalresourcefile` after the `run` command, for example:

```
C:\Program Files (x86)\Altova\MapForceServer2019\bin\MapForceServer.exe run SomeMapping.mfx --globalresourcefile="C:\Users\me\Documents\Altova\GlobalResources.xml" --globalresourceconfig="Default"
```

If you run the mapping through the MapForce Server API, call the method `SetOptions` two times before calling the `run` method. The first call is required to supply the Global Resource XML file path as option, and the second one is required to supply the Global Resource configuration name.

For more information, refer to the MapForce Server documentation (see https://www.altova.com/documentation.html).

### 12.2.6 Example: Run Mapping with Variable Input Files

Let's assume that, as part of your job duties, you frequently run a mapping that takes as input an XML file. Under normal circumstances, whenever you want to change the input XML of the mapping, you can open the properties of the source XML component and browse for the new input file, see Changing the Component Settings. This is easy to accomplish if it's a one time task. However, what if you need to change the input XML file of the mapping multiple times per day, or even per hour? For example, every morning you need to run the mapping and generate a report by using one XML file as mapping input, and every evening the same report must be generated from another XML file. This is where Global Resources can help you: instead of editing the mapping multiple times per day (or keeping multiple copies of it), you could configure the mapping to read from a file defined as a global resource (a so-called "file alias").

To address the requirement laid out in this example, the file alias could be configured to have two configurations:

1. "Default" - This configuration would supply a "morning" XML file as mapping input
2. "EveningReports" - This configuration would supply an "evening" XML file as mapping input.

Having these configurations in place would make it possible to run the mapping with either input file. Once the file alias is set up as shown below, you will be able to select the desired configuration from a drop-down list, before running the mapping.

#### Step 1: Create the Global Resource

The file alias can be created as follows:

1. On the Tools menu, click Global Resources. (Alternatively, click the Global Resource toolbar button.)
2. Click Add | File.
3. Enter a name in the Resource alias text box (in this example, "DailyReports" would be an appropriate name).
4. Click Browse and select the following file: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\mf-ExpReport.xml`.
5. Click Add Configuration and name it "EveningReports".
6. Click Browse and this time select the following file: `<Documents>\Altova\GlobalResources.xml`.
7. Set the configuration option to "Default".
8. Set the Resource alias to "DailyReports".
9. Set the Global Resource XML file path to "C:\Users\me\Documents\Altova\GlobalResources.xml".
10. Set the Global Resource configuration name to "Default".
11. Click OK.

This configuration will allow you to select either the "morning" or "evening" XML file as input, depending on your needs.
Step 2: Use the Global Resource in the mapping

The required Global Resource has now been created; however, the mapping is not using it yet. To change the mapping so that it reads from the previously defined file alias (Global Resource), do the following:

1. Open the following mapping `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\Tut-ExpReport.mfd`.
2. Right-click the header of the source component on the mapping, and select Properties from the context menu.
3. Next to Input XML file, click Browse.
4. Click Switch to Global Resources and select the file alias "DailyReports" defined previously.
5. Click Open. The input XML file path has now become `altova://file_resource/DailyReports`, which indicates that the path uses a Global Resource.

Step 3: Run the mapping with the desired configuration

You can now easily switch the input XML file before running the mapping, as follows:

- On the Tools menu, click Active Configuration | Default, to use the file `mf-ExpReport.xml` as input.
- On the Tools menu, click Active Configuration | EveningReports, to use the file `mf-ExpReport2.xml` as input.

Alternatively, select the required configuration from the Global Resources drop-down list.

To preview the mapping result with either configuration, click the Output tab and observe differences in the generated output.
12.2.7 Example: Generate Output to Variable Folders

This example illustrates how mapping output can be redirected to different folders by means of Global Resources.

Let's suppose that sometimes you need to generate the mapping output to one directory (for example, C:\Testing), while in certain cases output must be generated to another directory (for example, C:\Production). With Global Resources, this is possible by creating a folder alias with two configurations:

1. "Default" configuration - Generates output to C:\Testing
2. "Production" configuration - Generates output to C:\Production.

The steps below illustrate how to achieve this goal.

Step 1: Create the Global Resource

The folder alias can be created as follows:

1. On the Tools menu, click Global Resources. (Alternatively, click the Global Resource toolbar button.)
2. Click Add | Folder.
3. Enter a name in the Resource alias text box (in this example, "OutputDirectory" could be an appropriate name).
4. Click Browse and select the following folder: C:\Testing. (Make sure that this folder already exists on your operating system.)
5. Click Add Configuration and enter a name for the new configuration (in this example, "ProductionDirectory").
6. Click Browse and this time select the following folder: C:\Production. (Make sure that this folder already exists on your operating system.)

Step 2: Use the Global Resource in the mapping

The required Global Resource has now been created; however, the mapping is not using it yet. To change the mapping so that it uses from the previously defined folder alias (Global Resource), do the following:

1. Open the following mapping <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\Tut-ExpReport.mfd.
2. Right-click the target component on the mapping, and select Properties from the context menu.
3. Next to Output XML file, click Browse.
4. Click Switch to Global Resources, and then click Save.
5. When prompted to save the output XML file, enter output.xml (or another descriptive file name that you wish to give to the output file). The output XML file path has now become altova://folder_resource/OutputDirectory/output.xml, which indicates that the path is defined as a Global Resource.

Step 3: Run the mapping with the desired configuration

You can now easily switch to the desired mapping output folder file before running the mapping,
as follows:

- On the **Tools** menu, click **Active Configuration | Default**, and then click the **Output** tab to preview the mapping result. The mapping output (either a temporary or a permanent file, as explained below) will be generated in the **C:\Testing** directory.
- On the **Tools** menu, click **Active Configuration | ProductionDirectory**, and then click the **Output** tab. The mapping output (either a temporary or a permanent file, as explained below) will be generated in the **C:\Production** directory.

**Note:** The mapping output is written by default as a temporary file, unless you explicitly configured MapForce to write output to permanent files.

To configure MapForce to generate permanent files instead of temporary, do the following:

1. On the **Tools** menu, click **Options**.
2. In the **General** section, select the option **Write directly to final output files**.

### 12.2.8 Example: Switch Databases

When a mapping reads or writes data from a database, it is possible to switch the database connection immediately before mapping runtime (for example, from a release to a production database, and vice versa). This example illustrates how to accomplish this by means of Global Resources. Switching databases this way implies that both databases have the same structure but different data. For the purpose of this example, we will be working with the following Microsoft Access databases:

- **altova.mdb**, from the directory: `<Documents>\Altova\MapForce2019\MapForceExamples\`. This database plays the role of the default development database.
- **altova.mdb**, from the directory: `<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\`. This database plays the role of the production database.

The requirement is to easily supply to the mapping either of the two databases immediately before runtime, without editing the mapping. To achieve this requirement, we will create a database Global Resource (database alias) with two configurations:

1. **Default.** This configuration will point to the default development database.
2. **Release.** This configuration will point to the release database.

### Step 1: Create the Global Resource (database alias)

1. On the **Tools** menu, click **Global Resources**. (Alternatively, click the **Global Resource** toolbar button.)
2. Click **Add | Database**.
3. Enter a descriptive name in the **Resource alias** text box (for example, "SourceDatabase").
4. Click **Choose Database**, select **Microsoft Access (ADO)**, and browse for the development database (`<Documents>\Altova\MapForce2019\MapForceExamples\altova.mdb`).
5. Click **Add Configuration** and name it "ReleaseDatabase".
6. Click **Choose Database**, select **Microsoft Access (ADO)**, and this time browse for the
production database (<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\altova.mdb).

Step 2: Use the Global Resource in the mapping
Now that the database alias has been created, the mapping must be modified to use it.

1. Open the following mapping: <Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\PersonDB.mfd.
2. Right-click the database component, and select Properties from the context menu.
3. Click Change, and select the “SourceDatabase” alias created previously.

4. Click Connect.
5. When prompted to select the database objects, leave the default selection as is, and click OK.

Note: When a database connection is defined as a Global Resource, the settings on the Component Settings dialog are grayed out, as illustrated below. As mentioned by the text on the dialog box, the connectivity parameters can be changed from the Global Resources dialog box (which can be opened by clicking the Global Resource toolbar button). See also Databases as Global Resources.
Step 3: Run the mapping with the desired configuration

You can now easily switch to the desired database before running the mapping, as follows:

- On the **Tools** menu, click **Active Configuration | Default**, to read data from the development database.
- On the **Tools** menu, click **Active Configuration | ReleaseDatabase**, to read data from the production database.

Alternatively, select the required configuration from the **Global Resources** drop-down list.

When you switch configurations, a dialog box prompts you that the source database is about to be reloaded:
12.2.9 Example: Create an Application Workflow

This example illustrates how to create a simple workflow between Altova MapForce and Altova XMLSpy, using Global Resources. Specifically, it shows you how to trigger the execution of a MapForce mapping directly from XMLSpy, and open in XMLSpy the mapping output generated by MapForce. To make this possible, we will create a Global Resource of type "Result of MapForce Transformation", as illustrated below.

Step 1: Create the Global Resource

This step can be performed from both MapForce and XMLSpy.

1. On the Tools menu, click Global Resources. (Alternatively, click the Global Resource toolbar button.)
2. Click Add | File.
3. Enter a descriptive name in the Resource alias text box (in this example, "MappingResult").
4. Select the option Result of MapForce Transformation.
5. Click Browse and select the mapping <Documents>\Altova\MapForce2019 \MapForceExamples\Tutorial\Tut-ExpReport-multi.mfd. Be patient while the "Inputs" and "Outputs" sections on the dialog box are populated. As shown below, this mapping has one input and two outputs.

Note: Both databases used in this example contain similar data, so there are no differences in the generated output after you run the mapping.
For the scope of this example, we would like to generate each of the two output files to the folder **C:\temp**, and change the default file name. To achieve this, we will create a configuration for each desired output, as follows:

1. Under "Outputs", click **Browse** next to the first output and enter **C:\temp\file1.xml** as destination file name. This is the default configuration which will produce the first output file when triggered.
2. Click Add Configuration as a copy... and enter a name for the new configuration (in this example, “Output2”).

3. Under "Outputs", click Browse next to the second output and enter C:\temp\file2.xml as destination file name. This is the alternative configuration which produces the second output file.
Step 2: Trigger the workflow

The Global Resource created in the previous step can now be consumed from XMLSpy, as follows.

1. Run XMLSpy.
2. On the Tools menu, click Global Resources.
3. In the "Files" group, click the "MappingResult" Global Resource created previously, and then click View.
This executes the mapping, produces the default output (file1.xml) and loads it into the main pane of XMLSpy. The file is also saved as C:\temp\file1.xml.

To trigger the mapping execution with the alternative configuration, do the following:

1. On the **Tools** menu, click **Active Configuration | Output2**.

2. Click **Reload** when prompted.

As a result, the alternative output file is loaded into the main pane of XMLSpy. The file is also saved as: C:\temp\file2.xml.
12.3 Styling Mapping Output with StyleVision

In mappings where the target component is XML or XBRL, it is possible to preview and save the mapping output as HTML, RTF, PDF and Word 2007+ documents, provided that Altova StyleVision is installed on your computer. If you are using the Enterprise edition of StyleVision, then charts will also be rendered in these previews.

When a mapping supports preview in any of these formats, additional tabs become available next to the Output tab, for example:

<table>
<thead>
<tr>
<th>Mapping</th>
<th>DB Query</th>
<th>Output</th>
<th>HTML</th>
<th>RTF</th>
<th>PDF</th>
<th>Word 2007+</th>
</tr>
</thead>
</table>

*StyleVision preview tabs (MapForce Enterprise Edition)*

Note the following:

- When StyleVision Professional is installed, it is possible to preview HTML and RTF output. With StyleVision Enterprise, it is possible to preview HTML, RTF, PDF, and Word 2007+ output.
- Previewing mapping output as PDF requires Java, Acrobat Reader, and FOP (Formatting Objects Processor) version 0.93 or 1.0. FOP is installed together with StyleVision, unless you opted not to install it when installing StyleVision.
- In the 64-bit edition of MapForce, the Word 2007+ and RTF previews are opened as a non-embedded application.
- If your mapping contains components that act both as source and target (pass-through components), the StyleVision preview will only be possible for those components where the Preview button of the component has been set as active. For more information about such mappings, see Chained Mappings.

In order to preview data from a mapping in this way, the following is required:

- Altova StyleVision must be installed on your computer, either as a standalone installation, or as part of Altova MissionKit.
- The target component must have a StyleVision Power Stylesheet (SPS) file associated to it. The stylesheet file can be created or edited with StyleVision. You cannot edit or change the stylesheet in MapForce directly, but you can open it via MapForce in StyleVision. Once the stylesheet is ready, you can assign it to a target MapForce component, as shown below.

Assigning a StyleVision Power Stylesheet to a target component

1. In StyleVision, create the required stylesheet file. Make sure to use as source the same XML schema as that of the MapForce component.
2. In MapForce, right-click the target XML or XBRL component, and select Properties.
3. On the Component Settings dialog box, next to StyleVision Power Stylesheet file, browse for the stylesheet file created previously.
Note: The path to the StyleVision Power Stylesheet file can be absolute or relative, see Using Relative and Absolute Paths.

Saving the StyleVision-generated output
You can save the StyleVision-generated HTML, PDF, RTF, or Word 2007+ output to a file in a similar way as saving the result of any other mapping. Namely, after previewing the mapping, do one of the following:

- Click the Save generated output ( ) toolbar button.
- On the Output menu, click Save Output File.

Automating generation of HTML, PDF, RTF, Word 2007+ files with Altova product suite
If your mapping should generate HTML, PDF, RTF, and Word 2007+ files automatically (either on the same or on a different computer or even platform), this is possible with MapForce Server and StyleVision Server (these are separately licensed server products that extend the functionality of MapForce and StyleVision, respectively). In this scenario, each application plays the following distinct role:

- MapForce - enables you to design the mapping (.mfd file) which defines the data transformation inputs and outputs (for example, database to XML)
- MapForceServer - runs the executable mapping (.mfx file) at the command-line or from an API (either on the same or a different operating system)
- StyleVision - enables you to design the stylesheet (.sps file) required to transform mapping output to HTML, PDF, RTF, Word 2007+
- StyleVision Server - runs the .sps stylesheet which transforms the mapping output to a target desired format. This happens at the command line or from an API (either on the same or a different operating system).
- Both StyleVision Server and MapForce Server can optionally run under the management of FlowForce Server (licensed separately). In this scenario, MapForce mappings and StyleVision transformations can run as scheduled, triggered, or on-demand jobs, and thus be fully automated.

12.3.1 Examples of Mappings with StyleVision Stylesheets

Many of the mappings included in the MapForce examples folder (<Documents>\Altova \MapForce2019\MapForceExamples) have StyleVision Power Stylesheets (.sps files) assigned to their target components. When that is the case, the mapping contains the additional StyleVision preview tabs.
Styling Mapping Output with StyleVision

One such example is CompletePO.mfd available at the following path: `<Documents>\Altova\MapForce2019\MapForceExamples\CompletePO.mfd`. This mapping produces a purchase order in XML format. Right-click the target component, select Properties, and notice that it has an .sps file assigned to it.

Click the Output tab to view the output data in HTML format.
Another example is **YearlySales.mfd**. This mapping is available at the following path:

`<Documents>\Altova\MapForce2019\MapForceExamples\Tutorial\YearlySales.mfd>`
The stylesheet assigned to this mapping was designed (in StyleVision) in such a way that it is possible to control the type of the chart by changing the value of the `ChartType` element. This makes it possible to change the chart type directly from the mapping. Namely, you can change the default value of the constant to any value from 1 through 7. Notice that, if you place the mouse cursor over the `value-map` component, the possible values are displayed.

The default value of the constant is "2", which generates a 3D Pie chart in the output. To display other chart types, change this value to any other allowed value, click the **Output** tab, and observe the result.
12.4 Generating and Customizing Mapping Documentation

The **Generate Documentation** command generates detailed documentation about your mapping in HTML, MS Word, RTF or PDF. The documentation generated by this command can be freely altered and used; permission from Altova to do so is not required.

Documentation is generated for components you select in the Generate Documentation dialog box. You can either use the fixed design, or use a StyleVision Power Stylesheet (SPS) for the design. Using a StyleVision SPS enables you to customize the design of the generated documentation (see **User-Defined Design**).

**Notes**
- To use an SPS to generate mapping documentation, you must have StyleVision installed on your machine. Related elements are typically hyperlinked in the onscreen output, enabling you to navigate from component to component.
- To generate documentation in MS Word format, you must have MS Word (version 2000 or later) installed.

The screenshot below shows a portion of the **Lookup-standard.mfd** file available in the ...
\MapForceExamples folder.

Having opened a mapping file e.g. Lookup-standard.mfd:
- Select the menu option **File | Generate Documentation**. This opens the “Generate documentation” dialog box. The screenshot below shows the default dialog box settings.
Documentation Design

- Select "Use fixed design..." to use the built-in documentation template.
- Select "Use user-defined..." to use a predefined StyleVision Power Stylesheet created in StyleVision. The SPS files are available in the \Documentation\MapForce\folder.
- Click Browse to browse for a predefined SPS file.
- Click Edit to launch StyleVision and open the selected SPS in a StyleVision window.

The following predefined SPS stylesheets are available in the \Documentation\MapForce\folder:

- FunctionCallGraph.sps - shows the call graph of the main mapping and any user-defined functions.
- FunctionsUsedBy.sps - shows which functions are used directly, or indirectly, in the mapping.
- ImpactAnalysis.sps - lists every source and target node, and the route taken via various functions, to the target node.
- OverallDocumentation.sps - shows all nodes, connections, functions, and target nodes.

The output using this option outputs the maximum detail and is identical to the built-in "fixed design..." output.

Output Format

- The output format is specified here: either HTML, Microsoft Word, RTF, or PDF.
Microsoft Word documents are created with the `.doc` file extension when generated using a fixed design, and with a `.docx` file extension when generated using a StyleVision SPS. The PDF output format is only available if you use a StyleVision SPS to generate the documentation.

- Select "Split output to multiple files" if you would like separate input, output, constant components, user-defined functions from the Library component documentation. In fixed designs, links between multiple documents are created automatically.
- The "Show Result File..." option is enabled for all output options. When checked, the result files are displayed in default browser (HTML output), MS Word (MS Word output), and the default application for .rtf files (RTF output).

Path length limit

Allows you to define the maximum "path" length to be shown for items.
- E.g. `.../ShortPO/LineItems/LineItem`, which would be the maximum length for the default setting 3.

Include

- Allows you to define the specific components to appear in the documentation.

Details

Allows you to set the specific details to appear in the documentation.
- selecting "Library Names" would insert the "core" prefix for functions.
- You can document both connected, as well as unconnected nodes.

Note:

The Check/Uncheck All buttons allow you to check/uncheck all check boxes of that group.

Having used the default settings shown above, clicking OK, prompts you for the name of the output file and the location to which it should be saved. A portion of the fixed design generated documentation is shown below. Note that this shows a single output file.

This table shows the connections from the source component to the target component(s).
The sequence in which the components are documented is: Input, Output, Constant, User-defined functions, then Library functions.

E.g. **Input component** ShortPO:

- The first two items ShortPO and ShortPO/CustomerNr are not connected to any item in the target, thus the Connections column is empty.
- **ShortPO/LineItems** is directly connected to **CompletePO/LineItems** in the target.
- /Lineitems/Lineitem/ArticleNr has two connections:
  - directly to Lineitem/Article/Number in the target
  - to the User-defined function **LookupArticle**, with ArticleNr as the input parameter, and Name as the output parameter of the user-defined function.

The contents of the user-defined function are shown below.
Output component CompletePO: This table shows the connections to the target component from the source component(s).

Output **CompletePO (CompletePO.xsd)**

<table>
<thead>
<tr>
<th>Connections</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>File: CompletePO.xsd</td>
</tr>
<tr>
<td></td>
<td>Type: string</td>
</tr>
<tr>
<td></td>
<td>CompletePO</td>
</tr>
<tr>
<td></td>
<td>Type: restriction of xs: anyType [0..1]</td>
</tr>
<tr>
<td></td>
<td>CompletePO/CompletePO</td>
</tr>
<tr>
<td></td>
<td>Type: restriction of xs: anyType</td>
</tr>
<tr>
<td>ShortPO/LineItems</td>
<td>direct</td>
</tr>
<tr>
<td></td>
<td>Type: restriction of xs: anyType [1..*]</td>
</tr>
<tr>
<td>ShortPO/LineItems/LineItem</td>
<td>direct</td>
</tr>
<tr>
<td></td>
<td>CompletePO/LineItems/LineItem</td>
</tr>
<tr>
<td></td>
<td>Type: restriction of xs: anyType [1..*]</td>
</tr>
<tr>
<td>...LineItems/LineItem/Article</td>
<td>type: extension of ArticleType</td>
</tr>
<tr>
<td>...LineItems/LineItem/ArticleNr</td>
<td>type: xs:integer</td>
</tr>
<tr>
<td>...LineItems/LineItem/ArticleNr</td>
<td>user.LookupArticle == ArticleNr</td>
</tr>
</tbody>
</table>

- The first two items CompletePO and CompletePO/CompletePO are not connected to any item in the source component, thus the Connections column is empty.
- **CompletePO/LineItems** is directly connected to **ShortPO/LineItems** in the source component.
- LineItem/Article/Name is connected to the User-defined function **LookupArticle**, with LineItems/LineItem/ArticleNr as the source item.
### User-defined function defines

**user.LookupArticle**

**Input:** Articles (Articles.xsd)

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>File: Articles xml</td>
<td></td>
</tr>
<tr>
<td>Type: string</td>
<td></td>
</tr>
<tr>
<td>Articles</td>
<td></td>
</tr>
<tr>
<td>Type: restriction of xs:anyType [0..1]</td>
<td></td>
</tr>
<tr>
<td>Articles/Article</td>
<td></td>
</tr>
<tr>
<td>Type: ArticleType [1..∞]</td>
<td></td>
</tr>
<tr>
<td>Articles/ArticleNumber</td>
<td>core.equal =&gt; k</td>
</tr>
<tr>
<td>Type: xs:integer</td>
<td>core.filter =&gt; bool</td>
</tr>
<tr>
<td>Articles/ArticleName</td>
<td>core.filter =&gt; nodeseq</td>
</tr>
<tr>
<td>Type: xs:string</td>
<td>core.output =&gt; Name</td>
</tr>
<tr>
<td>Articles/Article/SinglePrice</td>
<td>core.output =&gt; Name</td>
</tr>
<tr>
<td>Type: xs:decimal</td>
<td>Type: string</td>
</tr>
</tbody>
</table>

**Input (required): core.ArticleNr**

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArticleNr</td>
<td></td>
</tr>
<tr>
<td>Type: string</td>
<td></td>
</tr>
<tr>
<td>core.equal =&gt; a</td>
<td>result =&gt;</td>
</tr>
<tr>
<td>core.filter =&gt; bool</td>
<td>on-true =&gt;</td>
</tr>
<tr>
<td>core.output =&gt; Name</td>
<td></td>
</tr>
<tr>
<td>Type: string</td>
<td></td>
</tr>
</tbody>
</table>
12.4.1 Predefined StyleVision Power Stylesheets

Function Call Graphs - PersonListByBranchOffice.mfd

This report shows call graphs of the main mapping and all user-defined functions.

Show all functions  Collapse all functions

Main mapping

|--- core.equal
|--- core.filter
|--- user.LookupPerson
 | |--- core.filter
 | |--- user.EqualAnd
 | | |--- core.equal
 | | |--- core.logical-and
 | |--- user.Person2Details
 | | |--- core.concat

user.LookupPerson

|--- core.filter
|--- user.EqualAnd
 | |--- core.equal
 | |--- core.logical-and
|--- user.Person2Details
 | |--- core.concat

user.EqualAnd

|--- core.equal
|--- core.logical-and

user.Person2Details

|--- core.concat
Functions Used By - PersonListByBranchOffice.mfd

Library core

<table>
<thead>
<tr>
<th>Function</th>
<th>Directly used by</th>
<th>Indirectly used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>core.equal</td>
<td>Main mapping</td>
<td>user.LookupPerson</td>
</tr>
<tr>
<td></td>
<td>user.EqualAnd</td>
<td></td>
</tr>
<tr>
<td>core.filter</td>
<td>Main mapping</td>
<td></td>
</tr>
<tr>
<td></td>
<td>user.LookupPerson</td>
<td></td>
</tr>
<tr>
<td>core.logical-and</td>
<td>user.EqualAnd</td>
<td>Main mapping</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.LookupPerson</td>
</tr>
<tr>
<td>core.concat</td>
<td>user.Person2Details</td>
<td>Main mapping</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.LookupPerson</td>
</tr>
</tbody>
</table>

Library user

<table>
<thead>
<tr>
<th>Function</th>
<th>Directly used by</th>
<th>Indirectly used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>user.LookupPerson</td>
<td>Main mapping</td>
<td></td>
</tr>
<tr>
<td>user.EqualAnd</td>
<td>user.LookupPerson</td>
<td>Main mapping</td>
</tr>
<tr>
<td>user.Person2Details</td>
<td>user.LookupPerson</td>
<td>Main mapping</td>
</tr>
</tbody>
</table>
Impact Analysis - PersonListByBranchOffice.mfd

This report lists every input and output node connection independently and is perfect for further impact analysis with modelling tools.

<table>
<thead>
<tr>
<th>Input Node</th>
<th>Functions</th>
<th>Output Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>OfficeName</td>
<td>core.equal, core.filter</td>
<td>PersonList</td>
</tr>
<tr>
<td>OfficeName</td>
<td>user.LookupPerson</td>
<td>PersonList/Person/Details</td>
</tr>
<tr>
<td>BranchOffices/Office</td>
<td>core.filter</td>
<td>PersonList</td>
</tr>
<tr>
<td>BranchOffices/Office/Name</td>
<td>core.equal, core.filter</td>
<td>PersonList</td>
</tr>
<tr>
<td>BranchOffices/Office/Contact</td>
<td></td>
<td>PersonList/Person</td>
</tr>
<tr>
<td>.../Office/Contact/first</td>
<td></td>
<td>PersonList/Person/First</td>
</tr>
<tr>
<td>.../Office/Contact/first</td>
<td>user.LookupPerson</td>
<td>PersonList/Person/Details</td>
</tr>
<tr>
<td>.../Office/Contact/last</td>
<td></td>
<td>PersonList/Person/Last</td>
</tr>
<tr>
<td>.../Office/Contact/last</td>
<td>user.LookupPerson</td>
<td>PersonList/Person/Details</td>
</tr>
</tbody>
</table>
12.4.2 Custom Design

Instead of the fixed design, you can create a customized design for the MapForce documentation. The customized design is created in a StyleVision SPS. Note that there are 4 predefined SPS Stylesheets supplied with MapForce, please see Documenting mapping projects.

Specifying the SPS to use for MapForce documentation

The SPS you wish to use for generating the documentation is specified in the Generate Documentation dialog (accessed via File | Generate Documentation). Select the "Use User-Defined Design..." radio button then click the dropdown arrow of the combo box and select the file you want. The default selection is the OverallDocumentation.sps entry.
These predefined SPS files are located in the ...\Documentation\MapForce folder.

Note: To use an SPS to generate documentation, you must have StyleVision installed on your machine.

Creating the SPS
A StyleVision Power Stylesheet (or SPS) is created using Altova StyleVision (https://www.altova.com/stylevision). An SPS for generating MapForce documentation must be based on the XML Schema that specifies the structure of the XML document that contains the MapForce documentation.

This schema is called MapForceDocumentation.xsd and is delivered with your MapForce installation package. It is stored in the ...\Documents\Altova\MapForce2019\Documentation folder.

When creating the SPS design in StyleVision, nodes from the MapForceDocumentation.xsd schema are placed in the design and assigned styles and properties. Note that the MapForceDocumentation.xsd includes the Documentation.xsd file located in the folder above it.

Additional components, such as links and images, can also be added to the SPS design. How to create an SPS design in StyleVision is described in detail in the StyleVision user manual.

The advantage of using an SPS for generating mapping documentation is that you have complete control over the design of the documentation. Note also that PDF output of the documentation is available only if an SPS is used; PDF output is not available if the fixed design is used.
12.5 Customizing Keyboard Shortcuts

You can define or change the keyboard shortcuts in MapForce as follows:

1. On the **Tools** menu, click **Customize**.
2. Click the **Keyboard** tab.

**To assign a new Shortcut to a command:**

1. Select the **Tools | Customize** command and click the Keyboard tab.
2. Click the **Category** combo box to select the menu name.
3. Select the **command** you want to assign a new shortcut to, in the Commands list box.
4. Click in the **Press New Shortcut Key**: text box, and press the shortcut keys that are to activate the command.

The shortcuts appear immediately in the text box. If the shortcut was assigned previously, then that function is displayed below the text box.

5. Click the **Assign** button to assign the shortcut. The shortcut now appears in the Current Keys list box.

(To **clear** the entry in the Press New Shortcut Key text box, press any of the control keys, **CTRL**, **ALT** or **SHIFT**).

**To de-assign or delete a shortcut:**

1. Click the shortcut you want to delete in the Current Keys list box.
2. Click the **Remove** button.
3. Click the **Close** button to confirm.
**Note:** The **Set accelerator for** does not currently have any function.

The currently assigned keyboard shortcuts are as follows:

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>Help Menu</td>
</tr>
<tr>
<td>F2</td>
<td>Next bookmark (in output window)</td>
</tr>
<tr>
<td>F3</td>
<td>Find Next</td>
</tr>
<tr>
<td>F10</td>
<td>Activate menu bar</td>
</tr>
<tr>
<td>Num +</td>
<td>Expand current item node</td>
</tr>
<tr>
<td>Num -</td>
<td>Collapse item node</td>
</tr>
<tr>
<td>Num *</td>
<td>Expand all from current item node</td>
</tr>
<tr>
<td>CTRL + TAB</td>
<td>Switches between open mappings</td>
</tr>
<tr>
<td>CTRL + F6</td>
<td>Cycle through open windows</td>
</tr>
<tr>
<td>CTRL + F4</td>
<td>Closes the active mapping document</td>
</tr>
<tr>
<td>Alt + F4</td>
<td>Closes MapForce</td>
</tr>
<tr>
<td>Alt + F, F, 1</td>
<td>Opens the last file</td>
</tr>
<tr>
<td>Alt + F, T, 1</td>
<td>Opens the last project</td>
</tr>
<tr>
<td>CTRL + N</td>
<td>File New</td>
</tr>
<tr>
<td>CTRL + O</td>
<td>File Open</td>
</tr>
<tr>
<td>CTRL + S</td>
<td>File Save</td>
</tr>
<tr>
<td>CTRL + P</td>
<td>File Print</td>
</tr>
<tr>
<td>CTRL + A</td>
<td>Select All</td>
</tr>
<tr>
<td>CTRL + X</td>
<td>Cut</td>
</tr>
<tr>
<td>CTRL + C</td>
<td>Copy</td>
</tr>
<tr>
<td>CTRL + V</td>
<td>Paste</td>
</tr>
<tr>
<td>CTRL + Z</td>
<td>Undo</td>
</tr>
<tr>
<td>CTRL + Y</td>
<td>Redo</td>
</tr>
<tr>
<td>Del</td>
<td>Delete component (with prompt)</td>
</tr>
<tr>
<td>Shift + Del</td>
<td>Delete component (no prompt)</td>
</tr>
<tr>
<td>CTRL + F</td>
<td>Find</td>
</tr>
<tr>
<td>F3</td>
<td>Find Next</td>
</tr>
<tr>
<td>Shift + F3</td>
<td>Find Previous</td>
</tr>
</tbody>
</table>

**Arrow keys**

(up / down) Select next item of component

Esc Abandon edits/close dialog box

Return Confirms a selection

**Output window hotkeys**

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL + F2</td>
<td>Insert Remove/Bookmark</td>
</tr>
<tr>
<td>F2</td>
<td>Next Bookmark</td>
</tr>
<tr>
<td>SHIFT + F2</td>
<td>Previous Bookmark</td>
</tr>
<tr>
<td>CTRL + SHIFT + F2</td>
<td>Remove All Bookmarks</td>
</tr>
</tbody>
</table>

**Zooming hotkeys**

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL + mouse wheel forward</td>
<td>Zoom In</td>
</tr>
<tr>
<td>CTRL + mouse wheel back</td>
<td>Zoom Out</td>
</tr>
</tbody>
</table>
CTRL + 0 (Zero)  Reset Zoom
12.6 Catalog Files

MapForce supports a subset of the OASIS XML catalogs mechanism. The catalog mechanism enables MapForce to retrieve commonly used schemas (as well as stylesheets and other files) from local user folders. This increases the overall processing speed, enables users to work offline (that is, not connected to a network), and improves the portability of documents (because URIs would then need to be changed only in the catalog files.)

The catalog mechanism in MapForce works as outlined below.

RootCatalog.xml

When MapForce starts, it loads a file called RootCatalog.xml (structure shown in listing below), which contains a list of catalog files that will be looked up. You can modify this file and enter as many catalog files to look up as you like, each in a nextCatalog element. Each of these catalog files is looked up and the URIs in them are resolved according to the mappings specified in them.

<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
    xmlns:spy="http://www.altova.com/catalog_ext"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:oasis:names:tc:entity:xmlns:xml:catalog Catalog.xsd">
    <nextCatalog catalog="%PersonalFolder%/Altova/%AppAndVersionName%/CustomCatalog.xml"/>
    <nextCatalog catalog="CoreCatalog.xml"/>
    <!-- Include all catalogs under common schemas folder on the first directory level -->
    <nextCatalog spy:recurseFrom="%AltovaCommonFolder%/Schemas" catalog="catalog.xml" spy:depth="1"/>
    <!-- Include all catalogs under common XBRL folder on the first directory level -->
    <nextCatalog spy:recurseFrom="%AltovaCommonFolder%/XBRL" catalog="catalog.xml" spy:depth="1"/>
</catalog>

In the listing above, notice that in the Schemas and XBRL folders of the folder identified by the variable %AltovaCommonFolder% are catalog files named catalog.xml. (The value of the %AltovaCommonFolder% variable is given in the table below.)

The catalog files in the Altova Common Folder map the pre-defined public and system identifiers of commonly used schemas (such as SVG and WSDL) and XBRL taxonomies to URIs that point to locally saved copies of the respective schemas. These schemas are installed in the Altova Common Folder when MapForce is installed. You should take care not to duplicate mappings in these files, as this could lead to errors.

CoreCatalog.xml, CustomCatalog.xml, and Catalog.xml

In the RootCatalog.xml listing above, notice that CoreCatalog.xml and CustomCatalog.xml are listed for lookup:
• **CoreCatalog.xml** contains certain Altova-specific mappings for locating schemas in the Altova Common Folder.

• **CustomCatalog.xml** is a skeleton file in which you can create your own mappings. You can add mappings to CustomCatalog.xml for any schema you require but that is not addressed by the catalog files in the Altova Common Folder. Do this using the supported elements of the OASIS catalog mechanism (see below).

• There are a number of **Catalog.xml** files in the Altova Common Folder. Each is inside the folder of a specific schema or XBRL taxonomy in the Altova Common Folder, and each maps public and/or system identifiers to URIs that point to locally saved copies of the respective schemas.

**Location of catalog files and schemas**

The files **RootCatalog.xml** and **CoreCatalog.xml** are installed in the MapForce application folder. The file **CustomCatalog.xml** is located in your MyDocuments/Altova/MapForce folder. The catalog.xml files are each in a specific schema folder, these schema folders being inside the folders: %AltovaCommonFolder%\Schemas and %AltovaCommonFolder%\XBRL.

**Shell environment variables and Altova variables**

Shell environment variables can be used in the nextCatalog element to specify the path to various system locations (see RootCatalog.xml listing above). The following shell environment variables are supported:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%AltovaCommonFolder%</td>
<td>C:\Program Files\Altova\Common2019</td>
</tr>
<tr>
<td>%DesktopFolder%</td>
<td>Full path to the Desktop folder for the current user.</td>
</tr>
<tr>
<td>%ProgramMenuFolder%</td>
<td>Full path to the Program Menu folder for the current user.</td>
</tr>
<tr>
<td>%StartMenuFolder%</td>
<td>Full path to Start Menu folder for the current user.</td>
</tr>
<tr>
<td>%StartUpFolder%</td>
<td>Full path to Start Up folder for the current user.</td>
</tr>
<tr>
<td>%TemplateFolder%</td>
<td>Full path to the Template folder for the current user.</td>
</tr>
<tr>
<td>%AdminToolsFolder%</td>
<td>Full path to the file system directory that stores administrative tools for the current user.</td>
</tr>
<tr>
<td>%AppDataFolder%</td>
<td>Full path to the Application Data folder for the current user.</td>
</tr>
<tr>
<td>%CommonAppDataFolder%</td>
<td>Full path to the file directory containing application data for all users.</td>
</tr>
<tr>
<td>%FavoritesFolder%</td>
<td>Full path of the Favorites folder for the current user.</td>
</tr>
<tr>
<td>%PersonalFolder%</td>
<td>Full path to the Personal folder for the current user.</td>
</tr>
<tr>
<td>%SendToFolder%</td>
<td>Full path to the SendTo folder for the current user.</td>
</tr>
</tbody>
</table>
### How catalogs work

Catalogs are commonly used to redirect a call to a DTD to a local URI. This is achieved by mapping, in the catalog file, public or system identifiers to the required local URI. So when the DOCTYPE declaration in an XML file is read, the public or system identifier locates the required local resource via the catalog file mapping.

For popular schemas, the **PUBLIC** identifier is usually pre-defined, thus requiring only that the URI in the catalog file point to the correct local copy. When the XML document is parsed, the **PUBLIC** identifier in it is read. If this identifier is found in a catalog file, the corresponding URL in the catalog file will be looked up and the schema will be read from this location. So, for example, if the following SVG file is opened in MapForce:

```xml
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="20" height="20" xml:space="preserve">
  <g style="fill:red; stroke:#000000">
    <rect x="0" y="0" width="15" height="15"/>
    <rect x="5" y="5" width="15" height="15"/>
  </g>
</svg>
```

This document is read and the catalog is searched for the **PUBLIC** identifier. Let’s say the catalog file contains the following entry:

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%FontsFolder%</td>
<td>Full path to the System Fonts folder.</td>
</tr>
<tr>
<td>%ProgramFilesFolder%</td>
<td>Full path to the Program Files folder for the current user.</td>
</tr>
<tr>
<td>%CommonFilesFolder%</td>
<td>Full path to the Common Files folder for the current user.</td>
</tr>
<tr>
<td>%WindowsFolder%</td>
<td>Full path to the Windows folder for the current user.</td>
</tr>
<tr>
<td>%SystemFolder%</td>
<td>Full path to the System folder for the current user.</td>
</tr>
<tr>
<td>%CommonAppDataFolder%</td>
<td>Full path to the file directory containing application data for all users.</td>
</tr>
<tr>
<td>%LocalAppDataFolder%</td>
<td>Full path to the file system directory that serves as the data repository for local (nonroaming) applications.</td>
</tr>
<tr>
<td>%MyPicturesFolder%</td>
<td>Full path to the MyPictures folder.</td>
</tr>
</tbody>
</table>
In this case, there is a match for the PUBLIC identifier, so the lookup for the SVG DTD is redirected to the URI schemas/svg/svg11.dtd (this path is relative to the catalog file), and this local file will be used as the DTD. If there is no mapping for the Public ID in the catalog, then the URL in the XML document will be used (in the example above: http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd).

The catalog subset supported by MapForce

When creating entries in CustomCatalog.xml (or any other catalog file that is to be read by MapForce), use only the following elements of the OASIS catalog specification. Each of the elements below is listed with an explanation of their attribute values. For a more detailed explanation, see the XML Catalogs specification. Note that each element can take the xml:base attribute, which is used to specify the base URI of that element.

- `<public publicId="PublicID of Resource" uri="URL of local file"/>
- `<system systemId="SystemID of Resource" uri="URL of local file"/>
- `<uri name="filename" uri="URL of file identified by filename"/>
- `<rewriteURI uriStartString="StartString of URI to rewrite" rewritePrefix="String to replace StartString"/>
- `<rewriteSystem systemIdStartString="StartString of SystemID" rewritePrefix="Replacement string to locate resource locally"/>

In cases where there is no public identifier, as with most stylesheets, the system identifier can be directly mapped to a URL via the system element. Also, a URI can be mapped to another URI using the uri element. The rewriteURI and rewriteSystem elements enable the rewriting of the starting part of a URI or system identifier, respectively. This allows the start of a filepath to be replaced and consequently enables the targeting of another directory. For more information on these elements, see the XML Catalogs specification.

File extensions and intelligent editing according to a schema

Via catalog files you can also specify that documents with a particular file extension should have MapForce's intelligent editing features applied in conformance with the rules in a schema you specify. For example, if you create a custom file extension .myhtml for (HTML) files that are to be valid according to the HTML DTD, then you can enable intelligent editing for files with these extensions by adding the following element of text to CustomCatalog.xml as a child of the <catalog> element.

```xml
<spy:fileExtHelper ext="myhtml" uri="schemas/xhtml/xhtml1-transitional.dtd"/>
```

This would enable intelligent editing (auto-completion, entry helpers, etc) of .myhtml files in MapForce according to the XHTML 1.0 Transitional DTD. Refer to the catalog.xml file in the %AltovaCommonFolder%\Schemas\xhtml folder, which contains similar entries.

XML Schema and catalogs

XML Schema information is built into MapForce and the validity of XML Schema documents is...
checked against this internal information. In an XML Schema document, therefore, no references should be made to any schema for XML Schema.

The catalog.xml file in the %AltovaCommonFolder%\Schemas\schema folder contains references to DTDs that implement older XML Schema specifications. You should not validate your XML Schema documents against either of these schemas. The referenced files are included solely to provide MapForce with entry helper info for editing purposes should you wish to create documents according to these older recommendations.

More information
For more information on catalogs, see the XML Catalogs specification.
Chapter 13

MapForce Plug-in for Visual Studio
13 MapForce Plug-in for Visual Studio

You can integrate MapForce 2019 into the Microsoft Visual Studio versions 2010/2012/2013/2015/2017. This unifies the best of both worlds, combining the mapping capabilities of MapForce with the development environment of Visual Studio. When the MapForce plug-in is enabled, you can create mapping projects and files directly from Visual Studio. You can also customize the MapForce options, including menus and toolbars, as you would do in the standalone version of MapForce.

13.1 Enabling the Plug-in

Prerequisites:

- Microsoft Visual Studio 2010/2012/2013/2015/2017
- MapForce (Enterprise or Professional Edition)

**Note:** To use MapForce as a Visual Studio plug-in, install the 32-bit version of both MapForce and MapForce integration package, since there is currently no support for 64-bit plug-ins in Visual Studio.

To enable the MapForce plug-in for Visual Studio, download and run the MapForce Integration Package and follow the on-screen installation instructions.

During installation, ensure that the **Install the Microsoft Visual studio plug-in** option is selected:

![Altova MapForce Integration Package - Setup](image)

When prompted, select the Visual Studio version(s) where the plug-in should be enabled, for example:
Enabling the Plug-in

Note: Only the Visual Studio versions installed on your operating system are available for selection.

Once the integration package has been installed, MapForce functionality becomes available in the Visual Studio environment.

Enabling the MapForce plug-in manually
It is possible that the plug-in was not automatically enabled during the installation process. To enable it, do the following:

1. Navigate to the directory where the Visual Studio IDE executable is installed (for example, c:\Program Files\Microsoft Visual Studio 8\Common7\IDE).
2. Run the command prompt as administrator and enter `devenv.exe /setup`.
3. Wait for the process to terminate normally before starting to use the application within Visual Studio.
13.2 Working with Mappings and Projects

When MapForce plug-in for Visual Studio is enabled, you create and open mappings and mapping projects in a way that is applicable to the Visual Studio environment, as opposed to the standalone MapForce graphical user interface. For example, when you create a new file in Visual Studio (using the File | New menu command), or when you add a new item to a project (using the Project | Add New Item menu command), you can select MapForce Files as file type.

In a similar way, when you create a new Visual Studio project, you can select MapForce Projects as project template. The following screen shot illustrates a sample New Project dialog box in Visual Studio 2017 with the MapForce Enterprise Edition plug-in enabled.
New Project dialog box (Visual Studio 2017 with MapForce Enterprise edition plug-in)

Opening existing MapForce files and projects is also done through the Visual Studio native functionality. When you need to open existing mapping files or projects, use the applicable Visual Studio menus (for example, **File | Open | Files**, or **File | Open | Project/Solutions**), and look for the MapForce-related file types.
13.3 Accessing Common Menus and Functions

When MapForce plug-in for Visual Studio is enabled, you can access common menus and functions as shown below. This is the default setup; however, you can change, if desired, the location of menus and toolbars from the Tools | Options menu of Visual Studio.

Global Resources
MapForce Global Resources are available in the MapForce | Manage Global Resources menu of Visual Studio.

Mapping debugging
The mapping debugging commands are available in the MapForce | Debug menu and in the Debug toolbar. You can also initiate debugging from the keyboard shortcuts (see Debugger Commands).

MapForce options
MapForce options are available in the Tools | MapForce Options menu of Visual Studio.

Mapping pane customization
When there is a MapForce mapping opened in the main pane of Visual Studio, the View | MapForce menu becomes available. It includes the same options as the standalone version of MapForce.

Libraries window
The MapForce Libraries window is not enabled by default in Visual Studio after you install the plug-in. If you work frequently with this window, you can enable it from the View | MapForce | Library Window menu (this menu becomes available in Visual Studio when there is a mapping file opened in the main window). Once the Libraries window is enabled, you can doc it to a particular position in the interface, like any other dockable component of Visual Studio.
The Libraries Window (Visual Studio 2017 with MapForce Enterprise edition plug-in)

**Toolbar and commands customization**

You can customize MapForce menus and toolbars from the **Tools | Customize** menu of Visual Studio.
Help and Support

MapForce Help, Support Center, Check for Updates and About menus are available in the Help | MapForce Help menu of Visual Studio.
Chapter 14

MapForce Plug-in for Eclipse
14 MapForce Plug-in for Eclipse

Eclipse is an open source framework that integrates different types of applications delivered in form of plug-ins. You can integrate MapForce Enterprise and Professional Edition into Eclipse versions 4.7 / 4.8 / 4.9 / 4.10 and access MapForce functionality directly from Eclipse.

MapForce Enterprise Edition plug-in for Eclipse

The following topics provide help on installing and using the MapForce plug-in for Eclipse.

- Installing the MapForce Plug-in for Eclipse
- The MapForce Perspective
- Accessing Common Menus and Functions
- Working with Mapping and Projects
- Extending MapForce Plug-in for Eclipse
14.1 Installing the MapForce Plug-in for Eclipse

Prerequisites:

- Java Runtime Environment (JRE) 6.0 or later (see http://www.oracle.com/technetwork/java/javase/downloads/index.html). Install a 32-bit or 64-bit JRE to match your version of MapForce (32-bit or 64-bit).
- Eclipse Platform 4.7 / 4.8 / 4.9 / 4.10 (see http://www.eclipse.org). Install a 32-bit or 64-bit Eclipse to match your version of MapForce (32-bit or 64-bit).
- MapForce Enterprise or Professional Edition.

Installing the MapForce Plug-in for Eclipse

2. Ensure that Eclipse is not running, and run the downloaded package.

   Eclipse must be closed while you install or uninstall the MapForce Integration Package.

3. When prompted, select the Install the Eclipse plug-in option, and then click Next.

4. When prompted to choose how the MapForce plug-in should be integrated into Eclipse, do one of the following:
   a. To complete the plug-in installation automatically (this is the recommended option),
select Let this wizard integrate Altova MapForce plug-in into Eclipse, and browse for the directory where the Eclipse executable (eclipse.exe) is located.

b. To complete the plug-in installation separately in Eclipse, click to clear the Let this wizard... check box (see instructions below).

2. Click Next, and complete the installation. If you chose the automatic integration, the MapForce perspective and menus become available in Eclipse next time when you start Eclipse.

Integrating the MapForce plug-in for Eclipse manually

1. In Eclipse, click the menu command Help | Install New Software.
2. In the Install dialog that pops up (screen shot below), click the Add button.
3. In the Add Repository dialog that pops up (screen shot below), click the Local button.
4. Browse for the folder `c:\Program Files\Altova\Common2019\eclipse\UpdateSite`, and select it. Provide a name for the site (such as 'Altova'), and click OK.

5. Repeat Steps 2 to 4, this time selecting the folder `c:\Program Files\Altova\MapForce2019\eclipse\UpdateSite`, and providing a name such as 'Altova MapForce'.
6. In the Work With combo box of the Install dialog, select the option -- Only Local Sites -- (see screen shot below). This causes all available plug-ins to be displayed in the pane.
below. Check the top-level check box of the Altova category folder (see screen shot below). Then click the Next button.

7. An Install Details screen allows you to review the items to be installed. Click Next to proceed.

8. In the Review Licenses screen that appears, select I accept the terms of the license agreement. (No license agreement additional to your MapForce Enterprise or Professional Edition license is required for the MapForce plug-in.) Then click Finish to complete the installation.

Note: If there are problems with the plug-in (missing icons, for example), start Eclipse from the command line with the -clean flag.
14.2 The MapForce Perspective

After you install the MapForce plug-in for Eclipse, a new perspective ("MapForce") becomes available in Eclipse. The layout of this perspective closely resembles the interface of the standalone edition of MapForce. To switch to the MapForce perspective, click **Window | Open Perspective | Other**, and choose MapForce from the list.

The MapForce perspective is just like any other Eclipse perspective—you can switch to it whenever required in Eclipse (**Window | Navigation | Next Perspective**). You can also customize the views it contains, and various other options, from Eclipse preferences. (To customize the MapForce perspective in Eclipse 4.4, switch to the MapForce perspective, and then select the menu command **Window | Customize Perspective**.) For more information about Eclipse perspectives, refer to Eclipse documentation. The following screen shot illustrates the Eclipse environment with the MapForce perspective switched on.
MapForce perspective (MapForce Enterprise Edition plug-in for Eclipse)

By default, the MapForce perspective in Eclipse is organized as follows:

- The mapping design window is available as an Eclipse editor. It has the same tabs and functionality as in the standalone edition of MapForce.
- The Libraries window is available as an Eclipse view, to the left of the main mapping editor. If this view is not visible, switch to the MapForce perspective, and then select the menu command Window | Show View | Libraries. The Libraries view enables you to work with predefined or custom-defined functions and function libraries.
- The Messages pane is available as an Eclipse view, under the main mapping editor. If the Message view is not visible, switch to the MapForce perspective, and then select the menu command Window | Show View | Messages. The messages view displays validation messages, errors, and warnings.
- The Overview pane is available as an Eclipse view. If the Overview view is not visible, switch to the MapForce perspective, and then select the menu command Window | Show View | Overview. This view enables you to quickly navigate to a particular region on the mapping design area when it is very big.

You can also configure Eclipse to switch to the MapForce perspective automatically when you open a MapForce mapping. To do this, select the menu command Window | Preferences. Select MapForce, and then select the **Automatically switch to MapForce perspective at file open** check box.
Preferences dialog box
14.3 Accessing Common Menus and Functions

In Eclipse, you can access most MapForce functionality from the same menus as in the standalone version, except for some Eclipse-specific variations which are listed below. This is the default setup; however, you can further customize the interface preferences from Eclipse, if desired (see The MapForce Perspective).

**Note:** In Eclipse, some MapForce menu groups or commands are disabled (or not available) if the context is not relevant. For example, the Insert menu becomes available only when a mapping design file (.mfd) is active in Eclipse.

For information about the MapForce standard menus, see Menu Reference.

**General MapForce commands**

In the standalone edition of MapForce, the commands applicable to mapping design files (such as Validate, Deploy to FlowForce Server, Generate Code, and others) are available in the File menu. In Eclipse, these commands are available in the MapForce menu, or in the MapForce toolbar. Note that the commands for opening or saving files (including MapForce project files) are available in the File menu of Eclipse.

**The MapForce toolbar in Eclipse**

The toolbar button opens the MapForce help file.

The toolbar button displays commands specific to MapForce files. When you expand this button, the available commands depend on the kind of file currently active in the Eclipse editor. For example, the commands specific to mapping design (.mfd) files are available when such a file is active (in focus) in the Eclipse editor.

**Mapping debugging**

The mapping debugging commands are available in the MapForce | Debug menu and in the Debug toolbar. Note that the mapping debugging shortcut keys (such as F5, F10, etc) are already reserved by Eclipse and are not available.

**Global Resources**

To access or manage Global Resources, do one of the following:

- Click to expand the MapForce toolbar button, and then click Global Resources.
- On the MapForce menu, click Global Resources.

**MapForce Projects**

In the standard edition of MapForce, the Project menu contains various commands applicable to mapping project (.mfp) files. In Eclipse, these commands exist as follows:
The commands to open or save a project are available from the Eclipse **File** menu.

Other project-specific commands are available as context commands. To display the context commands, create or open a MapForce project (.mfp) file in Eclipse, and then right-click the project.

Note that, in addition to standard MapForce projects (.mfp), in Eclipse you can also create projects of type "MapForce/Eclipse". Such projects have a dual nature, and can be configured for automatic build and generation of MapForce code. See Working with Mappings and Projects.

**MapForce Options**

MapForce options are available from the **Window | Preferences** menu. On the Preferences dialog box, select **MapForce**, and then click **Open MapForce Options Dialog**.
Libraries window
In Eclipse, the MapForce Libraries window is available as a view. This view is by default located to the left of the main editor window. (All MapForce-related views become visible in Eclipse interface when the MapForce perspective is switched on, see also The MapForce Perspective).

MapForce plug-in version
To see the currently installed version of the MapForce Plug-in for Eclipse, select the Eclipse menu option Help | About Eclipse. Then select the MapForce icon.

Help and Support
MapForce Help, Support Center, Check for Updates and About menus are available in the Help | MapForce Help menu of Eclipse.
14.4 Working with Mappings and Projects

When MapForce plug-in for Eclipse is installed, you can create from Eclipse the same mappings and mapping project types as in the standalone edition of MapForce, from within an Eclipse project. To design, test, compile, and deploy mappings, and to generate mapping code, you can either create a new Eclipse project or use an existing Eclipse project (for example, a Java project to which you want to add MapForce mappings).

In addition to this, you can work with all your mappings within a special project type that becomes available in Eclipse after you install the MapForce plug-in—the MapForce/Eclipse Project. Unless you choose to customize it, a MapForce/Eclipse project is by default assigned both a Java Builder and a MapForce Code Generation builder. Additionally, it has two Eclipse natures: MapForce nature and the JDT (Java Development tools) nature. As a result, a MapForce/Eclipse project behaves as follows when you save or change any of its resources (such as a mapping design file):

- If the **Project > Build automatically** menu option is enabled, the mapping code is generated automatically. When one or more MapForce project files exist in the MapForce/Eclipse project, the code generation language and output target folders are determined by the settings in each project file. Otherwise, Eclipse prompts you to choose a location.
- Any errors and output messages are shown in the Messages and Problems views.

This section contains the following topics:

- Creating a MapForce/Eclipse Project
- Creating New Mappings
- Importing Existing Mappings into an Eclipse Project
- Configuring Automatic Build and Generation of MapForce Code

14.4.1 Creating a MapForce/Eclipse Project

To create a MapForce/Eclipse project:

1. On the **File** menu, click **New | Other**.
2. Select the **MapForce/Eclipse Project** category.
3. Click **Next**.
4. Enter a project name and choose a location where to save the project. Leave the add MapForce builder to project and use JDT builder options as is.
5. Click Finish.

14.4.2 Creating New Mappings

You can create the following MapForce file types within an Eclipse project:

- MapForce mappings
- MapForce project files
- MapForce Web Service projects (available in MapForce Enterprise Edition)

To create any of these file types within an Eclipse project:

1. Create a new Eclipse project or open an existing one.
2. On the File menu, click New, and then click Other.
3. Select the required file type from the wizard dialog box, and then click **Next**.
14.4.3 Importing Existing Mappings into an Eclipse Project

To import MapForce mappings and their dependent files into an existing Eclipse project:

1. Open the project into which you want to import the files.
2. On the File menu, click Import.

4. Select a parent folder in your existing project, and then click Finish.
3. Select **File System**, and then click **Next**.
4. Next to **From directory**, browse for the location of the files you want to import, and then select the required files.

5. Next to **Into folder**, click **Browse**, and select the project into which you are adding the files (in this example, `MapForceEclipseProject1`).
6. Click **OK**, and then click **Finish**.

### 14.4.4 Configuring Automatic Build and Generation of MapForce Code

Automatic MapForce code building and generation is enabled by default in any MapForce/Eclipse project (see [Creating a MapForce/Eclipse Project](#)). If you want to enable automatic build and generation of MapForce code in an existing project which is not of type MapForce/Eclipse, you can do this by manually adding to it the **MapForce Code Generation** builder and the **MapForce** nature.

**To add the MapForce Code Generation builder to a project:**

- Add to the Eclipse `.project` file the lines highlighted below:

```xml
<buildSpec>
  <buildCommand>
    <name>org.eclipse.jdt.core.javabuilder</name>
  </buildCommand>
</buildSpec>
```
To add the MapForce nature to a project:

- Add to the Eclipse `.project` file the lines highlighted below:

```xml
<natures>
  <nature>org.eclipse.jdt.core.javanature</nature>
  <nature>com.altova.mapforceeclipseplugin.MapForceNature</nature>
</natures>
```

Tip: You can quickly open the `.project` file from the Navigator view of Eclipse (To enable this view, select the menu command Window | Show View | Navigator).

To switch automatic MapForce code generation on/off:

- On the Project menu, click Build automatically.

To disable the MapForce Code Generation builder:

1. On the Project menu, click Properties.
2. Click Builders.
3. Click to clear the **MapForce Code Generation** check box.
14.5 Extending MapForce Plug-in for Eclipse

The MapForce plug-in for Eclipse provides an Eclipse extension point with the ID `com.altova.mapforceeclipseplugin.MapForceAPI`. You can use this extension point to adapt, or extend the functionality of the MapForce plug-in. The extension point gives you access to the COM-Interface of the MapForce control and the MapForce API.

The MapForce Eclipse installation package contains a simple example of a plug-in that uses this extension point. It checks for any file open events of any new MapForce mappings, and sets the zoom level of the mapping view to 70%.

The JavaDoc documentation of the extension point is available in the MapForce plug-in installation directory (typically, `C:\Program Files\Altova\MapForce2019\eclipse\docs`).

Before you install and run the sample MapForce plug-in, ensure that the following prerequisites are met:

- You are using 32-bit Java, 32-bit Eclipse, 32-bit MapForce and 32-bit MapForce Integration Package.
- The JDT (Java Development Tools) plug-in is installed.
- The Eclipse PDE (plug-in development environment) is installed.

To import the sample MapForce plug-in project into your workspace:

1. Start Eclipse.
2. On the File menu, click Import.
3. Select General | Existing projects into Workspace, and click Next.
4. Click the **Browse…** button next to the “Select root directory” field and choose the sample project directory e.g. `C:\Program Files\Altova\MapForce2019\eclipse\workspace\MapForceExtension`. 
5. Select the **Copy projects into workspace** option, and then click **Finish**. A new project named "MapForceExtension" has been created in your workspace.

To run the sample extension plug-in:

1. Switch to the Java perspective.
2. In the **Run** menu, click **Run Configurations**.
3. Right click **Eclipse Application** and select **New**. (If you cannot see "Eclipse application" in the list, the Eclipse Plug-In Development Tools are not installed in your Eclipse
environment. To install Eclipse Plug-in Development Tools, click **Install New Software** in the **Help** menu and install "Eclipse Plugin Development Tools" from "The Eclipse Project Updates" download site.)

4. Enter a name for your new configuration (in this example, **SampleMapForcePlugin**), and then click **Apply**.

5. Check that the **MapForceClient** workspace plug-in is selected in the 'Plug-ins' tab.
6. Click **Run**. A new Eclipse Workbench opens.
7. Open any MapForce mapping in the new Workbench. It will now open with a zoom level of 70%.
Chapter 15

Menu Reference
15 Menu Reference

This reference section contains a description of the MapForce menu commands.
15.1 File

New
Creates a new mapping document, or mapping project (.mfp).

Open
Opens previously saved mapping design (.mfd), or mapping project (.mfp) files. Note that it is not possible to open mapping files which contain features not available in your MapForce edition.

Save
Saves the currently active mapping using the currently active file name.

Save As
Saves the currently active mapping with a different name, or allows you to supply a new name if this is the first time you save it.

Save All
Saves all currently open mapping files.

Reload
Reloads the currently active mapping file. You are asked if you want to lose your last changes.

Close
Closes the currently active mapping file. You are asked if you want to save the file before it closes.

Close All
Closes all currently open mapping files. You are asked if you want to save any of the unsaved mapping files.

Print
Opens the Print dialog box, from where you can print out your mapping as hard copy.
Print dialog box

**Use current** retains the currently defined zoom factor of the mapping. **Use optimal** scales the mapping to fit the page size. You can also specify the zoom factor numerically. Component scrollbars are not printed. You can also specify if you want to allow the graphics to be split over several pages or not.

**Print Preview**
Opens the same Print dialog box with the same settings as described above.

**Print Setup**
Opens the Print Setup dialog box in which you can define the printer you want to use and the paper settings.

**Validate Mapping**
Validates that all mappings (connectors) are valid and displays any warnings or errors (see [Validating mappings](#)).

**Mapping settings**
Opens the Mapping Settings dialog box where you can define the document-specific settings (see [Changing the mapping settings](#)).

**Generate code in selected language**
Generates code in the currently selected language of your mapping. The currently selected language is visible as a highlighted programming language icon in the toolbar: XSLT, XSLT 2, XQuery, Java, C#, or C++.

**Generate code in XSLT (XSLT2)**
This command generates the XSLT file(s) needed for the transformation from the source file(s).
Selecting this option opens the Browse for Folder dialog box where you select the location of the XSLT file. The name of the generated XSLT file(s) is defined in the Mapping Settings dialog box (see Changing the mapping settings).

**Generate code in | XQuery**
This command generates the XQuery file(s) needed for the transformation from the source file(s). Selecting this option opens the Browse for Folder dialog box where you select the location of the XQuery file. The name of the generated XQuery file(s) is defined in the Mapping Settings dialog box (see Changing the mapping settings).

**Generate code in | Java | C# | C++**
These commands generate source code for a complete application program needed for the transformation from the source file(s). Selecting this option opens the Browse for Folder dialog box, where you select the location of the generated files. The names of the generated application files (as well as the project files: *.csproj C# project file, *.sln solution file, *.vcproj visual C++ project file) are defined in the Mapping Settings dialog box (see Changing the mapping settings).

The file name created by the executed code appears in the **Output XML File** box of the Component settings dialog box if the target is an XML/Schema document.

**Compile to MapForce Server Execution File**
Generates a file that can be executed by MapForce Server to run the mapping transformation (see Compiling a MapForce mapping).

**Deploy to FlowForce Server**
Deploys the currently active mapping to the FlowForce Server (see Deploying a MapForce mapping).

**Generate documentation**
Generates documentation of your mapping projects in great detail in various output formats (see Generating and Customizing Mapping Documentation).

**Recent files**
Displays a list of the most recently opened files.

**Exit**
Exits the application. You are asked if you want to save any unsaved files.
15.2 Edit

Most of the commands in this menu become active when you view the result of a mapping in the Output tab, or preview XSLT code in the XSLT tab.

**Undo**
MapForce has an unlimited number of "Undo" steps that you can use to retrace you mapping steps.

**Redo**
The redo command allows you to redo previously undone commands. You can step backward and forward through the undo history using both these commands.

**Find**
Allows you to search for specific text in either the XSLT, XSLT2, XQuery or Output tab.

**Find Next** F3
Searches for the next occurrence of the same search string.

**Find Previous** Shift F3
Searches for the previous occurrence of the same search string.

**Cut/Copy/Paste/Delete**
The standard windows Edit commands, allow you to cut, copy etc., any components or functions visible in the mapping window.

**Select all**
Selects all components in the Mapping tab, or the text/code in the XSLT, XSLT2, XQuery or Output tab.
15.3 Insert

XML Schema / File
Adds to the mapping an XML schema or instance file. If you select an XML file which references a
schema, no additional information is required for the mapping. If you select an XML file without a
schema reference, you are prompted to generate a matching XML schema automatically (see
Generating an XML Schema ). If you select an XML schema file, you are prompted to include
optionally an XML instance file which supplies the data for preview.

Database
Adds to the mapping a database as source or target component (see Databases and MapForce ).

EDI
Adds to the mapping an EDI document which can be used as source or target component (see EDI ).

Text file
Adds to the mapping a flat file document, such as CSV or a fixed-length text file. Both types of file
can be used as source and target components. Additionally, if you want to process text files with
a structure other than CSV or fixed-length, you can use FlexText (see MapForce FlexText ).

Web Service Function
Adds to the mapping a call to a Web service (see Calling Web services ).

Excel 2007+ File
Adds to the mapping a Microsoft Excel 2007+ (.xlsx) file (see Microsoft OOXML Excel 2007+ ). If
you don't have Excel 2007 or later, you can still map to or from Excel 2007+ files. In this case,
you cannot preview the result in the Output tab, but you can still save it, by clicking Save Output
File on the Output menu.

XBRL Document
Adds to the mapping an XBRL instance or taxonomy document (see Adding XBRL Files as
Mapping Components ).

JSON Schema/File
Adds to the mapping a JSON schema or file (see Adding JSON Files as Mapping Components ).
Insert Input

When the mapping window displays a mapping, this command adds an input component to the mapping (see Supplying Parameters to the Mapping). When the mapping window displays a user-defined function, this command adds an input component to the user-defined function (see Parameters in User-Defined Functions).

Insert Output

When the mapping window displays a mapping, this command adds an output component to the mapping (see Returning String Values from a Mapping). When the mapping window displays a user-defined function, this command adds an output component to the user-defined function (see Parameters in User-Defined Functions).

Constant

Inserts a constant which supplies fixed data to an input connector. The data is entered into a dialog box when creating the component. You can select the following types of data: String, Number and All other.

Variable

Inserts an Intermediate Variable which is equivalent to a regular (non-inline) user-defined function. Variables are structural components, without instance files, and are used to simplify the mapping process (see Intermediate variables).

Sort: Nodes/Rows

Inserts a component which allows you to sort nodes (see Sort Nodes/Rows).

Filter: Nodes/Rows

Inserts a component that uses two input and output parameters: node/row and bool, and on-true, on-false. If the Boolean is true, then the value of the node/row parameter is forwarded to the on-true parameter. If the Boolean is false, then the complement value is passed on to the on-false parameter. For more information, see Filters and Conditions.

SQL-WHERE/ORDER

Inserts a component which allows you to filter database data conditionally (see SQL WHERE / ORDER Component).

Value-Map

Inserts a component that transforms an input value to an output value using a lookup table. This is useful when you need to map a set of values to another set of values (for example, month numbers to month names). For more information, see Using Value-Maps.
IF-Else Condition

Inserts a component of type “If-Else Condition” (see Filters and Conditions).

Exception

The exception component allows you to interrupt a mapping process when a specific condition is met, or define Fault messages when using WSDL mapping projects. Please see Adding Exceptions and Web Service Faults for more information.
15.4 Project

MapForce supports the Multiple Document Interface and allows you to group your mappings into mapping projects (see Working with Mapping Projects).

Reload Project
Reloads the currently active project and switches to the Project tab.

Close Project
Closes the currently active project.

Save Project
Saves the currently active project.

Add Files to Project
Allows you to add mappings to the current project through the Open dialog box.

Add Active File to Project
Adds the currently active file to the currently open project.

Create Folder
This option adds a new folder to the current project structure, and only becomes active when this is possible. See Managing Project Folders.

Open Mapping
Opens the currently highlighted/selected mapping in the Project tab.

Create Mapping for Operation
Creates a mapping file for the currently selected operation of the WSDL project. The operation name defined in the WSDL file is supplied in the "Save as" dialog box, which is opened automatically.

Add Mapping file for Operation
Allows you to add a previously saved mapping file to the currently active WSDL operation. Select the mapping file from the "Open" dialog box.

Insert Web Service...
Allows you to insert a Web Service based on an existing WSDL file.
Open file in XMLSpy
Opens the selected WSDL file, highlighted in the Project window, in XMLSpy.

Generate code for entire project
Generates project code for the entire project currently visible in the Project window. Code is generated in the currently selected default language for all of the mapping files *.mfd in each of the folders.

Generate code in...
Generates project code in the language you select from the context menu.

Properties
Opens a dialog box where you can define project-wide settings. See Setting the Code Generation Settings.

Recent projects - 1. 2. etc.
Displays a list of the most recently opened projects.
15.5 Component

Change Root Element
Allows you to change the root element of the XML instance document.

Edit Schema Definition in XMLSpy
Selecting this option, having previously clicked an XML-Schema/document, opens the XML Schema file in the Schema view of XMLSpy where you can edit it.

Edit FlexText Configuration
Opens FlexText and enables you to edit a previously created FlexText file.

Add/Remove/Edit Database Objects
Allows you to add, remove, or change the database objects within the database component.

Create mapping to EDI X12 997
The X12 997 Functional Acknowledgment reports the status of the EDI interchange. All errors encountered during processing of the document are reported in it. MapForce can automatically generate a X12 997 document in the main mapping area for you to send on to the recipient. See Generating an X12 997 Functional Acknowledgment.

Create mapping to EDI X12 999
The X12 999 Implementation Acknowledgment Transaction Set reports HIPAA implementation guide non-compliance, or application errors. Each EDI transaction sent to an organization must be responded to by sending a 999 transaction. See Generating an X12 999 Implementation Acknowledgment.

Refresh
Reloads the structure of the currently active database component from the database.

Add Duplicate Input Before
Inserts a copy/clone of the selected item before the currently selected item. Duplicate items do not have output icons, you cannot use them as data sources. For an example, see Map Multiple Sources to One Target section in the tutorial. Right clicking a duplicate item also allows you to reposition it using the menu items Move Up/Move Down, depending on where the item is.

Add Duplicate Input After
Inserts a copy/clone of the selected item after the currently selected item. Duplicate items do not have output icons, you cannot use them as data sources. For an example, see the Map Multiple Sources to One Target section in the tutorial. Right clicking a duplicate item also allows you to reposition it using the menu items Move Up/Move Down, depending on where the item is.
Remove Duplicate
Removes a previously defined duplicate item. For an example, see the Map Multiple Sources to One Target section in the tutorial.

Database Table Actions
Allows you to define the actions to be performed with the mapped data on the specific target database table. See Database Table Actions Settings for more information.

Query Database
Creates a Select statement based on the table/field you clicked in the database component. Clicking a table/field once makes this command active, and the select statement is automatically placed into the Select window.

Align Tree Left
Aligns all the items along the left hand window border.

Align Tree Right
Aligns all the items along the right hand window border. This display is useful when creating mappings to the target schema.

Properties
Opens a dialog box which displays the settings of the currently selected component. See Changing the Component Settings.
15.6 Connection

Auto Connect Matching Children
Activates or deactivates the "Auto Connect Matching Children" option, as well as the icon in the icon bar.

Settings for Connect Matching Children
Opens the Connect Matching Children dialog box in which you define the connection settings (see Connecting matching children).

Connect Matching Children
This command allows you to create multiple connectors for items of the same name, in both the source and target schemas. The settings you define in this dialog box are retained, and are applied when connecting two items, if the "Auto connect child items" icon in the title bar is active. Clicking the icon switches between an active and inactive state. For further information, see Connecting matching children.

Target Driven (Standard)
Changes the connector type to Standard mapping. For further information, see Target Driven (Standard) mapping.

Copy-all (Copy Child Items)
Creates connectors for all matching child items, where each of the child connectors are displayed as a subtree of the parent connector (see Copy-all connections).

Source Driven (Mixed Content)
Changes the connector type to Source Driven (Mixed Content). For further information, see Source Driven (Mixed Content) mapping.

Properties
Opens a dialog box in which you can define the specific (mixed content) settings of the current connector. Unavailable options are greyed out. These settings also apply to complexType items which do not have any text nodes. For further information, see Connection settings.
15.7 Function

Create User-Defined Function
Creates a new user-defined function (see User-Defined Functions).

Create User-Defined Function from Selection
Creates a new user-defined function based on the currently selected elements in the mapping window.

Function Settings
Opens the settings dialog box of the currently active user-defined function allowing you to change its settings.

Remove Function
Deletes the currently active user-defined function if you are working in a context which allows this.

Insert Input
When the mapping window displays a mapping, this command adds an input component to the mapping (see Simple Input). When the mapping window displays a user-defined function, this command adds an input component to the user-defined function (see Parameters in User-Defined Functions).

Insert Output
When the mapping window displays a mapping, this command adds an output component to the mapping (see Simple Output). When the mapping window displays a user-defined function, this command adds an output component to the user-defined function (see Parameters in User-Defined Functions).
15.8 Output

XSLT 1.0, XSLT 2.0, XQuery, Java, C#, C++, Built-in Execution Engine
Sets the transformation language in which the mapping should be executed (see Selecting a Transformation Language).

Validate Output File
Validates the output XML file against the referenced schema (see Validating the Mapping Output).

Save Output File
Saves the data visible in the Output pane to a file.

Save All Output Files
Saves all the generated output files of dynamic mappings. See Processing Multiple Input or Output Files Dynamically for more information.

Regenerate Output
Regenerates the data visible in the Output pane.

Run SQL-Script
If an SQL script is currently visible in the Output pane, the script executes the mapping to the target database, taking the defined table actions into account.

Insert/Remove Bookmark
Inserts a bookmark at the cursor position in the Output pane.

Next Bookmark
Navigates to the next bookmark in the Output pane.

Previous Bookmark
Navigates to the previous bookmark in the Output pane.

Remove All Bookmarks
Removes all currently defined bookmarks in the Output pane.

Pretty-Print XML Text
Reformats your XML document in the Output pane to give a structured display of the document. Each child node is offset from its parent by a single tab character. This is where the Tab size settings (i.e. inserting as tabs or spaces) defined in the Tabs group, take effect.
Text View Settings
Displays the Text View settings dialog box. This dialog box allows you to customize the text view settings the Output pane, XSLT pane, and XQuery pane, and also shows the currently defined hotkeys that apply in the window. For more information, see Text View Features.
15.9 Debug

Start Debugging (F11)
Starts or continues debugging until a breakpoint is hit or the mapping finishes.

Stop Debugging (Shift + F5)
Stops debugging. This command exits the debug mode and switches MapForce back to standard mode.

Step into (F11)
Executes the mapping until a single step is finished anywhere in the mapping. In the mapping debugger, a step is a logical group of dependent computations which normally produce a single item of a sequence.

Depending on the mapping context, this command roughly translates into "go to the left/go to target child/go to source parent".

Step over (F10)
Continues execution until the current step finishes (or finishes again for another item of the sequence), or an unrelated step finishes. This command steps over computations that are inputs of the current step.

Step out (Shift + F11)
Continues execution until the result of the current step is consumed or a step is executed that is not an input or child of the consumption. This command steps out of the current computation.

Depending on the mapping context, this command roughly translates into "go to the right/go to target parent/go to source child".

Minimal step (Ctrl + F11)
Continues execution until a value is produced or consumed. This command subdivides a step and will typically stop twice for each connection: once when its source produces a value and once when its target consumes it. MapForce does not necessarily compute values in the order the mapping would suggest, so production and consumption events do not always follow each other.
15.10 View

Show Annotations
Displays XML schema annotations (as well as EDI info) in the component window. If the Show Types icon is also active, then both sets of info are show in grid form.

<table>
<thead>
<tr>
<th>F1060</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
</tr>
<tr>
<td>ann.</td>
</tr>
<tr>
<td>thing</td>
</tr>
<tr>
<td>iden.</td>
</tr>
</tbody>
</table>

Show Types
Displays the schema datatypes for each element or attribute. If the Show Annotations icon is also active, then both sets of info are show in grid form.

Show library in Function Header
Displays the library name in parenthesis in the function title.

Show Tips
Displays a tooltip containing explanatory text when the mouse pointer is placed over a function.

XBRL Display Options
See Configuring the XBRL Label Display Options.

Show Selected Component Connectors
Switches between showing all mapping connectors, or those connectors relating to the currently selected components.

Show Connectors from Source to Target
Switches between showing:
- connectors that are directly connected to the currently selected component, or
- connectors linked to the currently selected component, originating from source and terminating at the target components.

Zoom
Opens the Zoom dialog box. You can enter the zoom factor numerically, or drag the slider to change the zoom factor interactively.

Back
Steps back through the currently open mappings of the mapping tab.
Forward
Steps forward through the currently open mappings of the mapping tab.

Status Bar
Switches on/off the Status Bar visible below the Messages window.

Library Window
Switches on/off the Library window.

Messages
Switches on/off the Validation output window. When generating code the Messages output window is automatically activated to show the validation result.

Overview
Switches on/off the Overview window. Drag the rectangle to navigate your Mapping view.

Project window
Switches on/off the Project window.
15.11 Tools

Global Resources
Opens the Manage Global Resources dialog box, where you can add, edit or delete settings applicable across multiple Altova applications (see Altova Global Resources).

Active Configuration
Allows you to select the currently active global resource configuration from a list of configurations previously defined in the Global Resources.

Create Reversed Mapping
Creates a "reversed" mapping from the currently active mapping in MapForce, which is to be the basis of a new mapping. Note that the result is not intended to be a complete mapping, only the direct connections between components are retained in the reversed mapping. It is very likely that the resulting mapping will not be valid or suitable for preview in the Output pane, without manual editing.

When you reverse a mapping, the source component becomes the target component, and target component becomes the source. If an input or output XML instance file have been assigned to a component, then they will be swapped.

The following data is retained:

- Direct connections between components
- Direct connections between components in a chained mapping
- The type of connection: Standard, Mixed content, Copy-All
- Pass-through component settings
- Database components

The following data is not retained:

- Connections via functions, filters, etc, along with the functions, filters, etc.
- User-defined functions
- Web service components

Restore Toolbars and Windows
Resets the toolbars, entry helper windows, docked windows etc. to their defaults. MapForce needs to be restarted for the changes to take effect.

Customize...
Opens a dialog box that lets you to customize the MapForce graphical user interface. This includes showing or hiding toolbars, as well as editing the context menus and keyboard shortcuts (see Customizing Keyboard Shortcuts).

Options
Opens a dialog box where you can change the default MapForce settings (see Changing the
MapForce Options).
15.12 Window

Cascade
This command rearranges all open document windows so that they are all cascaded (i.e. staggered) on top of each other.

Tile Horizontal
This command rearranges all open document windows as horizontal tiles, making them all visible at the same time.

Tile Vertical
This command rearranges all open document windows as vertical tiles, making them all visible at the same time.

This list shows all currently open windows, and lets you quickly switch between them. You can also use the Ctrl-TAB or CTRL F6 keyboard shortcuts to cycle through the open windows.
15.13 Help Menu

- Table of Contents
  - Description
  Opens the onscreen help manual of MapForce with the Table of Contents displayed in the left-hand-side pane of the Help window. The Table of Contents provides an overview of the entire Help document. Clicking an entry in the Table of Contents takes you to that topic.

- Index
  - Description
  Opens the onscreen help manual of MapForce with the Keyword Index displayed in the left-hand-side pane of the Help window. The index lists keywords and lets you navigate to a topic by double-clicking the keyword. If a keyword is linked to more than one topic, a list of these topics is displayed.

- Search
  - Description
  Opens the onscreen help manual of MapForce with the Search dialog displayed in the left-hand-side pane of the Help window. To search for a term, enter the term in the input field, and press Return. The Help system performs a full-text search on the entire Help documentation and returns a list of hits. Double-click any item to display that item.

- Software Activation
  - Description
  After you download your Altova product software, you can license—or activate—it using either a free evaluation key or a purchased permanent license key.

  - Free evaluation key. When you first start the software after downloading and installing it, the Software Activation dialog will pop up. In it is a button to request a free evaluation key-code. Enter your name, company, and e-mail address in the dialog that appears, and click Request Now! The evaluation key is sent to the e-mail address you entered and should reach you in a few minutes. Now enter the key in the key-code field of the Software Activation dialog box and click OK to start working with your Altova product. The software will be unlocked for a period of 30 days.
  - Permanent license key. The Software Activation dialog contains a button to purchase a permanent license key. Clicking this button takes you to Altova's online shop, where you can purchase a permanent license key for your product. There are two types of permanent license: single-user and multi-user. Both will be sent to you by e-mail. A single-user license contains your license-data and includes your name, company, e-mail, and key-code. A multi-user license
contains your license-data and includes your company name and key-code. Note that your license agreement does not allow you to install more than the licensed number of copies of your Altova software on the computers in your organization (per-seat license). Please make sure that you enter the data required in the registration dialog exactly as given in your license e-mail.

Note: When you enter your license information in the Software Activation dialog, ensure that you enter the data exactly as given in your license e-mail. For multi-user licenses, each user should enter his or her own name in the Name field.

Your license email and the different ways to license (activate) your Altova product

The license email that you receive from Altova will contain:

- Your license details (name, company, email, key-code)
- As an attachment, a license file with a .altova_licenses file extension

To activate your Altova product, you can do one of the following:

- Enter the email-supplied license details in the Altova product's Software Activation dialog, and click OK.
- Save the license file (.altova_licenses) to a suitable location, double-click the license file, enter any requested details in the dialog that appears, and finish by clicking Apply Keys.
- Save the license file (.altova_licenses) to any suitable location, and upload it from this location to the license pool of your Altova LicenseServer. You can then either: (i) acquire the license from your Altova product via the product's Software Activation dialog, or (ii) assign the license to the product from Altova LicenseServer.

For more information about licensing via LicenseServer, read the rest of this topic.

The Software Activation dialog (screenshot below) can be accessed at any time by clicking the Help | Software Activation command.

You can activate the software by either:

- Entering the license key information (click Enter a New Key Code), or
- Acquiring a license via an Altova LicenseServer on your network (click Use Altova LicenseServer, located at the bottom of the Software Activation dialog). Select the machine on which the LicenseServer you want to use has been installed. Note that the auto-discovery of License Servers works by means of a broadcast sent out on the LAN. As these broadcasts are limited to a subnet, License Server must be on the same subnet as the client machine for auto-discovery to work. If auto-discovery does not work, then type in the name of the server. The Altova LicenseServer must have a license for your Altova
product in its license pool. If a license is available in the LicenseServer pool, this is indicated in the Software Activation dialog (screenshot below), and you can click **Save** to acquire the license.

![Screenshot of Software Activation dialog](image)

After a machine-specific (aka installed) license has been acquired from a LicenseServer, it cannot be returned to the LicenseServer for a period of seven days. After that time, you can return the machine license to LicenseServer (click **Return License**) so that this license can be acquired from LicenseServer by another client. (A LicenseServer administrator, however, can unassign an acquired license at any time via the administrator's Web UI of LicenseServer.) Note that the returning of licenses applies only to machine-specific licenses, not to concurrent licenses.

**Check out license**

You can check out a license from the license pool for a period of up to 30 days so that the license is stored on the product machine. This enables you to work offline, which is useful, for example, if you wish to work in an environment where there is no access to your Altova LicenseServer (such as when your Altova product is installed on a laptop and you are traveling). While the license is checked out, LicenseServer displays the license as being in use, and the license cannot be used by any other machine. The license automatically reverts to the checked-in state when the check-out period ends. Alternatively, a checked-out license can be checked in at any time via the **Check in** button of the Software Activation dialog.

To check out a license, do the following: (i) In the Software Activation dialog, click **Check out License** (see screenshot above); (ii) In the License Check-out dialog that appears, select the check-out period you want and click **Check out**. The license will be checked out. The Software Activation dialog will display the check-out information, including the time when the check-out period ends. The **Check out License** button in the dialog changes to a **Check In** button. You can check the license in again at any time by clicking **Check In**. Because the
license automatically reverts to the checked-in status, make sure that the check-out period you select adequately covers the period during which you will be working offline.

**Note:** For license check-outs to be possible, it must be enabled on the LicenseServer. If this functionality has not been enabled, you will get an error message to this effect. In this event, contact your LicenseServer administrator.

*Copy Support Code*
Click Copy Support Code to copy license details to the clipboard. This is the data that you will need to provide when requesting support via the online support form.

Altova LicenseServer provides IT administrators with a real-time overview of all Altova licenses on a network, together with the details of each license, as well as client assignments and client usage of licenses. The advantage of using LicenseServer therefore lies in administrative features it offers for large-volume Altova license management. Altova LicenseServer is available free of cost from the Altova website. For more information about Altova LicenseServer and licensing via Altova LicenseServer, see the Altova LicenseServer documentation.

**Order Form**
When you are ready to order a licensed version of the software product, you can use either the Order license key button in the Software Activation dialog (see previous section) or the Help | Order Form command to proceed to the secure Altova Online Shop.

**Registration**
Opens the Altova Product Registration page in a tab of your browser. Registering your Altova software will help ensure that you are always kept up to date with the latest product information.

**Check for Updates**
Checks with the Altova server whether a newer version than yours is currently available and displays a message accordingly.

**Support Center**
A link to the Altova Support Center on the Internet. The Support Center provides FAQs,
discussion forums where problems are discussed, and access to Altova’s technical support staff.

▼ FAQ on the Web

Description
A link to Altova’s FAQ database on the Internet. The FAQ database is constantly updated as Altova support staff encounter new issues raised by customers.

▼ Download Components and Free Tools

Description
A link to Altova’s Component Download Center on the Internet. From here you can download a variety of companion software to use with Altova products. Such software ranges from XSLT and XSL-FO processors to Application Server Platforms. The software available at the Component Download Center is typically free of charge.

▼ MapForce on the Internet

Description
A link to the Altova website on the Internet. You can learn more about MapForce and related technologies and products at the Altova website.

▼ MapForce Training

Description
A link to the Online Training page at the Altova website. Here you can select from online courses conducted by Altova’s expert trainers.

▼ About MapForce

Description
Displays the splash window and version number of your product. If you are using the 64-bit version of MapForce, this is indicated with the suffix (x64) after the application name. There is no suffix for the 32-bit version.
Chapter 16

Code Generator
16 Code Generator

Code Generator is a MapForce built-in feature which enables you to generate Java, C++ or C# code from mapping files designed with MapForce. You can generate code not only from simple mappings with a single data source and target, but also from mappings with multiple sources and multiple targets. The result is a fully-featured and complete application which performs the mapping operation for you. Once you generate the code, you can execute the mapping by running the application directly as generated. You can also import the generated code into your own application, or extend it with your own functionality.

If your mapping uses XML schemas or DTDs, you can also optionally generate schema wrapper libraries (see Generating Code from XML Schemas or DTDs). The schema wrapper libraries enable you to work with XML data in an abstract way, without too much concern for the underlying XML Application Program Interface (API), such as MSXML, Apache Xerces, Microsoft System.Xml, or Java Application for XML Processing (JAXP).
16.1 Introduction to code generator

The primary goal of the generated code is to execute a MapForce mapping. In addition to this, you can optionally generate schema wrapper libraries for XML schemas used by the mapping, which enables you to read or write data to/from XML instances.

The generated code is expressed in C++, Java or C# programming languages.

<table>
<thead>
<tr>
<th>Target Language</th>
<th>C++</th>
<th>C#</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML DOM implementations</td>
<td>MSXML 6.0 Apache Xerces 3</td>
<td>System.Xml</td>
<td>JAXP</td>
</tr>
<tr>
<td>Database API</td>
<td>ADO</td>
<td>ADO.NET</td>
<td>JDBC</td>
</tr>
</tbody>
</table>

**C++**

You can configure whether the C++ generated output should use MSXML 6.0 or Apache Xerces 3. MapForce generates complete project (.vcproj) and solution (.sln) files for all supported versions of Visual Studio (see table above). The generated code optionally supports MFC.

**Note:** When building C++ code for Visual Studio and using a Xerces library precompiled for Visual C++, a compiler setting has to be changed in all projects of the solution:

1. Select all projects in the Solution Explorer.
2. On the Project menu, click Properties.
3. Click Configuration Properties | C/C++ | Language.
4. In the list of configurations, select All Configurations.
5. Change Treat wchar_t as Built-in Type to No (/Zc:wchar_t-)

**C#**

The generated C# code uses the .NET XML classes (System.Xml) and can be used from any .NET capable programming language, such as VB.NET, Managed C++, or J#. Project files can be generated for all supported versions of Visual Studio (see table above).

**Java**

The generated Java output is written against the industry-standard Java API for XML Processing (JAXP) and includes an Ant build file and project files for supported versions of Java and Eclipse (see table above).
**Generated output**
The designated destination folder will include all the libraries and files required to execute the mapping, namely:

- A variable number of Altova libraries required by the mapping (for example, Altova function libraries, database libraries, EDI libraries)
- A complete mapping application. When compiled and run, the application performs the mapping transformation.

**Code generator templates**
Output code is completely customizable via a simple yet powerful [template language](#) (SPL, from Spy Programming Language) which gives full control in mapping XML Schema built-in data-types to the primitive datatypes of a particular programming language. SPL allows you to easily replace the underlying parsing and validating engine, customize code according to your company's writing conventions, or use different base libraries such as the Microsoft Foundation Classes (MFC) and the Standard Template Library (STL).
16.2 What’s new ...

Version 2018

- Added support for Microsoft Visual Studio 2013, 2015, 2017
- End of support for Visual Studio 2005 and Xerces 2.x

Version 2014

- Removal of compatibility mode option for code generation

Version 2011

- Contains bug fixes and enhancements

Version 2010 R3

- Support for Microsoft Visual Studio 2010
- Support for MSXML 6.0 in generated C++ code
- Support for 64-bit targets for C++ and C# projects

Version 2010

- Enumeration facets from XML schemas are now available as symbolic constants in the generated classes (using 2007r3 templates)

Version 2009 sp1

- Apache Xerces version 3.x support added (older versions starting from Xerces 2.6.x are still supported)

Version 2009

- The generated mapping implementation was redesigned to support sequences and grouping. The API has not changed

Version 2008 R2

- Support for generation of Visual Studio 2008 project files for C# and C++ has been added
- Generated MapForce mapping code in C# and Java can use readers/writers, streams, strings or DOM documents as sources and targets

Version 2008

- The new 2007 R3-style SPL templates have been further enhanced:
  o It is now possible to remove single elements
  o Access to schema metadata (e.g. element names, facets, enumerations, occurrence, etc.) is provided
  o Complex types derived by extension are now generated as derived classes

Version 2007 R3
Code Generator has been redesigned for version 2007 release 3 to simplify usage of the generated code, reduce code volume and increase performance.

- Handling of XML documents and nodes with explicit ownership, to avoid memory leaks and to enable multi-threading
- New syntax to avoid name collisions
- New data types for simpler usage and higher performance (native types where possible, new null handling, ...)
- Attributes are no longer generated as collections
- Simple element content is now also treated like a special attribute, for consistency
- New internal object model (important for customized SPL templates)
- Compatibility mode to generate code in the style of older releases
- Type wrapper classes are now only generated on demand for smaller code
16.3 Generating C++ code


- You can generate code either from a single mapping design (.mfd), or from a mapping project (.mfp). If you generate code from a single mapping, the resulting application executes the respective mapping transformation. If you generate code from a MapForce project (.mfp) which includes multiple mappings, the resulting application executes in bulk all mappings included in the project.
- You can change the general code generation options from the Tools | Options menu, Generation tab.
- You can change the name of the generated mapping application and other settings from the File | Mapping settings menu. The default application name is Mapping.
- If your mapping contains database components, you can view database specific settings by clicking a database component, and then selecting the menu option Component | Properties.

A typical C++ solution generated by MapForce includes the following:

- Several Altova-signed libraries required by the mapping (all prefixed with Altova).
- The main mapping project (in this example, Mapping), which includes the mapping application and dependent files.

Sample C++ solution generated with MapForce
This section includes the following topics:

- Generating code from a mapping
- Generating code from a mapping project
- Building the project
- Running the application

### 16.3.1 Generating code from a mapping

**To generate C++ code from a mapping design file (.mfd):**

1. Review and select the desired code generation options (see Code generator options).
2. On the **File** menu, click **Generate code in | C++**.
3. Select a destination directory for the generated files, and then click OK to confirm. The result of code generation (error or success message) is displayed in the Messages window.

The default name of the generated application is **Mapping**. If required, you can change this, and other settings, from the Mapping Settings dialog box (see Changing the mapping settings).

### 16.3.2 Generating code from a mapping project

**To generate code from a mapping project (.mfp):**

1. If you haven't done so already, open the mapping project in MapForce.
2. Right-click the project in the Project window, and then click **Properties**.
3. Review and change the project settings if required (in particular, ensure that the target language and the output directory are set correctly), and then click **OK**.
4. On the **Project** menu, click **Generate code for the Entire Project**.

![Project Settings dialog box]

- **Project Name:** Project01
- **Output Name:** Project01
- **Output Directory:** C:\Project01\output\java
- **Language:** Java
- **Base package name:** com.mapforce

- **OK**
- **Cancel**
The progress and result of the code generation process (error or success message) is displayed in the Messages window.

By default, the name of the generated application is the same as the project name. If the project name contains spaces, these are converted to underscores in the generated code. By default, code is generated in the same directory as the MapForce project, in the output sub-directory.

To change the output directory and the name of the project, click the Project in the Project window, and then select Project | Properties from the menu. If your MapForce project contains folders, you can change the code generation settings for each individual folder (right click on the folder, and then select Properties). Otherwise, all project folders inherit the settings from the MapForce project.

16.3.3 Building the project

Once you generated the C++ code, building it in Visual Studio is the next step. To build the generated code:

1. Open the generated solution (.sln) file in Visual Studio.

   By default, the name of the solution file is Mapping.sln, and it is located in the Mapping subdirectory relative to the directory where you saved the generated code. If you changed the application name from the mapping settings, then the name of the .sln file is changed accordingly. For example, if you changed the application name to MyApplication, then the solution file is called MyApplication.sln, and it is located in the MyApplication subdirectory.

2. On the Build menu, click Configuration Manager.

3. Select the required build configuration (Debug, Release, Unicode Debug, Unicode Release). Note that only Unicode builds support the full Unicode character set in XML and
other files. The non-Unicode builds work with the local codepage of your Windows installation.

4. On the Build menu, click Build Solution.

### 16.3.4 Running the application

Once you compile the Visual Studio project, a command-line application is produced, called `Mapping.exe`. (Note that if you changed the application name from the mapping settings, then the executable name is changed accordingly.)

You can locate the mapping application in one of the following subdirectories relative to the .sln file, depending on the build option you chose:

- Debug
- Release
- Unicode Debug
- Unicode Release

To run the application, open a command prompt, change the current directory to the path of the executable, and run it, for example:

```
C:\codegen\DB_CompletePOcpp\Mapping\Debug>Mapping.exe
Mapping Application
Finished
C:\codegen\DB_CompletePOcpp\Mapping\Debug>
```
16.4 Generating C# code


- You can generate code either from a single mapping design (.mfd), or from a mapping project (.mfp). If you generate code from a single mapping, the resulting application executes the respective mapping transformation. If you generate code from a MapForce project (.mfp) which includes multiple mappings, the resulting application executes in bulk all mappings included in the project.
- You can change the general code generation options from the Tools | Options menu, Generation tab.
- You can change the name of the generated mapping application and other settings from the File | Mapping settings menu. The default application name is Mapping.
- If your mapping contains database components, you can view database specific settings by clicking a database component, and then selecting the menu option Component | Properties.

A typical C# solution generated by MapForce includes the following:

- Several Altova-signed libraries required by the mapping (all prefixed with Altova).
- The main mapping project (in this example, Mapping), which includes the mapping application and dependent files.

![Sample C# solution generated with MapForce](image)

This section includes the following topics:

- Generating code from a mapping
Generating code from a mapping project

To generate C# code from a mapping design file (.mfd):

1. Review and select the desired code generation options (see Code generator options).
2. On the File menu, click Generate code in | C#.
3. Select a destination directory for the generated files, and then click OK to confirm. The result of code generation (error or success message) is displayed in the Messages window.

The default name of the generated application is Mapping. If required, you can change this, and other settings, from the Mapping Settings dialog box (see Changing the mapping settings).

Generating code from a mapping project

To generate code from a mapping project (.mfp):

1. If you haven't done so already, open the mapping project in MapForce.
2. Right-click the project in the Project window, and then click Properties.
3. Review and change the project settings if required (in particular, ensure that the target language and the output directory are set correctly), and then click OK.
4. On the Project menu, click Generate code for the Entire Project.

The progress and result of the code generation process (error or success message) is displayed in the Messages window.
By default, the name of the generated application is the same as the project name. If the project name contains spaces, these are converted to underscores in the generated code. By default, code is generated in the same directory as the MapForce project, in the output sub-directory.

To change the output directory and the name of the project, click the Project in the Project window, and then select Project | Properties from the menu. If your MapForce project contains folders, you can change the code generation settings for each individual folder (right click on the folder, and then select Properties). Otherwise, all project folders inherit the settings from the MapForce project.

### 16.4.3 Building the project

Once you generated the C# code, building it in Visual Studio is the next step. To build the generated code:

1. Open the generated solution (.sln) file in Visual Studio.

   By default, the name of the solution file is `Mapping.sln`, and it is located in the `Mapping` subdirectory relative to the directory where you saved the generated code. If you changed the application name from the mapping settings, then the name of the .sln file is changed accordingly. For example, if you changed the application name to `MyApplication`, then the solution file is called `MyApplication.sln`, and it is located in the `MyApplication` subdirectory.

2. On the Build menu, click Configuration Manager.
3. Select the required build configuration (Debug, Release).
4. On the Build menu, click Build Solution.

### 16.4.4 Running the application

Once you compile the Visual Studio project, a command-line application is produced, called `Mapping.exe`. (Note that if you changed the application name from the mapping settings, then the executable name is changed accordingly.)

You can locate the mapping application in one of the following subdirectories relative to the .sln file, depending on the build option you chose:

- `bin\Debug`
- `bin\Release`

To run the application, open a command prompt, change the current directory to the path of the executable, and run it, for example:

```
C:\codegen\DB_CompletePOCs\Mapping\bin\Release>Mapping.exe
Mapping Application
Connecting to CustomersAndArticles database...
Finished
C:\codegen\DB_CompletePOCs\Mapping\bin\Release>
```
16.5 Generating Java code

You can generate program code for Java 1.7 or later. Note the following when generating code:

- You can generate code either from a single mapping design (.mfd), or from a mapping project (.mfp). If you generate code from a single mapping, the resulting application executes the respective mapping transformation. If you generate code from a MapForce project (.mfp) which includes multiple mappings, the resulting application executes in bulk all mappings included in the project.
- You can change the general code generation options from the Tools | Options menu, Generation tab.
- You can change the name of the generated mapping application and other settings from the File | Mapping settings menu. The default application name is Mapping.
- If your mapping contains database components, you can view database specific settings by clicking a database component, and then selecting the menu option Component | Properties.

A typical Java project generated by MapForce includes the following:

- Several Altova-signed Java packages required by the mapping (all prefixed with com.altova).
- The com.mapforce package, which includes the mapping application and dependent files (as shown below, it is possible to change the name of this package). The two most important files in this package are as follows:
  - The Java mapping application as a dialog application (MappingApplication.java).
  - The Java mapping application as a console application (MappingConsole.java).
- A build.xml file which you can execute with Apache Ant to compile the project and generate JAR files.
This section includes the following topics:

- Generating code from a mapping
- Generating code from a mapping project
- Handling JDBC references
- Building the project with Ant
- Example: Run and compile Java code with Eclipse and Ant

### 16.5.1 Generating code from a mapping

To generate Java code from a mapping design file (.mfd):

1. Review and select the desired code generation options (see Code generator options).
2. On the File menu, click Generate code in | Java.
3. Select a destination directory for the generated files, and then click OK to confirm. The result of code generation (error or success message) is displayed in the Messages window.

The default name of the generated application is Mapping, and the default name of the base package is com.mapforce. If required, you can change these from the Mapping Settings dialog box (see Changing the mapping settings).
16.5.2 Generating code from a mapping project

To generate code from a mapping project (.mfp):

1. If you haven't done so already, open the mapping project in MapForce.
2. Right-click the project in the Project window, and then click Properties.

3. Review and change the project settings if required (in particular, ensure that the target language and the output directory are set correctly), and then click OK.
4. On the Project menu, click Generate code for the Entire Project.

The progress and result of the code generation process (error or success message) is displayed in the Messages window.

By default, the name of the generated application is the same as the project name. If the project name contains spaces, these are converted to underscores in the generated code. By default, code is generated in the same directory as the MapForce project, in the output sub-directory.

To change the output directory and the name of the project, click the Project in the Project window, and then select Project | Properties from the menu. If your MapForce project contains folders, you can change the code generation settings for each individual folder (right click on the folder, and then select Properties). Otherwise, all project folders inherit the settings from the MapForce project.

16.5.3 Handling JDBC references

If the mapping connects to a database through JDBC, ensure that the JDBC drivers used by the mapping are installed on your system (see Creating a JDBC connection). To view the current JDBC settings of any database component, click it, and then select Component | Properties from the menu.
Additionally, if you build JAR files from generated Java code, add the "Class-Path" attribute for your database driver to the build.xml file. This ensures that the reference to the database driver is available in the manifest (MANIFEST.MF) file after you build the project.

To add the "Class-Path" attribute:

1. Add to the build.xml file a reference to the JAR file of the database driver, as a new "Class-Path" attribute. For example, for MySQL 5.1.16, the value of the "Class-Path" attribute looks as follows:

   <attribute name="Class-Path" value="mysql-connector-java-5.1.16-bin.jar"/>

   The manifest element of the build.xml file now looks similar to the screen shot below.

   ```xml
   <manifest file="U:\_TTP\Verification\related\42730\___\SAP_IDoc\CODE_Java\build.xml"
     <attribute name="Created-By" value="MapForce 2014"/>
    <attribute name="Main-Class" value="com.mapforce.Paccar866DBDELFOR"/>
    <attribute name="Class-Path" value="mysql-connector-java-5.1.16-bin.jar"/>
  </manifest>
   
   2. Copy the JAR file of the JDBC driver to the folder that contains the JAR file of the generated application.

16.5.4 Building the project with Ant

Apache Ant is a widely used Java library (and command-line tool) which automates building and compilation of Java projects (see http://ant.apache.org/manual/). Ant works with build files (such files define the sources and targets from which code must be compiled, as well as any specific build options). Since any MapForce-generated project includes a build.xml file recognized by Ant, you can easily build MapForce-generated projects with Ant.

Ant may be available on your system either as a standalone installation, or bundled within Eclipse (or other Java IDEs). For instructions on how to install Ant on your system, see http://ant.apache.org/manual/. For instructions on how to use Ant in Eclipse, refer to the Eclipse tutorial, as well as the Eclipse documentation.

You can quickly check whether the standalone version of Ant (not the one bundled with Eclipse) is available on your system by opening a command prompt and typing ant at the command line. When Ant is already available, the resulting message will be similar to: Buildfile: build.xml does not exist! This message indicates that Ant is installed and it is attempting to build a build.xml file, but the latter does not exist in the current directory. If you run Ant from a directory which includes a build.xml file, Ant executes the build.xml file instead, with whatever build options are defined in it.

To build a MapForce-generated Java project with Ant:

1. Open a command prompt and navigate to the directory where the Java project was
generated (note that the directory must contain the build.xml file).

2. At the command prompt, enter ant. This will compile and execute the Java code according to the options defined in the build.xml file.

To generate a JAR file with Ant:

- At the command prompt, enter ant jar.

If you need help with Ant command syntax and options, enter ant –help at the command line.

16.5.5 Example: Build a Java application with Eclipse and Ant

This example walks you through the steps required to generate a Java application with MapForce, and compile it outside of MapForce using the Eclipse Integrated Development Environment (IDE) and Apache Ant. After completing this example, you will have created and compiled a complete Java application which executes one of the mapping samples available by default in MapForce.

If you can already compile successfully other Java applications with Eclipse and Ant, there are no special requirements to run this example. Otherwise, note the following prerequisites:

- The Java Development Kit (JDK), Eclipse (https://www.eclipse.org/), and Ant (http://ant.apache.org/) must be installed on your system. Eclipse typically includes support for building projects with Ant (see also Building the project with Ant).
- The Windows PATH environment variable must include the path to the JDK's bin directory (for example, C:Javajdk1.7.0bin). This is a basic requirement for developing applications for the Java platform. For instructions, see http://docs.oracle.com/javase/tutorial/essential/environment/paths.html.

This example uses the following configuration:

- JDK 1.7
- Eclipse IDE for Java Developers (Luna Service Release 4.4.1)
- Ant 1.9.2, which is already integrated into the above-mentioned edition of Eclipse; therefore, it was not installed and configured separately.

The example is organized into the following sub-tasks:
16.5.5.1 Step 1: Generate Java code

To generate the Java code in MapForce:

1. On the File menu, click Open, and browse for the CompletePO.mfd mapping available in the <Documents>\Altova\MapForce2019\MapForceExamples\ directory.
2. On the Output menu, click Java. This changes the transformation language to Java.
3. On the File menu, click Generate code in | Java. When prompted, browse for the directory to which the Java project must be saved. For convenience, you may choose to save the project to C:\workspace\CompletePO\ (where C:\workspace is your default Eclipse workspace).

16.5.5.2 Step 2: Import the project into Eclipse

To import the project into Eclipse:

1. If you haven't done so already, run Eclipse and switch to the default Java perspective using the menu command Window | Open Perspective.
2. On the File menu, click Import, and then select Existing Projects into Workspace.
3. Click **Next**.
4. Browse for the folder where you have previously saved the generated code, and then click **Finish**. The Java project created by MapForce is now available in the Package Explorer view. If you cannot see the Package Explorer view, display it using the menu command **Window | Show View | Package Explorer**.
16.5.5.3 Step 3: Run the project as dialog application

To run the Java project as a GUI application:

1. In the Package Explorer view of Eclipse, click the MappingApplication.java file available in the com.mapforce package.
2. On the Run menu, click Run As | Java application.
3. On the MapForce application window, click Start to execute the mapping.
If Eclipse encounters system configuration or run-time errors, you will be prompted. Otherwise, the Java application executes the mapping transformation and generates the `CompletePO.xml` at the output path (in this example: `C:\workspace\CompletePO\CompletePO.xml`).

### 16.5.5.4 Step 4: Run the project as console application

To run the Java project as a console application:

1. In the Package Explorer view of Eclipse, click the `MappingConsole.java` file available in the `com.mapforce` package.
2. On the Run menu, click Run As | Java application.

If Eclipse detects system configuration or run-time errors, you will be prompted. Otherwise, the Java application executes the mapping transformation and generates the `CompletePO.xml` at the output path (in this example: `C:\workspace\CompletePO\CompletePO.xml`).

### 16.5.5.5 Step 5: Build the JAR file with Ant

To build the .jar file with Ant:

1. In the Package Explorer view of Eclipse, click the `build.xml` file available directly in the project root.
2. On the Run menu, click Run.
3. In the Run As dialog box, two possible options to run the Ant build file are displayed. If you choose the first option, Eclipse launches the Ant build with the default settings. If you choose the second option, you can change the settings of the Ant build before launching it. Select the second option.

4. Click to enable the targets that you wish to include in the Ant build. In this example, the targets test and jar are selected.
5. Click **Run**. Eclipse executes the Ant build file and displays the result in the Console view.
16.6 Integrating MapForce-Generated Code

MapForce-generated code can be integrated, or adapted to your specific application, even though the result of code generation is a complete and fully-functioning application. Some typical scenarios where you might want to change the generated code are as follows:

- Define custom source or target files for the mapping application
- Add custom error handling code
- In C# or Java generated code, you can also change the data type of the mapping input programmatically (for example, from string to stream).

This section provides instructions on how to achieve these goals, based on the DB_CompletePO.mfd sample mapping available in the <Documents>\Altova\MapForce2019\MapForceExamples\ directory.

As illustrated above, the sample mapping consists of two sources and one target:

- ShortPO.xml is a source XML file
- CustomersAndArticles.mdb is a source database
- CompletePO.xml is the target XML file.

In the generated code, these sources and targets will translate to two input and one output parameters supplied to the run method which executes the mapping (as described in the subsequent topics). For now, note the following basic points about code generation:

- The number of source and targets in the mapping design corresponds to the number of mapping parameters to the run method in the generated code.
If you change the number of sources or targets of the mapping, then you will need to re-generate the code accordingly.

If you make changes to the generated code, and then re-generate the code at the same location, all changes will be overwritten.

If a mapping includes database components, the generated `run` method includes the database connection object at the appropriate location. For example, if the mapping uses three sources (text content, XML content and a database) to map to a single output file, MapForce generates the following `run` method:

```java
void run(Input in1, Input in2, java.sql.Connection dbConn, Output out1);
```

The argument order is important. As you will see in the subsequent examples, you can modify `dbConn` parameters, or use the default parameters generated by MapForce when integrating your code.

### 16.6.1 Java example

This example uses Eclipse as Java IDE. To begin, generate Java code from the `DB_CompletePO` sample mapping available in the `<Documents>\Altova\MapForce2019\MapForceExamples\` directory, and then import the project into Eclipse.
To edit the generated Java console application, locate the `main(String[] args)` method of your generated application (see the screen shot above). If you did not change the default base package name before generating code, this method is in the `MappingConsole` class of the `com.mapforce` package. Otherwise, it is in the `MappingConsole` class of your custom defined package.

To edit the generated Java dialog application, locate the place in the code where the `run` method is invoked from your generated application. If you did not change the default base package name before generating code, the `run` method is invoked from the class called `MappingFrame.java` of the `com.mapforce` package.

The following code sample illustrates an extract from the `main` method in the generated Java console application. The mapping sources and targets are highlighted in yellow and are defined as parameters to the `run` method. Since this mapping uses a database connection, the corresponding parameter has a special structure. Namely, the connection consists of the connection string (in this case, `jdbc:odbc;;DRIVER=Microsoft Access Driver (*.mdb);DBQ=CustomersAndArticles.mdb`), as well as two empty arguments intended for the `Username` and `Password` (in clear text) for those databases where this data is necessary.

Note that the file paths in the code below have been changed from absolute to relative.

```java
com.altova.io.Input ShortPO2Source =
com.altova.io.InputStream.createInput("ShortPO.xml");
```
com.altova.io.FileOutput("CompletePO.xml");

MappingMapToCompletePOObject.run(
    com.altova.db.Dbs.newConnection(
        "jdbc:odbc;DRIVER=Microsoft Access Driver (*.mdb);DBQ=CustomersAndArticles.mdb",
        "",
        ""),
    ShortPO2Source,
    CompletePO2Target);

To define custom mapping source or target files:

- Locate the parameters passed to the run method and edit them as required. In the sample above, com.altova.db.Dbs.newConnection and ShortPO2Source is the mapping input and CompletePO2Target is the mapping output.

To add extra error handling code:

- Edit the code below the catch (Exception e) code (in case of a Java console application)
- Edit the code below the catch (Exception ex) code (in case of a Java dialog application)

For instructions on how to change the data type of parameters supplied as mapping input/output, see Changing the data type of the mapping input/output (C#, Java).

16.6.2 C# example

This example uses the Visual Studio 2010 IDE. To begin, generate C# code from the DB_CompletePO sample mapping available in the <Documents>\Altova\MapForce2019\MapForceExamples\ directory, and then open the solution in Visual Studio.

By default, the name of the solution file is Mapping.sln, and it is located in the Mapping subdirectory relative to the directory where you saved the generated code. If you changed the application name from the mapping settings, then the name of the .sln file is changed accordingly. For example, if you changed the application name to MyApplication, then the solution file is called MyApplication.sln, and it is located in the MyApplication subdirectory.
Sample C# solution generated with MapForce

Open the MappingConsole.cs file, and locate the `main(String[] args)` method. The following code sample illustrates an extract from the `main` method. The mapping sources and targets are highlighted in yellow and are defined as parameters to the `Run` method. Since this mapping reads data from a database, there is also an input parameter which is a database connection string. If necessary, you can modify the connection string of the database.

Note that the file paths in the code below have been changed from absolute to relative.

```csharp
Altova.IO.Input ShortPO2Source = Altova.IO.StreamInput.createInput("ShortPO.xml");
Altova.IO.Output CompletePO2Target = new Altova.IO.FileOutput("CompletePO.xml");

MappingMapToCompletePOObject.Run(
    "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=CustomersAndArticles.mdb; ",
    ShortPO2Source, CompletePO2Target);
```

To define custom mapping source or target files:

- Locate the parameters passed to the `Run` method and edit them as required. In the sample above, the mapping input is a connection string to the `CustomersAndArticles.mdb` and `ShortPO2Source`. The mapping output is `CompletePO2Target`.
To add extra error handling code:

- Edit the code below the `catch (Exception e)` code

For instructions on how to change the data type of parameters supplied as mapping input/output, see Changing the data type of the mapping input/output (C#, Java).

16.6.3 C++ example

This example uses the Visual Studio 2010 IDE. To begin, generate C++ code from the DB_CompletePO sample mapping available in the `<Documents>\Altova\MapForce2019\MapForceExamples` directory, and then open the solution in Visual Studio.

By default, the name of the solution file is `Mapping.sln`, and it is located in the `Mapping` subdirectory relative to the directory where you saved the generated code. If you changed the application name from the mapping settings, then the name of the .sln file is changed accordingly. For example, if you changed the application name to `MyApplication`, then the solution file is called `MyApplication.sln`, and it is located in the `MyApplication` subdirectory.

Sample C++ solution generated with MapForce

Open the `Mapping.cpp` file, and locate the `_tmain` method. The following code sample illustrates an extract from this method. The mapping sources and targets are defined as parameters to the `Run` method. Since this mapping reads data from a database, there is also an input parameter which is a database connection string. If necessary, you can modify the connection string of the database.
Note that the file paths in the code below have been changed from absolute to relative.

```csharp
MappingMapToCompletePO MappingMapToCompletePOObject;
    MappingMapToCompletePOObject.Run(
        _T("Provider=Microsoft.Jet.OLEDB.4.0; Data Source=CustomersAndArticles.mdb; "),
        _T("ShortPO.xml"),
        _T("CompletePO.xml"));
```

To define custom mapping source or target files:

- Locate the parameters passed to the `Run` method and edit them as required. In the code sample above, the mapping input is a connection string to the `CustomersAndArticles.mdb` database and `_T("ShortPO.xml")`. The mapping output is `_T("CompletePO.xml")`.

To add extra error handling code:

- Edit the code below the `catch (CAltovaException& e)` code.

### 16.6.4 Changing the data type of the mapping input/output (C#, Java)

This topic provides details on the object types you can use programmatically, if you intend to run MapForce mappings from a custom Java or C# application.

You can use several input and output objects (such as files, strings, DOM documents, and others) as parameters to the `run` method. The `run` method is the most important function of generated mapping classes. It has one parameter for each static source or input component in the mapping, and a final parameter for the output component. Components that process multiple files do not appear as parameters to the `run` method, because in this case the file names are processed dynamically inside the mapping.

The objects that you can provide as parameters to the `run` method are available in the `com.altova.io` package (Java) and Altova.IO namespace (C#). The base classes of the generated input and output objects are as follows:

**C#**

```csharp
Altova.IO.Input
Altova.IO.Output
```

**Java**

```java
com.altova.io.Input
com.altova.io.Output
```

The object types supported as input/output parameters to the `run` method, including their applicable input/output file formats, are listed in the following table.
<table>
<thead>
<tr>
<th>Object Type</th>
<th>XML</th>
<th>Microsoft Excel*</th>
<th>EDI**</th>
<th>FlexText*</th>
<th>CSV</th>
<th>Fixed-length files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Binary stream objects</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Strings</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>I/O Reader/Writer (character stream objects)</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>DOM documents</td>
<td>Y</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

* Formats supported only in MapForce Enterprise Edition
** Includes X12 and HL7

Files
File objects (identified in the code file names) have the following definition:

C#

```csharp
Altova.IO.FileInput(string filename)
Altova.IO.FileOutput(string filename)
```

Java

```java
com.altova.io.FileInput(String filename)
com.altova.io.FileOutput(String filename)
```

Binary stream objects
Binary stream objects in the generated code represent an alternative way to working with physical files; there are no advantages as far as memory use is concerned. Binary stream objects have the following definition:

C#

```csharp
Altova.IO.StreamInput(System.IO.Stream stream)
Altova.IO.StreamOutput(System.IO.Stream stream)
```

Java

```java
com.altova.io.StreamInput(java.io.InputStream stream)
com.altova.io.StreamOutput(java.io.OutputStream stream)
```

Notes:

- Binary stream objects are expected to be opened and ready-to-use before calling the `run` method.
- By default, the `run` method closes the stream when finished. To prevent this behaviour, insert the following code before calling the `run` method:

Java

```java
MappingMapToSomething.setCloseObjectsAfterRun(false);  // Java
```
Strings
String objects have the following definition:

C#
Altova.IO.StringInput(string content)
Altova.IO.StringOutput(StringBuilder content)

Java
com.altova.io.StringInput(String xmlcontent)
com.altova.io.StringOutput()

In Java, StringOutput does not take an argument. Content can be accessed with:

```
// mapping from String to (another) String
String MyText = "<here>is some XML text</here>";
Input input = new StringInput(MyText);
Output output = new StringOutput();
MappingMapToMyText.run(input, output);
String myTargetData = output.getString().toString();
```

The getString() method returns a StringBuffer, hence the need for toString().

In C#, StringOutput takes an argument (StringBuilder) which you need to provide beforehand. The StringBuilder may already contain data, so the mapping output is appended to it.

Excel sources/targets cannot map to or from strings.

I/O Reader/Writer (character stream objects)
Character stream objects have the following definition:

C#
Altova.IO.ReaderInput(System.IO.TextReader reader)
Altova.IO.WriterOutput(System.IO.TextWriter writer)

Java
com.altova.io.ReaderInput(java.io.Reader reader)
com.altova.io.WriterOutput(java.io.Writer writer)

Notes:
- Character stream objects are expected to be opened and ready-to-use before calling the run method.
• Excel sources/targets cannot be read from, or written to, character streams.
• By default, the run method closes the stream when finished. To prevent this behaviour, insert the following code before calling the run method:

Java

```java
MappingMapToSomething.setCloseObjectsAfterRun(false); // Java
```

C#

```csharp
MappingMapToSomething.CloseObjectsAfterRun = false; // C#
```

DOM documents

DOM documents have the following definition:

C#

```csharp
Altova.IO.DocumentInput(System.Xml.XmlDocument document)
Altova.IO.DocumentOutput(System.Xml.XmlDocument document)
```

Java

```java
```

Notes:

• The document passed to the DocumentOutput constructor as target must be empty.
• After calling run, the DOM Document generated by the constructor of DocumentOutput already contains mapped data so "save to document" is not necessary. After mapping, you can manipulate the document as necessary.
• Only XML content can be mapped to DOM documents.

Example

Let's assume you want to integrate the code generated by MapForce into your Java application. Your MapForce mapping consists of two source XML files and a target text file. When you generate the MapForce code, the run function looks as follows:

```java
void run(Input in1, Input in2, Output out1);
```

Let's also assume that your application requires that you map data from a local file and binary stream into a character stream. Since data is supplied from other sources, your application must declare the sources and targets as:

```java
String filename; // Declare the source of the first input
Java.io.InputStream stream; // Declare the source of the second input
Java.io.Writer writer; // Declare the output as character stream
```

The following wrappers must be constructed for the MapForce-generated run function:
Now you can call the MapForce generated run function:

```
MappingMapToSomething.run(input1, input2, output1);
```

The C# behavior is almost identical, except that `run` is called `Run`, and the .NET stream and reader/writer classes are named differently.

Using the same technique, you can also use other input and output types, such as strings or DOM documents.
16.7 Generating Code from XML Schemas or DTDs

When you generate code from a mapping, MapForce generates a complete application that executes all steps of the mapping automatically. Optionally, you can generate libraries for all the XML schemas used in the mapping. These allow your code to easily create or read XML instances that are used or created by the mapping code.

To generate libraries for all the XML schemas used in the mapping, select the **Generate Wrapper Classes** check box in the Options dialog (see Code Generator Options). Next time when you generate code, MapForce will create not only the mapping application, but also wrapper classes for all schemas used in the mapping, as follows:

<table>
<thead>
<tr>
<th>C++ or C#</th>
<th>Java</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altova</td>
<td>com.altova</td>
<td>Base library containing common runtime support, identical for every schema.</td>
</tr>
<tr>
<td>AltovaXML</td>
<td>com.altova.xml</td>
<td>Base library containing runtime support for XML, identical for every schema.</td>
</tr>
<tr>
<td>YourSchema</td>
<td>com.YourSchema</td>
<td>A library containing declarations generated from the input schema, named as the schema file or DTD. This library is a DOM (W3C Document Object Model) wrapper that allows you to read, modify and create XML documents easily and safely. All data is held inside the DOM, and there are methods for extracting data from the DOM, and to update and create data into the DOM. The generated C++ code supports either Microsoft MSXML or Apache Xerces 3. The syntax for using the generated code is identical for both DOM implementations. The generated C# code uses the .NET standard System.XML library as the underlying DOM implementation. The generated Java code uses JAXP (Java API for XML Processing) as the underlying DOM interface.</td>
</tr>
</tbody>
</table>

While prototyping an application from a frequently changing XML schema, you may need to frequently generate code to the same directory, so that the schema changes are immediately reflected in the code. Note that the generated test application and the Altova libraries are overwritten every time when you generate code into the same target directory. Therefore do not add code to the generated test application. Instead, integrate the Altova libraries into your project (see Integrating Schema Wrapper Libraries).

In addition to the base libraries listed above, some supporting libraries are also generated. The supporting libraries are used by the Altova base libraries and are not meant for custom
integrations, since they are subject to change.

Name generation and namespaces

MapForce generates classes corresponding to all declared elements or complex types which redefine any complex type in your XML Schema, preserving the class derivation as defined by extensions of complex types in your XML Schema. In the case of complex schemas which import schema components from multiple namespaces, MapForce preserves this information by generating the appropriate C# or C++ namespaces or Java packages.

Generally, the code generator tries to preserve the names for generated namespaces, classes and members from the original XML Schema. Characters that are not valid in identifiers in the target language are replaced by a "_". Names that would collide with other names or reserved words are made unique by appending a number. Name generation can be influenced by changing the default settings in the SPL (Spy Programming Language) template.

The namespaces from the XML Schema are converted to packages in Java or namespaces in C# or C++ code, using the namespace prefix from the schema as code namespace. The complete library is enclosed in a package or namespace derived from the schema file name, so you can use multiple generated libraries in one program without name conflicts.

Data Types

XML Schema has a more elaborate data type model than Java, C# or C++. Code Generator converts the built-in XML Schema types to language-specific primitive types, or to classes delivered with the Altova library. Complex types and derived types defined in the schema are converted to classes in the generated library. Enumeration facets from simple types are converted to symbolic constants.

The mapping of simple types can be configured in the SPL template, see SPL (Spy Programming Language).

If your XML instance files use schema types related to time and duration, these are converted to Altova native classes in the generated code. For information about the Altova library classes, see:

- Reference to Generated Classes (C++)
- Reference to Generated Classes (C#)
- Reference to Generated Classes (Java)

For information about type conversion and other details applicable to each language, see:

- About Schema Wrapper Libraries (C++)
- About Schema Wrapper Libraries (C#)
- About Schema Wrapper Libraries (Java)

Memory management

A DOM tree is comprised of nodes, which are always owned by a specific DOM document - even if the node is not currently part of the document's content. All generated classes are references to the DOM nodes they represent, not values. This means that assigning an instance of a generated class does not copy the value, it only creates an additional reference to the same data.
XML Schema support
The following XML Schema constructs are translated into code:

a) XML namespaces

b) Simple types:
   - Built-in XML schema types
   - Simple types derived by extension
   - Simple types derived by restriction
   - Facets
   - Enumerations
   - Patterns

c) Complex types:
   - Built-in anyType node
   - User-defined complex types
   - Derived by extension: Mapped to derived classes
   - Derived by restriction
   - Complex content
   - Simple content
   - Mixed content

The following advanced XML Schema features are not supported (or not fully supported) in generated wrapper classes:

- Wildcards: `xs:any` and `xs:anyAttribute`
- Content models (sequence, choice, all). Top-level compositor is available in SPL (Spy Programming Language), but is not enforced by generated classes.
- Default and fixed values for attributes. These are available in SPL (Spy Programming Language), but are not set or enforced by generated classes.
- The attributes `xsi:type`, abstract types. When you need to write the xsi:type attribute, use the `SetXsiType()` method of the generated classes.
- Union types: not all combinations are supported.
- Substitution groups are partially supported (resolved like "choice").
- Attribute `nullable="true"` and `xsi:nil`
- Uniqueness constraints
- Identity constraints (key and keyref)

16.7.1 About Schema Wrapper Libraries (C++)

Character Types
The generated C++ code can be compiled with or without Unicode support. Depending on this setting, the types `string_type` and `tstring` will both be defined as `std::string` or `std::wstring`, consisting of narrow or wide characters. To use Unicode characters in your XML file that are not representable with the current 8-bit character set, Unicode support must be enabled. Pay special attention to the `_T()` macros. This macro ensures that string constants are stored correctly, whether you're compiling for Unicode or non-Unicode programs.
### Data Types
The default mapping of XML Schema types to C++ data types is:

<table>
<thead>
<tr>
<th>XML Schema</th>
<th>C++</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>xs:string</td>
<td>string_type</td>
<td>string_type is defined as std::string or std::wstring</td>
</tr>
<tr>
<td>xs:boolean</td>
<td>bool</td>
<td></td>
</tr>
<tr>
<td>xs:decimal</td>
<td>double</td>
<td>C++ does not have a decimal type, so double is used.</td>
</tr>
<tr>
<td>xs:float, xs:double</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>xs:integer</td>
<td>__int64</td>
<td>xs:integer has unlimited range, mapped to __int64 for efficiency reasons.</td>
</tr>
<tr>
<td>xs:nonNegativeInteger</td>
<td>unsigned __int64</td>
<td>see above</td>
</tr>
<tr>
<td>xs:int</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>xs:unsignedInt</td>
<td>unsigned int</td>
<td></td>
</tr>
<tr>
<td>xs:dateTime, date,</td>
<td>altova::DateTime</td>
<td></td>
</tr>
<tr>
<td>time, gYearMonth,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gYear, gMonthDay,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gDay, gMonth</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xs:duration</td>
<td>altova::Duration</td>
<td></td>
</tr>
<tr>
<td>xs:hexBinary and</td>
<td>std::vector&lt;unsigned char&gt;</td>
<td>Encoding and decoding of binary data is done automatically.</td>
</tr>
<tr>
<td>xs:base64Binary</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xs:anySimpleType</td>
<td>string_type</td>
<td></td>
</tr>
</tbody>
</table>

All XML Schema types not contained in this list are derived types, and mapped to the same C++ type as their respective base type.

### Generated Classes
For each type in the schema, a class is generated that contains a member for each attribute and element of the type. The members are named the same as the attributes or elements in the original schema (in case of possible collisions, a number is appended). For simple types, assignment and conversion operators are generated. For simple types with enumeration facets, the methods `GetEnumerationValue()` and `SetEnumerationValue(int)` can be used together with generated constants for each enumeration value. In addition, the method `StaticInfo()` allows accessing schema information as one of the following types:

- `altova::meta::SimpleType`
- `altova::meta::ComplexType`

Classes generated from complex types include the method `SetXsiType()`, which enables you to
set the xsi:type attribute of the type. This method is useful when you want to create XML instance elements of a derived type.

In addition to the classes for the types declared in the XML Schema, a document class (identified with "CDoc" below) is generated. It contains all possible root elements as members, and various other methods. For more information about the class, see \[YourSchema]\::[CDoc].

**Note:** The actual class name depends on the name of the .xsd schema.

For each member attribute or element of a schema type, a new class is generated. For more information about such classes, see:

\[YourSchema]\::MemberAttribute
\[YourSchema]\::MemberElement

**Note:** The actual class names depend on the name of the schema attribute or element.

See also [Example: Using the Schema Wrapper Libraries](#).

**Error Handling**

Errors are reported by exceptions. The following exception classes are defined in the namespace altova:

<table>
<thead>
<tr>
<th>Class</th>
<th>Base Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>std::logic_error</td>
<td>Internal program logic error (independent of input data).</td>
</tr>
<tr>
<td>Exception</td>
<td>std::runtime_error</td>
<td>Base class for runtime errors.</td>
</tr>
<tr>
<td>InvalidArgumentsException</td>
<td>Exception</td>
<td>A method was called with invalid argument values.</td>
</tr>
<tr>
<td>ConversionException</td>
<td>Exception</td>
<td>Exception thrown when a type conversion fails.</td>
</tr>
<tr>
<td>StringParseException</td>
<td>ConversionException</td>
<td>A value in the lexical space cannot be converted to value space.</td>
</tr>
<tr>
<td>ValueNotRepresentableException</td>
<td>ConversionException</td>
<td>A value in the value space cannot be converted to lexical space.</td>
</tr>
<tr>
<td>OutOfRangeException</td>
<td>ConversionException</td>
<td>A source value cannot be represented in target domain.</td>
</tr>
<tr>
<td>InvalidOperationException</td>
<td>Exception</td>
<td>An operation was attempted that is not valid in the given context.</td>
</tr>
<tr>
<td>DataSourceUnavailableException</td>
<td>Exception</td>
<td>A problem occurred while loading an XML instance.</td>
</tr>
</tbody>
</table>
| DataTargetUnavailableException             | Exception        | A problem occurred while saving.
All exception classes contain a message text and a pointer to a possible inner exception.

<table>
<thead>
<tr>
<th>Method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>string_type message()</td>
<td>Returns a textual description of the exception.</td>
</tr>
<tr>
<td>std::exception inner()</td>
<td>Returns the exception that caused this exception, if available, or NULL.</td>
</tr>
</tbody>
</table>

**Accessing schema information**

The generated library allows accessing static schema information via the following classes. All methods are declared as `const`. The methods that return one of the metadata classes return a NULL object if the respective property does not exist.

```cpp
altova::meta::Attribute
altova::meta::ComplexType
altova::meta::Element
altova::meta::SimpleType
```

### 16.7.2 About Schema Wrapper Libraries (C#)

The default mapping of XML Schema types to C# data types is as follows.

<table>
<thead>
<tr>
<th>XML Schema</th>
<th>C#</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>xs:string</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>xs:boolean</td>
<td>bool</td>
<td></td>
</tr>
<tr>
<td>xs:decimal</td>
<td>decimal</td>
<td>xs:decimal has unlimited range and precision, mapped to decimal for efficiency reasons.</td>
</tr>
<tr>
<td>xs:float, xs:double</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>xs:long</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>xs:unsignedLong</td>
<td>ulong</td>
<td></td>
</tr>
<tr>
<td>xs:int</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>xs:unsignedInt</td>
<td>uint</td>
<td></td>
</tr>
<tr>
<td>xs:dateTime, date, time, gYearMonth, gYear, gMonthDay, gDay, gMonth</td>
<td>Altova.Types.DateTime</td>
<td></td>
</tr>
<tr>
<td>xs:duration</td>
<td>Altova.Types.Duration</td>
<td></td>
</tr>
</tbody>
</table>
Generating Code from XML Schemas or DTDs

XML Schema | C# | Remarks
--- | --- | ---
xs:hexBinary and xs:base64Binary | byte[] | Encoding and decoding of binary data is done automatically.

All XML Schema types not contained in this list are derived types, and mapped to the same C# type as their respective base type.

**Generated Classes**

For each type in the schema, a class is generated that contains a member for each attribute and element of the type. The members are named the same as the attributes or elements in the original schema (in case of possible collisions, a number is appended). For simple types, assignment and conversion operators are generated. For simple types with enumeration facets, the methods `GetEnumerationValue()` and `SetEnumerationValue(int)` can be used together with generated constants for each enumeration value. In addition, the method `StaticInfo()` allows accessing schema information as one of the following types:

- `Altova.Xml.Meta.SimpleType`
- `Altova.Xml.Meta.ComplexType`

Classes generated from complex types include the method `SetXsiType()`, which enables you to set the xsi:type attribute of the type. This method is useful when you want to create XML instance elements of a derived type.

In addition to the classes for the types declared in the XML Schema, a document class (identified with “Doc” below) is generated. It contains all possible root elements as members, and various other methods. For more information about the class, see `[YourSchema].[Doc].`

**Note:** The actual class name depends on the name of the .xsd schema.

For each member attribute or element of a schema type, a new class is generated. For more information about such classes, see:

- `[YourSchemaType].MemberAttribute`
- `[YourSchemaType].MemberElement`

**Note:** The actual class names depend on the name of the schema attribute or element.

**Error Handling**

Errors are reported by exceptions. The following exception classes are defined in the namespace `Altova`:

<table>
<thead>
<tr>
<th>Class</th>
<th>Base Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ConversionException</code></td>
<td><code>Exception</code></td>
<td>Exception thrown when a type conversion fails</td>
</tr>
<tr>
<td><code>StringParseException</code></td>
<td><code>ConversionException</code></td>
<td>A value in the lexical space cannot</td>
</tr>
</tbody>
</table>
### Accessing schema information

The generated library allows accessing static schema information via the following classes:

- `Altova.Xml.Meta.Attribute`
- `Altova.Xml.Meta.ComplexType`
- `Altova.Xml.Meta.Element`
- `Altova.Xml.Meta.SimpleType`

The properties that return one of the metadata classes return null if the respective property does not exist.

### 16.7.3 About Schema Wrapper Libraries (Java)

The default mapping of XML Schema types to Java data types is as follows:

<table>
<thead>
<tr>
<th>XML Schema</th>
<th>Java</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xs:string</code></td>
<td><code>String</code></td>
<td></td>
</tr>
<tr>
<td><code>xs:boolean</code></td>
<td><code>boolean</code></td>
<td></td>
</tr>
<tr>
<td><code>xs:decimal</code></td>
<td><code>java.math.BigDecimal</code></td>
<td></td>
</tr>
<tr>
<td><code>xs:float</code>, <code>xs:double</code></td>
<td><code>double</code></td>
<td></td>
</tr>
</tbody>
</table>
### XML Schema

<table>
<thead>
<tr>
<th>XML Schema</th>
<th>Java</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>xs:integer</td>
<td>java.math.BigInteger</td>
<td></td>
</tr>
<tr>
<td>xs:long</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>xs:unsignedLong</td>
<td>java.math.BigInteger</td>
<td>Java does not have unsigned types.</td>
</tr>
<tr>
<td>xs:int</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>xs:unsignedInt</td>
<td>long</td>
<td>Java does not have unsigned types.</td>
</tr>
<tr>
<td>xs:dateTime, date, time, gYearMonth, gYear, gMonthDay, gDay, gMonth</td>
<td>com.altova.types.DateTime</td>
<td></td>
</tr>
<tr>
<td>xs:duration</td>
<td>com.altova.types.Duration</td>
<td></td>
</tr>
<tr>
<td>xs:hexBinary and xs:base64Binary</td>
<td>byte[]</td>
<td>Encoding and decoding of binary data is done automatically.</td>
</tr>
<tr>
<td>xs:anySimpleType</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

All XML Schema types not contained in this list are derived types, and mapped to the same Java type as their respective base type.

### Generated Classes

For each type in the schema, a class is generated that contains a member for each attribute and element of the type. The members are named the same as the attributes or elements in the original schema (in case of possible collisions, a number is appended). For simple types, assignment and conversion operators are generated. For simple types with enumeration facets, the methods `GetEnumerationValue()` and `SetEnumerationValue(int)` can be used together with generated constants for each enumeration value. In addition, the method `StaticInfo()` allows accessing schema information as one of the following types:

- `com.altova.xml.meta.SimpleType`
- `com.altova.xml.meta.ComplexType`

Classes generated from complex types include the method `SetXsiType()`, which enables you to set the `xsi:type` attribute of the type. This method is useful when you want to create XML instance elements of a derived type.

In addition to the classes for the types declared in the XML Schema, a document class (identified with "Doc" below) is generated. It contains all possible root elements as members, and various other methods. For more information about the class, see `com.[YourSchema].[Doc]`.

**Note:** The actual class name depends on the name of the `.xsd` schema.

For each member attribute or element of a schema type, a new class is generated. For more information about such classes, see:
Note: The actual class names depend on the name of the schema attribute or element.

Error Handling
Errors are reported by exceptions. The following exception classes are defined in the namespace com.altova:

<table>
<thead>
<tr>
<th>Class</th>
<th>Base Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SourceInstanceUnavailableException</td>
<td>Exception</td>
<td>A problem occurred while loading an XML instance.</td>
</tr>
<tr>
<td>TargetInstanceUnavailableException</td>
<td>Exception</td>
<td>A problem occurred while saving an XML instance.</td>
</tr>
</tbody>
</table>

In addition, the following Java exceptions are commonly used:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.Error</td>
<td>Internal program logic error (independent of input data)</td>
</tr>
<tr>
<td>java.lang.Exception</td>
<td>Base class for runtime errors</td>
</tr>
<tr>
<td>java.lang.IllegalArgumentException</td>
<td>A method was called with invalid argument values, or a type conversion failed.</td>
</tr>
<tr>
<td>java.lang.ArithmeticException</td>
<td>Exception thrown when a numeric type conversion fails.</td>
</tr>
</tbody>
</table>

Accessing schema information
The generated library allows accessing static schema information via the following classes:

com.altova.xml.meta.Attribute
com.altova.xml.meta.ComplexType
com.altova.xml.meta.Element
com.altova.xml.meta.SimpleType

The properties that return one of the metadata classes return null if the respective property does not exist.

16.7.4 Integrating Schema Wrapper Libraries
To use the Altova libraries in your custom project, refer to the libraries from your project (or include them into your project), as shown below for each language.
C#  
To integrate the Altova libraries into an existing C# project:

1. After MapForce generates code from a schema (for example, YourSchema.xsd), build the generated YourSchema.sln solution in Visual Studio. This solution is in a project folder with the same name as the schema.
2. Right-click your existing project in Visual Studio, and select Add Reference.
3. On the Browse tab, browse for the following libraries: Altova.dll, AltovaXML.dll, and YourSchema.dll located in the output directory of the generated projects (for example, bin\Debug).

C++  
The easiest way to integrate the libraries into an existing C++ project is to add the generated project files to your solution. For example, let’s assume that you generated code from a schema called Library.xsd and selected c:\codegen\cpp\library as target directory. The generated libraries in this case are available at:

- c:\codegen\cpp\library\Altova.vcxproj
- c:\codegen\cpp\library\AltovaXML\AltovaXML.vcxproj
- c:\codegen\cpp\library\Library.vcxproj

First, open the generated c:\codegen\cpp\library\Library.sln solution and build it in Visual Studio.
Next, open your existing Visual Studio solution (in Visual Studio 2010, in this example), right-click it, select **Add | Existing Project**, and add the project files listed above, one by one. Be patient while Visual Studio parses the files. Next, right-click your project and select **Properties**. In the Property Pages dialog box, select **Common Properties | Framework and References**, and then click **Add New Reference**. Next, select and add each of the following projects: **Altova**, **AltovaXML**, and **Library**.

See also the MSDN documentation for using functionality from a custom library, as applicable to your version of Visual Studio, for example:

- If you chose to generate static libraries, see [https://msdn.microsoft.com/en-us/library/ms235627(v=vs.100).aspx](https://msdn.microsoft.com/en-us/library/ms235627(v=vs.100).aspx)
- If you chose to generate dynamic libraries, see [https://msdn.microsoft.com/en-us/library/ms235636(v=vs.100).aspx](https://msdn.microsoft.com/en-us/library/ms235636(v=vs.100).aspx)

The option to generate static or dynamic libraries is available in code generation options (see **Code generator options**).

**Java**

One of the ways to integrate the Altova packages into your Java project is to copy the **com** directory of the generated code to the directory which stores the source packages of your Java project (for example, **C:Workspace\MyJavaProject\src**). For example, let’s assume that you generated code in **c:\codegen\java\library**. The generated Altova classes in this case are available at **c:\codegen\java\library\com**.
After copying the libraries, refresh the project. To refresh the project in Eclipse, select it in the Package Explorer, and press \textit{F5}. To refresh the project in NetBeans IDE 8.0, select the menu command \textit{Source} \textvert \textit{Scan for External Changes}.

Once you perform the copy operation, the Altova packages are available in the Package Explorer (in case of Eclipse), or under “Source Packages” in the Projects pane (in case of NetBeans IDE).

\begin{itemize}
\item \textit{Altova packages in Eclipse 4.4}
\item \textit{Altova packages in NetBeans IDE 8.0.2}
\end{itemize}

16.7.5 Example: Using the Schema Wrapper Libraries

This example illustrates how to use the generated schema wrapper libraries in order to write or read programmatically XML documents conformant to the schema. Before using the sample code,
take some time to understand the structure of the included example schema. You will need this schema to generate the code libraries used in this example. Understanding the example schema will help you get started with the code generated from your schema and adapt it to your needs.

16.7.5.1 Example Schema

The schema used in this example describes a library of books. The complete definition of the schema is shown below. Save this code listing as Library.xsd if you want to get the same results as this example. You will need this schema to generate the code libraries used in this example.

```xml
<?xml version="1.0" encoding="UTF-8"?>
        targetNamespace="http://www.nanonull.com/LibrarySample"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">
    <xs:element name="Library">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Book" type="BookType" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="LastUpdated" type="xs:dateTime"/>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="BookType">
        <xs:sequence>
            <xs:element name="Title" type="xs:string"/>
            <xs:element name="Author" type="xs:string" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="ID" type="xs:integer" use="required"/>
        <xs:attribute name="Format" type="BookFormatType" use="required"/>
    </xs:complexType>
    <xs:complexType name="DictionaryType">
        <xs:complexContent>
            <xs:extension base="BookType">
                <xs:sequence>
                    <xs:element name="FromLang" type="xs:string"/>
                    <xs:element name="ToLang" type="xs:string"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:simpleType name="BookFormatType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Hardcover"/>
            <xs:enumeration value="Paperback"/>
            <xs:enumeration value="Audiobook"/>
            <xs:enumeration value="E-book"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```
Library is a root element of a complexType which can be graphically represented as follows in the schema view of XMLSpy:

As shown above, the library has a LastUpdated attribute (defined as xs:dateTime), and stores a sequence of books. Each book is an xs:complexType and has two attributes: an ID (defined as xs:integer), and a Format. The format of any book can be hardcover, paperback, audiobook, or e-book. In the schema, Format is defined as xs:simpleType which uses an enumeration of the above-mentioned values.

Each book also has a Title element (defined as xs:string), as well as one or several Author elements (defined as xs:string).

The library may also contain books that are dictionaries. Dictionaries have the type DictionaryType, which is derived by extension from the BookType. In other words, a dictionary inherits all attributes and elements of a Book, plus two additional elements: FromLang and ToLang, as illustrated below.
The FromLang and ToLang elements store the source and destination language of the dictionary.

An XML instance file valid according to the schema above could therefore look as shown in the listing below (provided that it is in the same directory as the schema file):

```xml
<?xml version="1.0" encoding="utf-8"?>
  <Book ID="1" Format="E-book">
    <Title>The XMLSpy Handbook</Title>
    <Author>Altova</Author>
  </Book>
  <Book ID="2" Format="Paperback" xmlns:n1="http://www.nanonull.com/LibrarySample" xsi:type="n1:DictionaryType">
    <Title>English-German Dictionary</Title>
    <Author>John Doe</Author>
    <FromLang>English</FromLang>
    <ToLang>German</ToLang>
  </Book>
</Library>
```

The next topics illustrate how to read from such a file programmatically, or write to such a file programmatically. To begin, generate the schema wrapper code from the schema above, using the steps described in Generating Code from XML Schemas or DTD.

### 16.7.5.2 Reading and Writing XML Documents (C++)

After you generate code from the Library schema (see Example Schema), a test C++ application is created, along with several supporting Altova libraries.
About the generated C++ libraries

The following diagram illustrates some of the most important classes of the generated code.

The central class of the generated code is the `CLibrary` class, which represents the XML document. Such a class is generated for every schema and its name depends on the schema file name (`Library.xsd`, in this example). As shown in the diagram, this class provides methods for loading documents from files, binary streams, or strings (or saving documents to files, streams, strings). For a list of all members exposed by this class, see the class reference (`[YourSchema]::[CDoc]`).

The `Library2` field of the `CLibrary` class represents the actual root of the document. The number at the end is meant to avoid a naming conflict with the class name. `Library` is an element in the XML file, so in the C++ code it has a template class as type (`MemberElement`). The template class exposes methods and properties for interacting with the `Library` element. In general, each attribute and each element of a type in the schema is typed in the generated code with the `MemberAttribute` and `MemberElement` template classes, respectively. For more information, see `[YourSchema]::MemberAttribute` and `[YourSchema]::MemberElement` class reference.

The class `CLibraryType` is generated from the schema complex type with the same name, as mentioned in About Schema Wrapper Libraries (C++). Notice that the `CLibraryType` class contains a field `Book`, and a field `LastUpdated`. According to the logic already mentioned above, these correspond to the `Book` element and `LastUpdated` attribute in the schema, and enable you to manipulate programmatically (append, remove, etc) elements and attributes in the instance XML document.
Since the **DictionaryType** is a complex type derived from **BookType** in the schema, this relationship is also reflected in the generated classes. As illustrated in the diagram, the class **CDictionaryType** inherits the **CBookType** class.

If your XML schema defines simple types as enumerations, the enumerated values become available as **Enum** values in the generated code. In the schema used in this example, a book format can be hardcover, paperback, e-book, and so on. Therefore, in the generated code, these values would be available through an **Enum** that is a member of the **CBookFormatType** class.

**Writing an XML document**

1. Open the **LibraryTest.sln** solution in Visual Studio generated from the Library schema mentioned earlier in this example.

   While prototyping an application from a frequently changing XML schema, you may need to frequently generate code to the same directory, so that the schema changes are immediately reflected in the code. Note that the generated test application and the Altova libraries are overwritten every time when you generate code into the same target directory. Therefore do not add code to the generated test application. Instead, integrate the Altova libraries into your project (see [Integrating Schema Wrapper Libraries](#)).

2. In Solution Explorer, open the **LibraryTest.cpp** file, and edit the **Example()** method as shown below.

```cpp
#include <ctime> // required to get current time
using namespace Library; // required to work with Altova libraries

void Example()
{
    // Create a new, empty XML document
    CLibrary libDoc = CLibrary::CreateDocument();

    // Create the root element <Library> and add it to the document
    CLibraryType lib = libDoc.Library2.append();

    // Get current time and set the "LastUpdated" attribute using Altova classes
    time_t t = time(NULL);
    struct tm * now = localtime( & t );
    altova::DateTime dt = altova::DateTime(now->tm_year + 1900, now->tm_mon + 1, now->tm_mday, now->tm_hour, now->tm_min, now->tm_sec);
    lib.LastUpdated = dt;

    // Create a new <Book> and add it to the library
    CBookType book = lib.Book.append();

    // Set the "ID" attribute of the book
    book.ID = 1;

    // Set the "Format" attribute of the <Book> using an enumeration constant
    book.Format.SetEnumerationValue( CBookFormatType::k_Paperback );
}
// Add the <Title> and <Author> elements, and set values
book.Title.append() = _T("The XML Spy Handbook");
book.Author.append() = _T("Altova");

// Append a dictionary (book of derived type) and populate its attributes and elements
CDictionaryType dictionary = CDictionaryType(lib.Book.append().GetNode());
dictionary.ID = 2;
dictionary.Format.SetEnumerationValue(CBookFormatType::k_E_book);
dictionary.Title.append() = _T("English-German Dictionary");
dictionary.Author.append() = _T("John Doe");
dictionary.FromLang.append() = _T("English");
dictionary.ToLang.append() = _T("German");

// Since dictionary a derived type, set the xsi:type attribute of the book element
dictionary.SetXsiType();

// Optionally, set the schema location
libDoc.SetSchemaLocation(_T("Library.xsd"));

// Save the XML document to a file with default encoding (UTF-8),
// "true" causes the file to be pretty-printed.
libDoc.SaveToFile(_T("GeneratedLibrary.xml"), true);

// Destroy the document
libDoc.DestroyDocument();
}

3. Press F5 to start debugging. If the code was executed successfully, a GeneratedLibrary.xml file is created in the solution output directory.

Reading an XML document
2. Save the code below as Library1.xml to a directory that can be read by the program code (for example, the same directory as LibraryTest.sln).

```
<?xml version="1.0" encoding="utf-8"?>
  <Book ID="1" Format="E-book">
    <Title>The XMLSpy Handbook</Title>
    <Author>Altova</Author>
  </Book>
  <Book ID="2" Format="Paperback" xmlns:n1="http://www.nanonull.com/LibrarySample" xsi:type="n1:DictionaryType">
    <Title>English-German Dictionary</Title>
    <Author>John Doe</Author>
    <FromLang>English</FromLang>
  </Book>
</Library>
```
3. In Solution Explorer, open the `LibraryTest.cpp` file, and edit the `Example()` method as shown below.

```cpp
using namespace Library;
void Example()
{
    // Load XML document
    CLibrary libDoc = CLibrary::LoadFromFile(_T("Library1.xml"));

    // Get the first (and only) root element <Library>
    CLibraryType lib = libDoc.Library2.first();

    // Check whether an element exists:
    if (!lib.Book.exists())
    {
        tcout << "This library is empty.\n" << std::endl;
        return;
    }

    // iteration: for each <Book>...
    for (Iterator<CBookType> itBook = lib.Book.all(); itBook; ++itBook)
    {
        // output values of ISBN attribute and (first and only) title element
        tcout << "ID: " << itBook->ID << std::endl;
        tcout << "Title: " << tstring(itBook->Title.first()) << std::endl;

        // read and compare an enumeration value
        if (itBook->Format.GetEnumerationValue() ==
            CBookFormatType::k_Paperback)
        {
            tcout << "This is a paperback book.\n" << std::endl;

            // for each <Author>...
            for (CBookType::Author::iterator itAuthor =
                itBook->Author.all(); itAuthor; ++itAuthor)
            {
                tcout << "Author: " << tstring(itAuthor) << std::endl;

                // alternative: use count and index
                for (unsigned int j = 0; j < itBook->Author.count(); ++j)
                {
                    tcout << "Author: " << tstring(itBook->Author[j]) << std::endl;
                }
            }
        }

        // Destroy the document
        libDoc.DestroyDocument();
    }
}
```

4. Press **F5** to start debugging.
16.7.5.3  Reading and Writing XML Documents (C#)

After you generate code from the Library schema (see Example Schema), a test C# application is created, along with several supporting Altova libraries.

About the generated C# libraries
The following diagram illustrates some of the most important classes of the generated code.
The central class of the generated code is the `Library2` class, which represents the XML document. Such a class is generated for every schema and its name depends on the schema file name (e.g., `Library.xsd`, in this example). Note that this class is called `Library2` to avoid a possible conflict with the namespace name. As shown in the diagram, this class provides methods for loading documents from files, binary streams, or strings (or saving documents to files, streams, or strings). For a description of this class, see the class reference `+[YourSchema].[Doc]`.
The `Library3` member of the `Library2` class represents the actual root of the document. Again, the number at the end is meant to avoid a naming conflict with the class name.

According to the code generation rules mentioned in About Schema Wrapper Libraries (C#), member classes are generated for each attribute and for each element of a type. In the generated code, the name of such member classes is prefixed with `MemberAttribute_` and `MemberElement_`, respectively. In the diagram above, examples of such classes are `MemberAttribute_ID` and `MemberElement_Author`, generated from the `Author` element and `ID` attribute of a book, respectively. Such classes enable you to manipulate programatically the corresponding elements and attributes in the instance XML document (for example, append, remove, set value, etc). For more information, see `[YourSchemaType].MemberAttribute` and `[YourSchemaType].MemberElement` class reference.

Since the `DictionaryType` is a complex type derived from `BookType` in the schema, this relationship is also reflected in the generated classes. As illustrated in the diagram, the class `DictionaryType` inherits the `BookType` class.

If your XML schema defines simple types as enumerations, the enumerated values become available as `Enum` values in the generated code. In the schema used in this example, a book format can be hardcover, paperback, e-book, and so on. Therefore, in the generated code, these values would be available through an `Enum` that is a member of the `BookFormatType` class.

### Writing an XML document

1. Open the `LibraryTest.sln` solution in Visual Studio generated from the Library schema mentioned earlier in this example.

   While prototyping an application from a frequently changing XML schema, you may need to frequently generate code to the same directory, so that the schema changes are immediately reflected in the code. Note that the generated test application and the Altova libraries are overwritten every time when you generate code into the same target directory. Therefore do not add code to the generated test application. Instead, integrate the Altova libraries into your project (see Integrating Schema Wrapper Libraries).

2. In Solution Explorer, open the `LibraryTest.cs` file, and edit the `Example()` method as shown below.

   ```csharp
   protected static void Example()
   {
       // Create a new XML library
       Library2 doc = Library2.CreateDocument();
       // Append the root element
       LibraryType root = doc.Library3.Append();

       // Create the library generation date using Altova DateTime class
       Altova.Types.DateTime dt = new Altova.Types.DateTime(System.DateTime.Now);
       // Append the date to the root
       root.LastUpdated.Value = dt;

       // Add a new book
   }
   ```
// Set the value of the ID attribute
book.ID.Value = 1;
// Set the format of the book (enumeration)
book.Format.EnumerationValue = BookFormatType.EnumValues.eHardcover;
// Set the Title and Author elements
book.Author.Append().Value = "Altova";

// Append a dictionary (book of derived type) and populate its attributes and elements
DictionaryType dictionary = new DictionaryType(root.Book.Append().Node);
dictionary.ID.Value = 2;
dictionary.Title.Append().Value = "English-German Dictionary";
dictionary.Author.Append().Value = "John Doe";
dictionary.FromLang.Append().Value = "English";
dictionary.ToLang.Append().Value = "German";
// Since it's a derived type, make sure to set the xsi:type attribute of the book element
dictionary.SetXsiType();

// Optionally, set the schema location (adjust the path if your schema is not in the same folder as the generated instance file)
doc.SetSchemaLocation("Library.xsd");

// Save the XML document with the "pretty print" option enabled
doc.SaveToFile("GeneratedLibrary.xml", true);
}

3. Press F5 to start debugging. If the code was executed successfully, a GeneratedLibrary.xml file is created in the solution output directory (typically, bin/Debug).

Reading an XML document
2. Save the code below as Library.xml to the output directory of the project (by default, bin/Debug). This is the file that will be read by the program code.

```xml
<?xml version="1.0" encoding="utf-8"?>
  <Book ID="1" Format="E-book">
    <Title>The XMLSpy Handbook</Title>
    <Author>Altova</Author>
  </Book>
  <Book ID="2" Format="Paperback" xmlns:n1="http://www.nanonull.com/"
```
3. In Solution Explorer, open the `LibraryTest.cs` file, and edit the `Example()` method as shown below.

```csharp
protected static void Example()
{
    // Load the XML file into a new Library instance
    Library2 doc = Library2.LoadFromFile("Library.xml");
    // Get the root element
    LibraryType root = doc.Library3.First;

    // Read the library generation date
    Altova.Types.DateTime dt = root.LastUpdated.Value;
    string dt_as_string = dt.ToString(DateTimeFormat.W3_dateTime);
    Console.WriteLine("The library generation date is: " +
    dt_as_string);

    // Iteration: for each <Book>...
    foreach (BookType book in root.Book)
    {
        // Output values of ID attribute and (first and only) title element
        Console.WriteLine("ID:    " + book.ID.Value);
        Console.WriteLine("Title: " + book.Title.First.Value);

        // Read and compare an enumeration value
        if (book.Format.EnumerationValue ==
            BookFormatType.EnumValues.ePaperback)
            Console.WriteLine("This is a paperback book.");

        // Iteration: for each <Author>
        foreach (xs.stringType author in book.Author)
            Console.WriteLine("Author: " + author.Value);

        // Determine if this book is of derived type
        {
            // Find the value of the xsi:type attribute
            string xsiTypeValue =
            // Get the namespace URI and the lookup prefix of this namespace
            string prefix =
                book.Node.GetPrefixOfNamespace(namespaceUri);
```
4. Press F5 to start debugging. If the code was executed successfully, Library.xml will be read by the program code, and its contents displayed as console output.

Reading and writing elements and attributes

Values of attributes and elements can be accessed using the Value property of the generated member element or attribute class, for example:

```csharp
// Output values of ID attribute and (first and only) title element
Console.WriteLine("ID: " + book.ID.Value);
Console.WriteLine("Title: " + book.Title.First.Value);
```

To get the value of the Title element in this particular example, we also used the First() method, since this is the first (and only) Title element of a book. For cases when you need to pick a specific element from a list by index, use the At() method.

The class generated for each member element of a type implements the standard System.Collections(IEnumerable) interface. This makes it possible to loop through multiple elements of the same type. In this particular example, you can loop through all books of a Library object as follows:

```csharp
// Iteration: for each <Book>...
foreach (BookType book in root.Book)
{
    // your code here...
}
```

To add a new element, use the Append() method. For example, the following code appends the root element to the document:
// Append the root element to the library
LibraryType root = doc.Library3.Append();

You can set the value of an attribute (like ID in this example) as follows:

// Set the value of the ID attribute
book.ID.Value = 1;

For further information, see [YourSchemaType].MemberAttribute and 
[YourSchemaType].MemberElement class reference.

Reading and writing enumeration values

If your XML schema defines simple types as enumerations, the enumerated values become
available as Enum values in the generated code. In the schema used in this example, a book
format can be hardcover, paperback, e-book, and so on. Therefore, in the generated code, these
values would be available through an Enum:

To assign enumeration values to an object, use code such as the one below:

// Set the format of the book (enumeration)
book.Format.EnumerationValue = BookFormatType.EnumValues.eHardcover;

You can read such enumeration values from XML instance documents as follows:

// Read and compare an enumeration value
Console.WriteLine("This is a paperback book.");

When an "if" condition is not enough, create a switch to determine each enumeration value and
process it as required.

Working with xs:dateTime and xs:duration types

If the schema from which you generated code uses time and duration types such as
xs:dateTime, or xs:duration, these are converted to Altova native classes in generated code.
Therefore, to write a date or duration value to the XML document, do the following:

1. Construct an Altova.Types.DateTime or Altova.Types.Duration object (either from
System.DateTime, or by using parts such as hours and minutes, see
Altova.Types.DateTime and Altova.Types.Duration for more information).
2. Set the object as value of the required element or attribute, for example:

```csharp
// Create the library generation date using Altova DateTime class
Altova.Types.DateTime dt = new Altova.Types.DateTime(System.DateTime.Now);
// Append the date to the root
root.LastUpdated.Value = dt;
```

To read a date or duration from an XML document, do the following:

1. Declare the element value (or attribute) as Altova.Types.DateTime or
   Altova.Types.Duration object.
2. Format the required element or attribute, for example:

```csharp
// Read the library generation date
Altova.Types.DateTime dt = root.LastUpdated.Value;
string dt_as_string = dt.ToString(DateTimeFormat.W3_dateTime);
Console.WriteLine("The library generation date is: "+ dt_as_string);
```

For more information, see Altova.Types.DateTime and Altova.Types.Duration class reference.

Working with derived types
If your XML schema defines derived types, you can preserve type derivation in XML documents
that you create or load programmatically. Taking the schema used in this example, the following
code listing illustrates how to create a new book of derived type DictionaryType:

```csharp
// Append a dictionary (book of derived type) and populate its attributes and
elements
DictionaryType dictionary = new DictionaryType(root.Book.Append().Node);
dictionary.ID.Value = 2;
dictionary.Title.Append().Value = "English-German Dictionary";
dictionary.Author.Append().Value = "John Doe";
dictionary.FromLanguage.Append().Value = "English";
dictionary.ToLanguage.Append().Value = "German";

// Since it's a derived type, make sure to set the xsi:type attribute of the
book element
dictionary.SetXsiType();
```

Note that it is important to set the xsi:type attribute of the newly created book. This ensures
that the book type will be interpreted correctly by the schema when the XML document is
validated.

When you load data from an XML document, the following code listing shows how to identify a
book of derived type DictionaryType in the loaded XML instance. First, the code finds the value
of the xsi:type attribute of the book node. If the namespace URI of this node is http://
www.nanonull.com/LibrarySample, and if the URI lookup prefix and type matches the value of
the xsi:type attribute, then this is a dictionary:
// Determine if this book is of derived type
{
    // Find the value of the xsi:type attribute
    string xsiTypeValue =
    // Get the namespace URI and the lookup prefix of this namespace
    string prefix = book.Node.GetPrefixOfNamespace(namespaceUri);

    // if this book has DictionaryType
    if (namespaceUri == "http://www.nanonull.com/LibrarySample" &&
    xsiTypeValue.Equals(prefix + ":DictionaryType"))
    {
        // output additional fields
        DictionaryType dictionary = new DictionaryType(book.Node);
        Console.WriteLine("Language from: ",
        dictionary.FromLang.First.Value);
        Console.WriteLine("Language to: ",
        dictionary.ToLang.First.Value);
    }
    else
    {
        throw new Exception("Unexpected book type");
    }
}

16.7.5.4 Reading and Writing XML Documents (Java)

After you generate code from the Library schema (see Example Schema), a test Java project is created, along with several supporting Altova libraries.

About the generated Java libraries
The following diagram illustrates some of the most important classes of the generated code.
The central class of the generated code is the `Library2` class, which represents the XML document. Such a class is generated for every schema and its name depends on the schema file name (`Library.xsd`, in this example). Note that this class is called `Library2` to avoid a possible conflict with the namespace name. As shown in the diagram, this class provides methods for
loading documents from files, binary streams, or strings (or saving documents to files, streams, strings). For a description of this class, see the class reference (com.[YourSchema].[Doc]).

The Library3 member of the Library2 class represents the actual root of the document. Again, the number at the end is meant to avoid a naming conflict with the class name.

According to the code generation rules mentioned in About Generated Java Code, member classes are generated for each attribute and for each element of a type. In the generated code, the name of such member classes is prefixed with MemberAttribute_ and MemberElement_, respectively. In the diagram above, examples of such classes are MemberAttribute_ID and MemberElement_Author, generated from the Author element and ID attribute of a book, respectively. Such classes enable you to manipulate programmatically the corresponding elements and attributes in the instance XML document (for example, append, remove, set value, etc). For more information, see com.[YourSchema].[YourSchemaType].MemberAttribute and com.[YourSchema].[YourSchemaType].MemberElement class reference.

Since the DictionaryType is a complex type derived from BookType in the schema, this relationship is also reflected in the generated classes. As illustrated in the diagram, the class DictionaryType inherits the BookType class.

If your XML schema defines simple types as enumerations, the enumerated values become available as Enum values in the generated code. In the schema used in this example, a book format can be hardcover, paperback, e-book, and so on. Therefore, in the generated code, these values would be available through an Enum that is a member of the BookFormatType class.

Writing an XML document

1. On the File menu of Eclipse, click Import, select Existing Projects into Workspace, and click Next.
2. Next to Select root directory, click Browse, select the directory to which you generated the Java code, and then click Finish.

While prototyping an application from a frequently changing XML schema, you may need to frequently generate code to the same directory, so that the schema changes are immediately reflected in the code. Note that the generated test application and the Altova libraries are overwritten every time when you generate code into the same target directory. Therefore do not add code to the generated test application. Instead, integrate the Altova libraries into your project (see Integrating Schema Wrapper Libraries).

4. Edit the Example() method as shown below.

```
protected static void example() throws Exception {
    // create a new, empty XML document
    Library2 libDoc = Library2.createDocument();

    // create the root element <Library> and add it to the document
    LibraryType lib = libDoc.Library3.append();

    // set the "LastUpdated" attribute
```
com.altova.types.DateTime dt = new com.altova.types.DateTime(DateTime.now());
lib.LastUpdated.setValue(dt);

// create a new <Book> and populate its elements and attributes
BookType book = lib.Book.append();
book.ID.setValue(java.math.BigInteger.valueOf(1));
book.Format.setEnumerationValue(BookFormatType.EPAPERBACK);
book.Title.append().setValue("The XML Spy Handbook");
book.Author.append().setValue("Altova");

// create a dictionary (book of derived type) and populate its elements
and attributes
DictionaryType dict = new DictionaryType(lib.Book.append().getNode());
dict.ID.setValue(java.math.BigInteger.valueOf(2));
dict.Title.append().setValue("English-German Dictionary");
dict.Format.setEnumerationValue(BookFormatType.EE_BOOK);
dict.Author.append().setValue("John Doe");
dict.FromLang.append().setValue("English");
dict.ToLang.append().setValue("German");
dict.setXsiType();

// set the schema location (this is optional)
libDoc.setSchemaLocation("Library.xsd");

// save the XML document to a file with default encoding (UTF-8). "true"
causes the file to be pretty-printed.
libDoc.saveToFile("Library1.xml", true);
}

5. Build the Java project and run it. If the code is executed successfully, a Library1.xml file
is created in the project directory.

Reading an XML document

1. On the File menu of Eclipse, click Import, select Existing Projects into Workspace, and
   click Next.
2. Next to Select root directory, click Browse, select the directory to which you
generated the Java code, and then click Finish.
3. Save the code below as Library1.xml to a local directory (you will need to refer to the
   path of the Library1.xml file from the sample code below).

```xml
<?xml version="1.0" encoding="utf-8"?>
.Library xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmns="http://www.nanonull.com/LibrarySample"
xsi:schemaLocation="http://www.nanonull.com/LibrarySample Library.xsd"
LastUpdated="2016-02-03T17:10:08.4977404">
  <Book ID="1" Format="E-book">
    <Title>The XMLSpy Handbook</Title>
    <Author>Altova</Author>
  </Book>
  <Book ID="2" Format="Paperback" xmlns:n1="http://www.nanonull.com/LibrarySample"
    xsi:type="n1:DictionaryType">
    <Title>English-German Dictionary</Title>
</Library>
```
5. Edit the `Example()` method as shown below.

```java
protected static void example() throws Exception {
    // load XML document from a path, make sure to adjust the path as necessary
    Library2 libDoc = Library2.loadFromFile("Library1.xml");

    // get the first (and only) root element <Library>
    LibraryType lib = libDoc.Library3.first();

    // check whether an element exists:
    if (!lib.Book.exists()) {
        System.out.println("This library is empty.");
        return;
    }

    // read a DateTime schema type
    com.altova.types.DateTime dt = lib.LastUpdated.getValue();
    System.out.println("The library was last updated on: " +
            dt.toDateString());

    // iteration: for each <Book>...
    for (java.util.Iterator itBook = lib.Book.iterator(); itBook.hasNext();)
    {
        BookType book = (BookType) itBook.next();
        // output values of ID attribute and (first and only) title element
        System.out.println("ID: " + book.ID.getValue());
        System.out.println("Title: " + book.Title.first().getValue());

        // read and compare an enumeration value
        if (book.Format.getEnumerationValue() == BookFormatType.EPAPERBACK)
            System.out.println("This is a paperback book.");

        // for each <Author>...
        for (java.util.Iterator itAuthor = book.Author.iterator(); itAuthor.hasNext();)
        {
            System.out.println("Author: " + ((com.Library.xs.stringType)
                    itAuthor.next()).getValue());

        // find the derived type of this book
        // by looking at the value of the xsi:type attribute, using DOM
        if (bookNode.getAttributes().getNamedItem("xsi:type") != null) {
            // Get the value of the xsi:type attribute
            String xsiTypeValue =
```
bookNode.getAttributes().getNamedItem("xsi:type").getNodeValue();

    // Get the namespace URI and lookup prefix of this namespace
    String namespaceUri = bookNode.getNamespaceURI();
    String lookupPrefix = bookNode.lookupPrefix(namespaceUri);

    // If xsi:type matches the namespace URI and type of the book
    if (namespaceUri == "http://www.nanonull.com/LibrarySample"
        && xsiTypeValue.equals(lookupPrefix + ":DictionaryType")) {
        // ...then this is a book of derived type (dictionary)

            DictionaryType dictionary = new DictionaryType(book.getNode());
    // output the value of the "FromLang" and "ToLang" elements
            System.out.println("From language: " +
                dictionary.FromLang.first().getValue());
            System.out.println("To language: " +
                dictionary.ToLang.first().getValue());
    }
    else {
        // throw an error
        throw new java.lang.Error("This book has an unknown type.");
    }
}

6. Build the Java project and run it. If the code is executed successfully, Library1.xml will be read by the program code, and its contents displayed in the Console view.

Reading and writing elements and attributes

Values of attributes and elements can be accessed using the getValue() method of the generated member element or attribute class, for example:

```
// output values of ID attribute and (first and only) title element
System.out.println("ID: " + book.ID.getValue());
System.out.println("Title: " + book.Title.first().getValue());
```

To get the value of the Title element in this particular example, we also used the first() method, since this is the first (and only) Title element of a book. For cases when you need to pick a specific element from a list by index, use the at() method.

To iterate through multiple elements, use either index-based iteration or java.util.Iterator. For example, you can iterate through the books of a library as follows:

```
// index-based iteration
for (int j = 0; j < lib.Book.count(); ++j) {
    // your code here
```
To add a new element, use the `append()` method. For example, the following code appends an empty root `Library` element to the document:

```java
// create the root element <Library> and add it to the document
LibraryType lib = libDoc.Library3.append();
```

Once an element is appended, you can set the value of any of its elements or an attributes by using the `setValue()` method.

```java
// set the value of the Title element
book.Title.append().setValue("The XML Spy Handbook");
// set the value of the ID attribute
book.ID.setValue(java.math.BigInteger.valueOf(1));
```

For further information, see `com.[YourSchema].MemberAttribute` and `com.[YourSchema].MemberElement` class reference.

### Reading and writing enumeration values

If your XML schema defines simple types as enumerations, the enumerated values become available as `Enum` values in the generated code. In the schema used in this example, a book format can be hardcover, paperback, e-book, and so on. Therefore, in the generated code, these values would be available through an `Enum` (see the `BookFormatType` class diagram above). To assign enumeration values to an object, use code such as the one below:

```java
// set an enumeration value
book.Format.setEnumerationValue(BookFormatType.EPAPERBACK);
```

You can read such enumeration values from XML instance documents as follows:

```java
// read an enumeration value
if (book.Format.getEnumerationValue() == BookFormatType.EPAPERBACK) {
    System.out.println("This is a paperback book.");
}
```

When an "if" condition is not enough, create a switch to determine each enumeration value and process it as required.

### Working with xs:dateTime and xs:duration types

If the schema from which you generated code uses time and duration types such as `xs:dateTime`, or `xs:duration`, these are converted to Altova native classes in generated code.
Therefore, to write a date or duration value to the XML document, do the following:

1. Construct a `com.altova.types.DateTime` or `com.altova.types.Duration` object.
2. Set the object as value of the required element or attribute, for example:

   ```java
   // set the value of an attribute of DateTime type
   com.altova.types.DateTime dt = new com.altova.types.DateTime(DateTime.now());
   lib.LastUpdated.setValue(dt);
   ```

To read a date or duration from an XML document:

1. Declare the element value (or attribute) as `com.altova.types.DateTime` or `com.altova.types.Duration` object.
2. Format the required element or attribute, for example:

   ```java
   // read a DateTime type
   com.altova.types.DateTime dt = lib.LastUpdated.getValue();
   System.out.println("The library was last updated on: "+
   dt.toDateString());
   ```

For more information, see `com.altova.types.DateTime` and `com.altova.types.Duration` class reference.

**Working with derived types**

If your XML schema defines derived types, you can preserve type derivation in XML documents that you create or load programmatically. Taking the schema used in this example, the following code listing illustrates how to create a new book of derived type `DictionaryType`:

```java
// create a dictionary (book of derived type) and populate its elements and attributes
DictionaryType dict = new DictionaryType(lib.Book.append().getNode());
dict.ID.setValue(java.math.BigInteger.valueOf(2));
dict.Title.append().setValue("English-German Dictionary");
dict.Format.setEnumerationValue(BookFormatType.EE_BOOK);
dict.Author.append().setValue("John Doe");
dict.FromLang.append().setValue("English");
dict.ToLang.append().setValue("German");
dict.setXsiType();
```

Note that it is important to set the `xsi:type` attribute of the newly created book. This ensures that the book type will be interpreted correctly by the schema when the XML document is validated.

When you load data from an XML document, the following code listing shows how to identify a book of derived type `DictionaryType` in the loaded XML instance. First, the code finds the value of the `xsi:type` attribute of the book node. If the namespace URI of this node is `http://www.nanonull.com/LibrarySample`, and if the URI lookup prefix and type matches the value of the `xsi:type` attribute, then this is a dictionary:

```java
// find the derived type of this book
```
// by looking at the value of the xsi:type attribute, using DOM
if (bookNode.getAttributes().getNamedItem("xsi:type") != null) {
    // Get the value of the xsi:type attribute
    String xsiTypeValue =
        bookNode.getAttributes().getNamedItem("xsi:type").getNodeValue();

    // Get the namespace URI and lookup prefix of the book node
    String namespaceUri = bookNode.getNamespaceURI();
    String lookupPrefix = bookNode.lookupPrefix(namespaceUri);

    // If xsi:type matches the namespace URI and type of the book node
    if (namespaceUri == "http://www.nanonull.com/LibrarySample"
        && xsiTypeValue.equals(lookupPrefix + "DictionaryType")) {
        // ...then this is a book of derived type (dictionary)

        DictionaryType dictionary =
            new DictionaryType(book.getNode());
        // output the value of the "FromLang" and "ToLang" elements
        System.out.println("From language: ");
        dictionary.FromLang.first().getValue();
        System.out.println("To language: ");
        dictionary.ToLang.first().getValue();
    }
    else {
        // throw an error
        throw new java.lang.Error("This book has an unknown type.");
    }
}
16.8 Reference to Generated Classes (C++)

This chapter includes a description of C++ classes generated with MapForce from a DTD or XML schema (see Generating Code from XML Schemas or DTDs). You can integrate these classes into your code to read, modify, and write XML documents.

Note: The generated code may include other supporting classes, which are not listed here and are subject to modification.

16.8.1 altova::DateTime

This class enables you to process XML attributes or elements that have date and time types, such as `xs:dateTime`.

Constructors

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DateTime()</td>
<td>Initializes a new instance of the DateTime class to 12:00:00 midnight, January 1, 0001.</td>
</tr>
<tr>
<td>DateTime(__int64 value, short timezone)</td>
<td>Initializes a new instance of the DateTime class. The value parameter represents the number of ticks (100-nanosecond intervals) that have elapsed since 12:00:00 midnight, January 1, 0001.</td>
</tr>
<tr>
<td>DateTime(int year, unsigned char month, unsigned char day, unsigned char hour, unsigned char minute, double second)</td>
<td>Initializes a new instance of the DateTime class to the year, month, day, hour, minute, and second supplied as argument.</td>
</tr>
<tr>
<td>DateTime(int year, unsigned char month, unsigned char day, unsigned char hour, unsigned char minute, double second, short timezone)</td>
<td>Initializes a new instance of the DateTime class to the year, month, day, hour, minute, second and timezone supplied as argument. The timezone is expressed in minutes and can be positive or negative. For example, the timezone &quot;UTC-01:00&quot; is expressed as &quot;-60&quot;.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char Day() const</td>
<td>Returns the day of month of the current DateTime object. The return values range from 1 through 31.</td>
</tr>
<tr>
<td>int DayOfYear() const</td>
<td>Returns the day of year of the current DateTime object. The return values range from 1 through 366.</td>
</tr>
<tr>
<td>bool HasTimezone() const</td>
<td>Returns Boolean true if the current DateTime object has a timezone.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>timezone defined; false otherwise.</td>
<td></td>
</tr>
<tr>
<td>unsigned char Hour() const</td>
<td>Returns the hour of the current DateTime object. The return values range from 0 through 23.</td>
</tr>
<tr>
<td>static bool IsLeapYear(int year)</td>
<td>Returns Boolean true if the year of the DateTime class is a leap year; false otherwise.</td>
</tr>
<tr>
<td>unsigned char Minute() const</td>
<td>Returns the minute of the current DateTime object. The return values range from 0 through 59.</td>
</tr>
<tr>
<td>unsigned char Month() const</td>
<td>Returns the month of the current DateTime object. The return values range from 1 through 12.</td>
</tr>
<tr>
<td>__int64 NormalizedValue() const</td>
<td>Returns the value of the DateTime object expressed as the Coordinated Universal Time (UTC).</td>
</tr>
<tr>
<td>double Second() const</td>
<td>Returns the second of the current DateTime object. The return values range from 0 through 59.</td>
</tr>
<tr>
<td>void SetTimezone(short tz)</td>
<td>Sets the timezone of the current DateTime object to the timezone value supplied as argument. The tz argument is expressed in minutes and can be positive or negative.</td>
</tr>
<tr>
<td>short Timezone() const</td>
<td>Returns the timezone, in minutes, of the current DateTime object. Before using this method, make sure that the object actually has a timezone, by calling the HasTimezone() method.</td>
</tr>
<tr>
<td>__int64 Value() const</td>
<td>Returns the value of the DateTime object, expressed in the number of ticks (100-nanosecond intervals) that have elapsed since 12:00:00 midnight, January 1, 0001.</td>
</tr>
<tr>
<td>int Weekday() const</td>
<td>Returns the day of week of the current DateTime object, as an integer. Values range from 0 through 6, where 0 is Monday (ISO-8601).</td>
</tr>
<tr>
<td>int Weeknumber() const</td>
<td>Returns the number of week in the year of the current DateTime object. The return values are according to ISO-8601.</td>
</tr>
<tr>
<td>int WeekOfMonth() const</td>
<td>Returns the number of week in the month of the current DateTime object. The return values are according to ISO-8601.</td>
</tr>
<tr>
<td>int Year() const</td>
<td>Returns the year of the current DateTime object.</td>
</tr>
</tbody>
</table>

Example

```cpp
void Example()
{
    // initialize a new DateTime instance to 12:00:00 midnight, January 1st, 0001
```
```cpp
altova::DateTime dt1 = altova::DateTime();

// initialize a new DateTime instance using the year, month, day, hour,
// minute, and second
altova::DateTime dt2 = altova::DateTime(2015, 11, 10, 9, 8, 7);

// initialize a new DateTime instance using the year, month, day, hour,
// minute, second, and UTC +01:00 timezone
altova::DateTime dt = altova::DateTime(2015, 11, 22, 13, 53, 7, 60);

// Get the value of this DateTime object
std::cout << "The number of ticks of the DateTime object is: " <<
dt.Value() << std::endl;

// Get the year
cout << "The year is: " << dt.Year() << endl;
// Get the month
cout << "The month is: " << (int)dt.Month() << endl;
// Get the day of the month
cout << "The day of the month is: " << (int) dt.Day() << endl;
// Get the day of the year
cout << "The day of the year is: " << dt.DayOfYear() << endl;
// Get the hour
cout << "The hour is: " << (int) dt.Hour() << endl;
// Get the minute
cout << "The minute is: " << (int) dt.Minute() << endl;
// Get the second
cout << "The second is: " << dt.Second() << endl;
// Get the weekday
cout << "The weekday is: " << dt.Weekday() << endl;
// Get the week number
cout << "The week of year is: " << dt.Weeknumber() << endl;
// Get the week in month
cout << "The week of month is: " << dt.WeekOfMonth() << endl;

// Check whether a DateTime instance has a timezone
if (dt.HasTimezone() == TRUE)
{
    // output the value of the Timezone
    cout << "The timezone is: " << dt.Timezone() << endl;
}
else
{
    cout << "No timezone has been defined." << endl;
}

// Construct a DateTime object with a timezone UTC+01:00 (Vienna)
altova::DateTime vienna_dt = DateTime(2015, 11, 23, 14, 30, 59, +60);

// Output the result in readable format
cout << "The Vienna time: "
    << (int) vienna_dt.Month()
    << ";" << (int) vienna_dt.Day()
    << "\n" << (int) vienna_dt.Hour()
    << ";" << (int) vienna_dt.Minute()
    << ";" << (int) vienna_dt.Second()
    << endl;
```

```
// Convert the value to UTC time
DateTime utc_dt = DateTime(vienna_dt.NormalizedValue());
// Output the result in readable format
cout << "The UTC time:    "
    << (int) utc_dt.Month()
    << "." << (int) utc_dt.Day()
    << " " << (int) utc_dt.Hour()
    << ":" << (int) utc_dt.Minute()
    << ":" << (int) utc_dt.Second()
    << endl;

// Check if a year is a leap year
int year = 2016;
if( altova::DateTime::IsLeapYear(year) )
{ cout << year << " is a leap year" << endl; }
else
{ cout << year << " is not a leap year" << endl; }
```

### 16.8.2 altova::Duration

This class enables you to process XML attributes or elements of type `xs:duration`.

#### Constructors

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration()</td>
<td>Initializes a new instance of the <code>Duration</code> class to an empty value.</td>
</tr>
<tr>
<td>Duration(const DayTimeDuration&amp; dt)</td>
<td>Initializes a new instance of the <code>Duration</code> class to a duration defined by the <code>dt</code> argument (see <code>altova::DayTimeDuration</code>).</td>
</tr>
<tr>
<td>Duration(const YearMonthDuration&amp; ym)</td>
<td>Initializes a new instance of the <code>Duration</code> class to the duration defined by the <code>ym</code> argument (see <code>altova::YearMonthDuration</code>).</td>
</tr>
<tr>
<td>Duration(const YearMonthDuration&amp; ym, const DayTimeDuration&amp; dt)</td>
<td>Initializes a new instance of the <code>Duration</code> class to the duration defined by both the <code>dt</code> and the <code>ym</code> arguments (see <code>altova::YearMonthDuration</code> and <code>altova::DayTimeDuration</code>).</td>
</tr>
</tbody>
</table>

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int Days() const</td>
<td>Returns the number of days in the current <code>Duration</code> instance.</td>
</tr>
<tr>
<td>DayTimeDuration DayTime() const</td>
<td>Returns the day and time duration in the current <code>Duration</code> instance, expressed as a <code>DayTimeDuration</code> object (see <code>altova::DayTimeDuration</code>).</td>
</tr>
</tbody>
</table>
### Name | Description
---|---
| **int Hours() const** | Returns the number of hours in the current `Duration` instance. |
| **bool IsNegative() const** | Returns Boolean `true` if the current `Duration` instance is negative. |
| **bool IsPositive() const** | Returns Boolean `true` if the current `Duration` instance is positive. |
| **int Minutes() const** | Returns the number of minutes in the current `Duration` instance. |
| **int Months() const** | Returns the number of months in the current `Duration` instance. |
| **double Seconds() const** | Returns the number of seconds in the current `Duration` instance. |
| **YearMonthDuration YearMonth() const** | Returns the year and month duration in the current `Duration` instance, expressed as a `YearMonthDuration` object (see `altova::YearMonthDuration`). |
| **int Years() const** | Returns the number of years in the current `Duration` instance. |

**Example**

The following code listing illustrates creating a new `Duration` object, as well as reading values from it.

```cpp
void ExampleDuration()
{
    // Create an empty Duration object
    altova::Duration empty_duration = altova::Duration();

    // Create a Duration object using an existing duration value
    altova::Duration duration1 = altova::Duration(empty_duration);

    // Create a YearMonth duration of six years and five months
    altova::YearMonthDuration yrduration = altova::YearMonthDuration(6, 5);

    // Create a DayTime duration of four days, three hours, two minutes, and one second
    altova::DayTimeDuration dtduration = altova::DayTimeDuration(4, 3, 2, 1);

    // Create a Duration object by combining the two previously created durations
    altova::Duration duration = altova::Duration(yrduration, dtduration);
}
```
// Get the number of years in this Duration instance
cout << "Years:  " << duration.Years() << endl;

// Get the number of months in this Duration instance
cout << "Months: " << duration.Months() << endl;

// Get the number of days in this Duration instance
cout << "Days:   " << duration.Days() << endl;

// Get the number of hours in this Duration instance
cout << "Hours:  " << duration.Hours() << endl;

// Get the number of hours in this Duration instance
cout << "Minutes: " << duration.Minutes() << endl;

// Get the number of seconds in this Duration instance
cout << "Seconds: " << duration.Seconds() << endl;

}  

16.8.3 altova::DayTimeDuration

This class enables you to process XML schema duration types that consist of a day and time part.

Constructors

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DayTimeDuration()</td>
<td>Initializes a new instance of the DayTimeDuration class to an empty value.</td>
</tr>
<tr>
<td>DayTimeDuration(int days, int hours, int minutes, double seconds)</td>
<td>Initializes a new instance of the DayTimeDuration class to the number of days, hours, minutes, and seconds supplied as arguments.</td>
</tr>
<tr>
<td>explicit DayTimeDuration(__int64 value)</td>
<td>Initializes a new instance of the DayTimeDuration class to a duration that consists of as many ticks (100-nanosecond intervals) as supplied in the value argument.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int Days() const</td>
<td>Returns the number of days in the current DayTimeDuration instance.</td>
</tr>
<tr>
<td>int Hours() const</td>
<td>Returns the number of hours in the current DayTimeDuration instance.</td>
</tr>
<tr>
<td>bool IsNegative() const</td>
<td>Returns Boolean true if the current DayTimeDuration instance is negative.</td>
</tr>
</tbody>
</table>
### altova::YearMonthDuration

This class enables you to process XML schema duration types that consist of a year and month part.

#### Constructors

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YearMonthDuration()</td>
<td>Initializes a new instance of the <code>YearMonthDuration</code> class to an empty value.</td>
</tr>
<tr>
<td>YearMonthDuration(int years, int months)</td>
<td>Initializes a new instance of the <code>YearMonthDuration</code> class to the number of years and months supplied in the <code>years</code> and <code>months</code> arguments.</td>
</tr>
<tr>
<td>explicit YearMonthDuration(int value)</td>
<td>Initializes a new instance of the <code>YearMonthDuration</code> class to a duration that consists of as many ticks (100-nanosecond intervals) as supplied in the <code>value</code> argument.</td>
</tr>
</tbody>
</table>

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool IsNegative() const</td>
<td>Returns Boolean <code>true</code> if the current <code>YearMonthDuration</code> instance is negative.</td>
</tr>
<tr>
<td>bool IsPositive() const</td>
<td>Returns Boolean <code>true</code> if the current <code>YearMonthDuration</code> instance is positive.</td>
</tr>
<tr>
<td>int Months() const</td>
<td>Returns the number of months in the current <code>YearMonthDuration</code> instance.</td>
</tr>
<tr>
<td>int Value() const</td>
<td>Returns the value (in ticks) of the current <code>YearMonthDuration</code> instance.</td>
</tr>
<tr>
<td>int Years()</td>
<td>Returns the number of years in the current <code>YearMonthDuration</code> instance.</td>
</tr>
</tbody>
</table>
### 16.8.5 `altova::meta::Attribute`

This class enables you to access schema information about classes generated from attributes. Note that this class is not meant to provide dynamic information about particular instances of an attribute in an XML document. Instead, it enables you to obtain programmatically information about a particular attribute defined in the XML schema.

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SimpleType GetDataType()</code></td>
<td>Returns the type of the attribute content.</td>
</tr>
<tr>
<td><code>string_type GetLocalName()</code></td>
<td>Returns the local name of the attribute.</td>
</tr>
<tr>
<td><code>string_type GetNamespaceURI()</code></td>
<td>Returns the namespace URI of the attribute.</td>
</tr>
<tr>
<td><code>bool IsRequired()</code></td>
<td>Returns true if the attribute is required.</td>
</tr>
</tbody>
</table>

**Operators**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool operator()</code></td>
<td>Returns true if this is not the NULL Attribute.</td>
</tr>
<tr>
<td><code>bool operator!()</code></td>
<td>Returns true if this is the NULL Attribute.</td>
</tr>
</tbody>
</table>

### 16.8.6 `altova::meta::ComplexType`

This class enables you to access schema information about classes generated from complex types. Note that this class is not meant to provide dynamic information about particular instances of a complex type in an XML document. Instead, it enables you to obtain programmatically information about a particular complex type defined in the XML schema.

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Attribute FindAttribute(const char_type* localName, const char_type* namespaceURI)</code></td>
<td>Finds the attribute with the specified local name and namespace URI.</td>
</tr>
<tr>
<td><code>Element FindElement(const char_type* localName, const char_type* namespaceURI)</code></td>
<td>Finds the element with the specified local name and namespace URI.</td>
</tr>
<tr>
<td><code>std::vector&lt;Attribute&gt; GetAttributes()</code></td>
<td>Returns a list of all attributes.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ComplexType GetBaseType()</td>
<td>Returns the base type of this type.</td>
</tr>
<tr>
<td>SimpleType GetContentType()</td>
<td>Returns the simple type of the content.</td>
</tr>
<tr>
<td>std::vector&lt;Element&gt; GetElements()</td>
<td>Returns a list of all elements.</td>
</tr>
<tr>
<td>string_type GetLocalName()</td>
<td>Returns the local name of the type.</td>
</tr>
<tr>
<td>string_type GetNamespaceURI()</td>
<td>Returns the namespace URI of the type.</td>
</tr>
</tbody>
</table>

### Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool operator()</td>
<td>Returns true if this is not the NULL ComplexType.</td>
</tr>
<tr>
<td>bool operator!()</td>
<td>Returns true if this is the NULL ComplexType.</td>
</tr>
</tbody>
</table>

#### 16.8.7 altova::meta::Element

This class enables you to access information about classes generated from schema elements. Note that this class is not meant to provide dynamic information about particular instances of an element in an XML document. Instead, it enables you to obtain programmatically information about a particular element defined in the XML schema.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComplexType GetDataType()</td>
<td>Returns the type of the element. Note that this is always a complex type even if declared as simple in the original schema. Use GetContentType() of the returned object to get the simple content type.</td>
</tr>
<tr>
<td>string_type GetLocalName()</td>
<td>Returns the local name of the element.</td>
</tr>
<tr>
<td>unsigned int GetMaxOccurs()</td>
<td>Returns the maxOccurs value defined in the schema.</td>
</tr>
<tr>
<td>unsigned int GetMinOccurs()</td>
<td>Returns the minOccurs value defined in the schema.</td>
</tr>
<tr>
<td>string_type GetNsapaceURI()</td>
<td>Returns the namespace URI of the element.</td>
</tr>
</tbody>
</table>

### Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool operator()</td>
<td>Returns true if this is not the NULL Element.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>bool operator!()</td>
<td>Returns true if this is the NULL Element.</td>
</tr>
</tbody>
</table>

### 16.8.8 \texttt{altova::meta::SimpleType}

This class enables you to access schema information about classes generated from simple types. Note that this class is not meant to provide dynamic information about particular instances of simple types in an XML document. Instead, it enables you to obtain programmatically information about a particular simple type defined in the XML schema.

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleType GetBaseType()</td>
<td>Returns the base type of this type.</td>
</tr>
<tr>
<td>std::vector&lt;string_type&gt; GetEnumerations()</td>
<td>Returns a list of all enumeration facets.</td>
</tr>
<tr>
<td>unsigned int GetFractionDigits()</td>
<td>Returns the value of this facet.</td>
</tr>
<tr>
<td>unsigned int GetLength()</td>
<td>Returns the value of this facet.</td>
</tr>
<tr>
<td>string_type GetLocalName()</td>
<td>Returns the local name of the type.</td>
</tr>
<tr>
<td>string_type GetMaxExclusive()</td>
<td>Returns the value of this facet.</td>
</tr>
<tr>
<td>string_type GetMaxInclusive()</td>
<td>Returns the value of this facet.</td>
</tr>
<tr>
<td>unsigned int GetMaxLength()</td>
<td>Returns the value of this facet.</td>
</tr>
<tr>
<td>string_type GetMinExclusive()</td>
<td>Returns the value of this facet.</td>
</tr>
<tr>
<td>string_type GetMinInclusive()</td>
<td>Returns the value of this facet.</td>
</tr>
<tr>
<td>unsigned int GetMinLength()</td>
<td>Returns the value of this facet.</td>
</tr>
<tr>
<td>string_type GetNamespaceURI()</td>
<td>Returns the namespace URI of the type.</td>
</tr>
<tr>
<td>std::vector&lt;string_type&gt; GetPatterns()</td>
<td>Returns a list of all pattern facets.</td>
</tr>
<tr>
<td>unsigned int GetTotalDigits()</td>
<td>Returns the value of this facet.</td>
</tr>
<tr>
<td>WhitespaceType GetWhitespace()</td>
<td>Returns the value of the whitespace facet, which is one of:</td>
</tr>
<tr>
<td></td>
<td>- \texttt{Whitespace_Unknown}</td>
</tr>
<tr>
<td></td>
<td>- \texttt{Whitespace_Preserve}</td>
</tr>
</tbody>
</table>
Name | Description
--- | ---
| |  
- Whitespace.Replace
- Whitespace.Collapse

Operators

Name | Description
--- | ---
bool operator() | Returns true if this is not the NULL SimpleType.
bool operator!() | Returns true if this is the NULL SimpleType.

16.8.9  **[YourSchema]**::**[CDoc]**

When code is generated from an XML Schema, the generated code provides a document class with the same name as the schema. This class contains all possible root elements as members, as well as the following methods. Note that, in the method names below, “CDoc” stands for the name of the generated document class itself.

Methods

Name | Description
--- | ---
static CDoc CreateDocument() | Creates a new, empty XML document. Must be released using DestroyDocument().
void DestroyDocument() | Destroys a document. All references to the document and its nodes are invalidated. This must be called when you finished working with a document.
static CDoc LoadFromBinary(const std::vector<unsigned char>& xml) | Loads an XML document from a byte array.
static CDoc LoadFromString(const string_type& xml) | Loads an XML document from a string.
std::vector<unsigned char> SaveToBinary(bool prettyPrint) | Saves an XML document to a byte array. When set to true, the prettyPrint argument re-formats the XML document for better readability.
std::vector<unsigned char> SaveToBinary(bool prettyPrint, const string_type & encoding) | Saves an XML document to a byte array, with optional "pretty-print" formatting, with the specified encoding.
std::vector<unsigned char> SaveToBinary(bool prettyPrint, const string_type & encoding, bool ) | Saves an XML document to a byte array, with optional "pretty-print" formatting, with the specified encoding. Byte order and Unicode byte-order mark
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void SaveToFile(const string_type &amp; fileName, bool prettyPrint)</td>
<td>Saves an XML document to a file, with optional &quot;pretty-print&quot; formatting.</td>
</tr>
<tr>
<td>void SaveToFile(const string_type &amp; fileName, bool omitXmlDecl)</td>
<td>Saves an XML document to a file. If the omitXmlDecl argument is set to true, the XML declaration will not be written.</td>
</tr>
<tr>
<td>void SaveToFile(const string_type &amp; fileName, bool omitXmlDecl, const string_type &amp; encoding)</td>
<td>Saves an XML document to a file with the specified encoding. If the omitXmlDecl argument is set to true, the XML declaration will not be written.</td>
</tr>
<tr>
<td>void SaveToFile(const string_type &amp; fileName, bool prettyPrint, bool omitXmlDecl, const string_type &amp; encoding, bool bBigEndian, bool bBOM)</td>
<td>Saves an XML document to a file, with optional &quot;pretty-print&quot; formatting, with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.</td>
</tr>
<tr>
<td>void SaveToFile(const string_type &amp; fileName, bool prettyPrint, bool omitXmlDecl, const string_type &amp; encoding, bool bBigEndian, bool bBOM, const string_type &amp; lineend)</td>
<td>Saves an XML document to a file, with optional &quot;pretty-print&quot; formatting, with the specified encoding and the specified line end. Byte order and Unicode byte-order mark can be specified for Unicode encodings. This method is only available if you generated the code for the Xerces3 XML library (see Code Generator Options).</td>
</tr>
<tr>
<td>void SaveToFile(const string_type &amp; fileName, bool prettyPrint, bool omitXmlDecl, const string_type &amp; encoding, const string_type &amp; lineend)</td>
<td>Saves an XML document to a file, with optional &quot;pretty-print&quot; formatting, with the specified encoding and the specified line end. This method is only available if you generated the code for the Xerces3 XML library (see Code Generator Options).</td>
</tr>
<tr>
<td>void SaveToFile(const string_type &amp; fileName, bool prettyPrint, const string_type &amp; encoding)</td>
<td>Saves an XML document to a file, with optional &quot;pretty-print&quot; formatting, with the specified encoding.</td>
</tr>
<tr>
<td>void SaveToFile(const string_type &amp; fileName, bool prettyPrint, const string_type &amp; encoding, bool bBigEndian, bool bBOM)</td>
<td>Saves an XML document to a file, with optional &quot;pretty-print&quot; formatting, with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.</td>
</tr>
<tr>
<td>void SaveToFile(const string_type &amp; fileName, bool prettyPrint, const string_type &amp; encoding, bool bBigEndian, bool bBOM, const string_type &amp; lineend)</td>
<td>Saves an XML document to a file with the specified encoding and the specified line end. Byte order and Unicode byte-order mark can be specified for Unicode encodings. This method is only available if you generated the code for the Xerces3 XML library (see Code Generator Options).</td>
</tr>
</tbody>
</table>
### Name

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saves an XML document to a file, with optional &quot;pretty-print&quot; formatting, with the specified encoding and the specified line end. This method is only available if you generated the code for the Xerces3 XML library (see Code Generator Options).</td>
<td>void SaveToFile(const string_type&amp; fileName, bool prettyPrint, const string_type &amp; encoding, const string_type &amp; lineend)</td>
</tr>
<tr>
<td>Saves an XML document to a string, with optional &quot;pretty-print&quot; formatting.</td>
<td>string_type SaveToString(bool prettyPrint)</td>
</tr>
<tr>
<td>Saves an XML document to a string, with optional &quot;pretty-print&quot; formatting. If the omitXmlDecl argument is set to true, the XML declaration will not be written.</td>
<td>string_type SaveToString(bool prettyPrint, bool omitXmlDecl)</td>
</tr>
<tr>
<td>Adds a DOCTYPE declaration with the specified system ID. A root element must already exist. This method is not supported for MSXML, since it is not possible to add a DOCTYPE declaration to a document in memory.</td>
<td>void SetDTDLocation(const string_type &amp; dtdLocation)</td>
</tr>
<tr>
<td>Adds an xsi:schemaLocation or xsi:noNamespaceSchemaLocation attribute to the root element. A root element must already exist.</td>
<td>void SetSchemaLocation(const string_type &amp; schemaLocation)</td>
</tr>
</tbody>
</table>

### 16.8.10 [YourSchema]::MemberAttribute

When code is generated from an XML schema, a class such as this one is created for each member attribute of a type.

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool exists()</td>
<td>Returns true if the attribute exists.</td>
</tr>
<tr>
<td>int GetEnumerationValue()</td>
<td>Generated for enumeration types only. Returns one of the constants generated for the possible values, or &quot;Invalid&quot; if the value does not match any of the enumerated values in the schema.</td>
</tr>
<tr>
<td>altova::meta::Attribute info()</td>
<td>Returns an object for querying schema information (see altova::meta::Attribute).</td>
</tr>
<tr>
<td>void remove()</td>
<td>Removes the attribute from its parent element.</td>
</tr>
<tr>
<td>void SetEnumerationValue(int)</td>
<td>Generated for enumeration types only. Pass one of the constants generated for the possible values to</td>
</tr>
</tbody>
</table>
When code is generated from an XML schema, a class such as this one is created for each member element of a type. In the descriptions below, "MemberType" stands for the name of the member element itself.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterator&lt;MemberType&gt; all()</td>
<td>Returns an object for iterating instances of the member element.</td>
</tr>
<tr>
<td>MemberType append()</td>
<td>Creates a new element and appends it to its parent.</td>
</tr>
<tr>
<td>unsigned int count()</td>
<td>Returns the count of elements.</td>
</tr>
<tr>
<td>int GetEnumerationValue()</td>
<td>Generated for enumeration types only. Returns one of the constants generated for the possible values, or <strong>Invalid</strong> if the value does not match any of the enumerated values in the schema.</td>
</tr>
<tr>
<td>bool exists()</td>
<td>Returns true if at least one element exists.</td>
</tr>
<tr>
<td>MemberType first()</td>
<td>Returns the first instance of the member element.</td>
</tr>
<tr>
<td>MemberType operator[](unsigned int index)</td>
<td>Returns the member element specified by the index.</td>
</tr>
<tr>
<td>altova::meta::Element info()</td>
<td>Returns an object for querying schema information (see <a href="#">altova::meta::Element</a>).</td>
</tr>
<tr>
<td>MemberType last()</td>
<td>Returns the last instance of the member element.</td>
</tr>
<tr>
<td>void remove()</td>
<td>Deletes all occurrences of the element from its parent.</td>
</tr>
<tr>
<td>void remove(unsigned int index)</td>
<td>Deletes the occurrence of the element specified by the index.</td>
</tr>
<tr>
<td>void SetEnumerationValue(int)</td>
<td>Generated for enumeration types only. Pass one of the constants generated for the possible values to this method to set the value.</td>
</tr>
</tbody>
</table>
16.9 Reference to Generated Classes (C#)

This chapter includes a description of C# classes generated with MapForce from a DTD or XML schema (see Generating Code from XML Schemas or DTDs). You can integrate these classes into your code to read, modify, and write XML documents.

Note: The generated code may include other supporting classes, which are not listed here and are subject to modification.

16.9.1 Altova.Types.DateTime

This class enables you to process XML attributes or elements that have date and time types, such as xs:dateTime.

Constructors

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DateTime(DateTime obj)</code></td>
<td>Initializes a new instance of the <code>DateTime</code> class to the <code>DateTime</code> object supplied as argument.</td>
</tr>
<tr>
<td><code>DateTime(System.DateTime newvalue)</code></td>
<td>Initializes a new instance of the <code>DateTime</code> class to the <code>System.DateTime</code> object supplied as argument.</td>
</tr>
<tr>
<td><code>DateTime(int year, int month, int day, int hour, int minute, double second, int offsetTZ)</code></td>
<td>Initializes a new instance of the <code>DateTime</code> class to the year, month, day, hour, minute, second, and timezone offset supplied as arguments.</td>
</tr>
<tr>
<td><code>DateTime(int year, int month, int day, int hour, int minute, double second)</code></td>
<td>Initializes a new instance of the <code>DateTime</code> class to the year, month, day, hour, minute, and second supplied as arguments.</td>
</tr>
<tr>
<td><code>DateTime(int year, int month, int day)</code></td>
<td>Initializes a new instance of the <code>DateTime</code> class to the year, month and day supplied as arguments.</td>
</tr>
</tbody>
</table>

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool HasTimezone</code></td>
<td>Gets a Boolean value which indicates if the <code>DateTime</code> has a timezone.</td>
</tr>
<tr>
<td><code>static DateTime Now</code></td>
<td>Gets a <code>DateTime</code> object that is set to the current date and time on this computer.</td>
</tr>
<tr>
<td><code>short TimezoneOffset</code></td>
<td>Gets or sets the timezone offset, in minutes, of the <code>DateTime</code> object.</td>
</tr>
</tbody>
</table>
| `System.DateTime Value` | Gets or sets the value of the `DateTime` object as a
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>System.DateTime value.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>int</strong> CompareTo(object obj)</td>
<td>The DateTime class implements the IComparable interface. This method compares the current instance of DateTime to another object and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object. See also <a href="https://msdn.microsoft.com/en-us/library/system.icomparable.compareto(v=vs.110).aspx">https://msdn.microsoft.com/en-us/library/system.icomparable.compareto(v=vs.110).aspx</a></td>
</tr>
<tr>
<td>override <strong>bool</strong> Equals(object obj)</td>
<td>Returns true if the specified object is equal to the current object; false otherwise.</td>
</tr>
<tr>
<td><strong>System.DateTime</strong> GetDateTime(bool correctTZ)</td>
<td>Returns a System.DateTime object from the current Altova.Types.DateTime instance. The correctTZ Boolean argument specifies whether the time of the returned object must be adjusted according to the timezone of the current Altova.Types.DateTime instance.</td>
</tr>
<tr>
<td>override <strong>int</strong> GetHashCode()</td>
<td>Returns the hash code of the current instance.</td>
</tr>
<tr>
<td><strong>int</strong> GetWeekOfMonth()</td>
<td>Returns the number of the week in month as an integer.</td>
</tr>
<tr>
<td><strong>static</strong> DateTime Parse(string s)</td>
<td>Creates a DateTime object from the string supplied as argument. For example, the following sample string values would be converted successfully to a DateTime object: 2015-01-01T23:23:23 2015-01-01 2015-11 23:23:23 An exception is raised if the string cannot be converted to a DateTime object. Note that this method is static and can only be called on the Altova.Types.DateTime class itself, not on an instance of the class.</td>
</tr>
<tr>
<td><strong>static</strong> DateTime Parse(string s, DateFormat format)</td>
<td>Creates a DateTime object from a string, using the format supplied as argument. For the list of possible formats, see Altova.Types.DateFormat. An exception is raised if the string cannot be converted</td>
</tr>
</tbody>
</table>
### Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!=</td>
<td>Determines if DateTime a is not equal to DateTime b.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Determines if DateTime a is less than DateTime b.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Determines if DateTime a is less than or equal to DateTime b.</td>
</tr>
<tr>
<td>==</td>
<td>Determines if DateTime a is equal to DateTime b.</td>
</tr>
<tr>
<td>&gt;</td>
<td>Determines if DateTime a is greater than DateTime b.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Determines if DateTime a is greater than or equal to DateTime b.</td>
</tr>
</tbody>
</table>

### Examples

Before using the following code listings in your program, ensure the Altova types are imported:

```csharp
using Altova.Types;
```

The following code listing illustrates various ways to create DateTime objects:

```csharp
protected static void DateTimeExample1()
{
    // Create a DateTime object from the current system time
    Altova.Types.DateTime dt = new Altova.Types.DateTime(System.DateTime.Now);
    Console.WriteLine("The current time is: " + dt.ToString());

    // Create an Altova DateTime object from parts (no timezone)
    Altova.Types.DateTime dt1 = new Altova.Types.DateTime(2015, 10, 12, 10, 50, 33);
    Console.WriteLine("My custom time is : " + dt1.ToString());
}
```
// Create an Altova DateTime object from parts (with UTC+60 minutes timezone)
Altova.Types.DateTime dt2 = new Altova.Types.DateTime(2015, 10, 12, 10, 50, 33, 60);
Console.WriteLine("My custom time with timezone is: " + dt2.ToString());

// Create an Altova DateTime object by parsing a string
Console.WriteLine("Time created from string: " + dt3.ToString());

// Create an Altova DateTime object by parsing a string formatted as schema date
Altova.Types.DateTime dt4 = Altova.Types.DateTime.Parse("2015-01-01", DateTimeFormat.W3_date);
Console.WriteLine("Time created from string formatted as schema date: " + dt4.ToString());

The following code listing illustrates various ways to format DateTime objects:

```csharp
protected static void DateTimeExample2()
{
    // Create a DateTime object from the current system time
    Altova.Types.DateTime dt = new Altova.Types.DateTime(System.DateTime.Now);

    // Output the unformatted DateTime
    Console.WriteLine("Unformatted time: " + dt.ToString());

    // Output this DateTime formatted using various formats
    Console.WriteLine("S_DateTime: " + dt.ToString(DateTimeFormat.S_DateTime));
    Console.WriteLine("S_Days: " + dt.ToString(DateTimeFormat.S_Days));
    Console.WriteLine("S_Seconds: " + dt.ToString(DateTimeFormat.S_Seconds));
    Console.WriteLine("W3_date: " + dt.ToString(DateTimeFormat.W3_date));
    Console.WriteLine("W3_dateTime: " + dt.ToString(DateTimeFormat.W3_dateTime));
    Console.WriteLine("W3_gDay: " + dt.ToString(DateTimeFormat.W3_gDay));
    Console.WriteLine("W3_gMonth: " + dt.ToString(DateTimeFormat.W3_gMonth));
    Console.WriteLine("W3_gMonthDay: " + dt.ToString(DateTimeFormat.W3_gMonthDay));
    Console.WriteLine("W3_gYear: " + dt.ToString(DateTimeFormat.W3_gYear));
    Console.WriteLine("W3_gYearMonth: " + dt.ToString(DateTimeFormat.W3_gYearMonth));
    Console.WriteLine("W3_time: " + dt.ToString(DateTimeFormat.W3_time));
}```
### 16.9.2 `Altova.Types.DateTimeFormat`

The `DateTimeFormat` enum type has the following constant values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>S_DateTime</code></td>
<td>Formats the value as standard dateTime, with a precision of a ten-millionth of a second, including timezone.</td>
<td>2015-11-12 12:19:03.9019132 +01:00</td>
</tr>
<tr>
<td><code>S_Days</code></td>
<td>Formats the value as number of days elapsed since the UNIX epoch.</td>
<td>735913.6318973451087962962963</td>
</tr>
<tr>
<td><code>S_Seconds</code></td>
<td>Formats the value as number of seconds elapsed since the UNIX epoch, with a precision of a ten-millionth of a second.</td>
<td>63582937678.0769062</td>
</tr>
<tr>
<td><code>W3_date</code></td>
<td>Formats the value as schema date.</td>
<td>2015-11-12</td>
</tr>
<tr>
<td><code>W3_dateTime</code></td>
<td>Formats the value as schema dateTime.</td>
<td>2015-11-12T15:12:14.5194251</td>
</tr>
<tr>
<td><code>W3_gDay</code></td>
<td>Formats the value as schema gDay.</td>
<td>---12 (assuming that the date is 12th of the month)</td>
</tr>
<tr>
<td><code>W3_gMonth</code></td>
<td>Formats the value as schema gMonth.</td>
<td>--11 (assuming that the month is November)</td>
</tr>
<tr>
<td><code>W3_gMonthDay</code></td>
<td>Formats the value as schema gMonthDay.</td>
<td>--11-12 (assuming that the date is 12th of November)</td>
</tr>
<tr>
<td><code>W3_gYear</code></td>
<td>Formats the value as schema gYear.</td>
<td>2015 (assuming that the year is 2015)</td>
</tr>
<tr>
<td><code>W3_gYearMonth</code></td>
<td>Formats the value as schema gYearMonth.</td>
<td>2015-11 (assuming that the year is 2015 and the month is November)</td>
</tr>
<tr>
<td><code>W3_time</code></td>
<td>Formats the value as schema time, with a precision of a ten-millionth of a second.</td>
<td>15:19:07.5582719</td>
</tr>
</tbody>
</table>
### 16.9.3 Altova.Types.Duration

This class enables you to process XML attributes or elements of type `xs:duration`.

#### Constructors

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Duration(Duration obj)</code></td>
<td>Initializes a new instance of the <code>Duration</code> class to the <code>Duration</code> object supplied as argument.</td>
</tr>
<tr>
<td><code>Duration(System.TimeSpan newvalue)</code></td>
<td>Initializes a new instance of the <code>Duration</code> class to the <code>System.TimeSpan</code> object supplied as argument.</td>
</tr>
<tr>
<td><code>Duration(long ticks)</code></td>
<td>Initializes a new instance of the <code>Duration</code> class to the number of ticks supplied as argument.</td>
</tr>
<tr>
<td><code>Duration(int newyears, int newmonths, int days, int hours, int minutes, int seconds, double partseconds, bool bnegative)</code></td>
<td>Initializes a new instance of the <code>Duration</code> class to a duration built from parts supplied as arguments.</td>
</tr>
</tbody>
</table>

#### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int Months</code></td>
<td>Gets or sets the number of months of the current instance of <code>Duration</code>.</td>
</tr>
<tr>
<td><code>System.TimeSpan Value</code></td>
<td>Gets or sets the value (as <code>System.TimeSpan</code>) of the current instance of <code>Duration</code>.</td>
</tr>
<tr>
<td><code>int Years</code></td>
<td>Gets or sets the number of years of the current instance of <code>Duration</code>.</td>
</tr>
</tbody>
</table>

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>override bool Equals(object other)</code></td>
<td>Returns <code>true</code> if the specified object is equal to the current object; <code>false</code> otherwise.</td>
</tr>
<tr>
<td><code>override int GetHashCode()</code></td>
<td>Returns the hash code of the current instance.</td>
</tr>
<tr>
<td><code>bool IsNegative()</code></td>
<td>Returns <code>true</code> if the current instance of <code>Duration</code> represents a negative duration.</td>
</tr>
</tbody>
</table>
### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| static Duration Parse( string s, ParseType pt ) | Returns an Altova.Types.Duration object parsed from the string supplied as argument, using the parse type supplied as argument. Valid parse type values:  
  - DURATION       Parse duration assuming that year, month, day, as well as time duration parts exist.  
  - YEARMONTH      Parse duration assuming that only year and month parts exist.  
  - DAYTIME        Parse duration assuming that only the day and time parts exist.  
  Note that this method is static and can only be called on the class itself, not on an instance of the class. |
| override string ToString()                | Converts the current Duration instance to string. For example, a time span of 3 hours, 4 minutes, and 5 seconds would be converted to "PT3H4M5S". |
| string ToYearMonthString()                | Converts the current Duration instance to string, using the "Year and Month" parse type. |

### Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!=</td>
<td>Determines if Duration a is not equal to Duration b.</td>
</tr>
<tr>
<td>==</td>
<td>Determines if Duration a is equal to Duration b.</td>
</tr>
</tbody>
</table>

### Examples

Before using the following code listings in your program, ensure the Altova types are imported:

```csharp
using Altova.Types;
```

The following code listing illustrates various ways to create Duration objects:

```csharp
protected static void DurationExample1()
{
    // Create a new time span of 3 hours, 4 minutes, and 5 seconds
    System.TimeSpan ts = new TimeSpan(3, 4, 5);
    // Create a Duration from the time span
    Duration dr = new Duration(ts);
    // The output is: PT3H4M5S
    Console.WriteLine("Duration created from TimeSpan: " + dr.ToString());

    // Create a negative Altova.Types.Duration from 6 years, 5 months, 4 days, 3 hours,
    
```
```csharp
// 2 minutes, 1 second, and .33 of a second
Duration dr1 = new Duration(6, 5, 4, 3, 2, 1, .33, true);
// The output is: -P6Y5M4DT3H2M1.33S
Console.WriteLine("Duration created from parts: " + dr1.ToString());

// Create a Duration from a string using the DAYTIME parse type
Duration dr2 = Altova.Types.Duration.Parse("-P4DT3H2M1S",
Duration.ParseType.DAYTIME);
// The output is -P4DT3H2M1S
Console.WriteLine("Duration created from string: " + dr2.ToString());

// Create a duration from ticks
Duration dr3 = new Duration(System.DateTime.UtcNow.Ticks);
// Output the result
Console.WriteLine("Duration created from ticks: " + dr3.ToString());
}

The following code listing illustrates getting values from Duration objects:

```csharp
protected static void DurationExample2()
{
    // Create a negative Altova.Types.Duration from 6 years, 5 months, 4
days, 3 hours,
    // 2 minutes, 1 second, and .33 of a second
    Duration dr = new Duration(6, 5, 4, 3, 2, 1, .33, true);
    // The output is: -P6Y5M4DT3H2M1.33S
    Console.WriteLine("The complete duration is: " + dr.ToString());

    // Get only the year and month part as string
    string dr1 = dr.ToYearMonthString();
    Console.WriteLine("The YEARMONTH part is: " + dr1);

    // Get the number of years in duration
    Console.WriteLine("Years: " + dr.Years);

    // Get the number of months in duration
    Console.WriteLine("Months: " + dr.Months);
}
```

### 16.9.4 Altova.Xml.Meta.Attribute

This class enables you to access schema information about classes generated from attributes. Note that this class is not meant to provide dynamic information about particular instances of an attribute in an XML document. Instead, it enables you to obtain programmatically information about a particular attribute defined in the XML schema.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleType DataType</td>
<td>Returns the type of the attribute content</td>
</tr>
</tbody>
</table>
### 16.9.5 Altova.Xml.Meta.ComplexType

This class enables you to access schema information about classes generated from complex types. Note that this class is not meant to provide dynamic information about particular instances of a complex type in an XML document. Instead, it enables you to obtain programmatically information about a particular complex type defined in the XML schema.

#### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute[] Attributes</td>
<td>Returns a list of all attributes.</td>
</tr>
<tr>
<td>ComplexType BaseType</td>
<td>Returns the base type of this type or null if no base type exists.</td>
</tr>
<tr>
<td>SimpleType ContentType</td>
<td>Returns the simple type of the content.</td>
</tr>
<tr>
<td>Element[] Elements</td>
<td>Returns a list of all elements.</td>
</tr>
<tr>
<td>string LocalName</td>
<td>Returns the local name of the type.</td>
</tr>
<tr>
<td>string NamespaceURI</td>
<td>Returns the namespace URI of the type.</td>
</tr>
<tr>
<td>XmlQualifiedName QualifiedName</td>
<td>Returns the qualified name of this type.</td>
</tr>
</tbody>
</table>

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComplexType BaseType</td>
<td>Returns the base type of this type.</td>
</tr>
<tr>
<td>bool Equals(obj)</td>
<td>Checks if two info objects refer to the same type, based on qualified name comparison. Returns true if the type has the same qualified name.</td>
</tr>
<tr>
<td>Attribute FindAttribute(string localName, string namespaceURI)</td>
<td>Finds the attribute with the specified local name and namespace URI.</td>
</tr>
</tbody>
</table>
### 16.9.6 Altova.Xml.Meta.Element

This class enables you to access information about classes generated from schema elements. Note that this class is not meant to provide dynamic information about particular instances of an element in an XML document. Instead, it enables you to obtain programmatically information about a particular element defined in the XML schema.

#### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComplexType</td>
<td>Returns the type of the element. Note that this is always a complex type even if declared as simple in the original schema. Use the <code>ContentType</code> property of the returned object to get the simple content type.</td>
</tr>
<tr>
<td>string LocalName</td>
<td>Returns the local name of the element.</td>
</tr>
<tr>
<td>int MaxOccurs</td>
<td>Returns the <code>maxOccurs</code> value defined in the schema.</td>
</tr>
<tr>
<td>int MinOccurs</td>
<td>Returns the <code>minOccurs</code> value defined in the schema.</td>
</tr>
<tr>
<td>string NamespaceURI</td>
<td>Returns the namespace URI of the element.</td>
</tr>
<tr>
<td>XmlQualifiedName</td>
<td>Returns the qualified name of the element.</td>
</tr>
</tbody>
</table>

### 16.9.7 Altova.Xml.Meta.SimpleType

This class enables you to access schema information about classes generated from simple types. Note that this class is not meant to provide dynamic information about particular instances of simple types in an XML document. Instead, it enables you to obtain programmatically information about a particular simple type defined in the XML schema.

#### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleType</td>
<td>Returns the base type of this type.</td>
</tr>
<tr>
<td>string[] Enumerations</td>
<td>Returns a list of all enumeration facets.</td>
</tr>
<tr>
<td>int FractionDigits</td>
<td>Returns the value of this facet.</td>
</tr>
</tbody>
</table>
### 16.9.8 [YourSchema].[Doc]

When code is generated from an XML Schema, the generated code provides a document class with the same name as the schema. This class contains all possible root elements as members, as well as the members listed below. Note that, in the method names below, "Doc" stands for the name of the generated document class itself.

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>static</strong> Doc CreateDocument()</td>
<td>Creates a new, empty XML document.</td>
</tr>
<tr>
<td><strong>static</strong> Doc CreateDocument(string encoding)</td>
<td>Creates a new, empty XML document, with encoding of type &quot;encoding&quot;.</td>
</tr>
<tr>
<td><strong>static</strong> Doc LoadFromBinary(byte[] binary)</td>
<td>Loads an XML document from a byte array.</td>
</tr>
<tr>
<td><strong>static</strong> Doc LoadFromFile(string filename)</td>
<td>Loads an XML document from a file.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>static</code> Doc LoadFromString(<code>string xmlstring</code>)</td>
<td>Loads an XML document from a string.</td>
</tr>
<tr>
<td><code>byte</code>[] SaveToBinary(<code>bool prettyPrint</code>)</td>
<td>Saves an XML document to a byte array, with optional &quot;pretty-print&quot; formatting.</td>
</tr>
<tr>
<td><code>byte</code>[] SaveToBinary(<code>bool prettyPrint, string encoding</code>)</td>
<td>Saves an XML document to a byte array, with optional &quot;pretty-print&quot; formatting, with the specified encoding.</td>
</tr>
<tr>
<td><code>byte</code>[] SaveToBinary(<code>bool prettyPrint, string encoding, bool bBigEndian, bool bBOM</code>)</td>
<td>Saves an XML document to a byte array, with optional &quot;pretty-print&quot; formatting, with the specified encoding, byte order, and BOM (Byte Order Mark).</td>
</tr>
<tr>
<td><code>void</code> SaveToFile(<code>string fileName, bool prettyPrint</code>)</td>
<td>Saves an XML document to a file, with optional &quot;pretty-print&quot; formatting.</td>
</tr>
<tr>
<td><code>void</code> SaveToFile(<code>string fileName, bool prettyPrint, bool omitXmlDecl</code>)</td>
<td>Saves an XML document to a file, with optional &quot;pretty-print&quot; formatting. When <code>omitXmlDecl</code> is true, the XML declaration will not be written.</td>
</tr>
<tr>
<td><code>void</code> SaveToFile(<code>string fileName, bool prettyPrint, bool omitXmlDecl, string encoding</code>)</td>
<td>Saves an XML document to a file, with optional &quot;pretty-print&quot; formatting, with the specified encoding. When <code>omitXmlDecl</code> is true, the XML declaration will not be written.</td>
</tr>
<tr>
<td><code>void</code> SaveToFile(<code>string fileName, bool prettyPrint, string encoding, string lineend</code>)</td>
<td>Saves an XML document to a file, with optional &quot;pretty-print&quot; formatting, with the specified encoding, and line ending character(s).</td>
</tr>
<tr>
<td><code>void</code> SaveToFile(<code>string fileName, bool prettyPrint, bool omitXmlDecl, string encoding, string lineend</code>)</td>
<td>Saves an XML document to a file, with optional &quot;pretty-print&quot; formatting, with the specified encoding, and line ending character(s). When <code>omitXmlDecl</code> is true, the XML declaration will not be written.</td>
</tr>
<tr>
<td><code>void</code> SaveToFile(<code>string fileName, bool prettyPrint, bool omitXmlDecl, string encoding, bool bBigEndian, bool bBOM, string lineend</code>)</td>
<td>Saves an XML document to a file, with optional &quot;pretty-print&quot; formatting, with the specified encoding, byte order, BOM (Byte Order Mark), and line ending character(s). When <code>omitXmlDecl</code> is true, the XML declaration will not be written.</td>
</tr>
<tr>
<td><code>void</code> SaveToFileWithLineEnd(<code>string fileName, bool</code>)</td>
<td>Saves an XML document to a file, with optional &quot;pretty-print&quot; formatting, and line ending character(s). When <code>omitXmlDecl</code> is true, the XML declaration will not be written.</td>
</tr>
</tbody>
</table>
## 16.9.9 \[YourSchemaType\].MemberAttribute

When code is generated from an XML schema, a class is created for each member attribute of a type. In the descriptions below, "AttributeType" stands for the type of the member attribute itself.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool Exists()</td>
<td>Returns true if the attribute exists.</td>
</tr>
<tr>
<td>void Remove()</td>
<td>Removes the attribute from its parent element.</td>
</tr>
</tbody>
</table>

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int EnumerationValue</td>
<td>Generated for enumeration types only. Sets or gets the attribute value using one of the constants generated for the possible values. Returns <strong>Invalid</strong> if the value does not match any of the enumerated values in the schema.</td>
</tr>
<tr>
<td>Altova.Xml.Meta.Attribute Info</td>
<td>Returns an object for querying schema information (see [Altova.Xml.Meta.Attribute]).</td>
</tr>
<tr>
<td>AttributeType Value</td>
<td>Sets or gets the attribute value.</td>
</tr>
</tbody>
</table>
16.9.10  [YourSchemaType].MemberElement

When code is generated from an XML schema, a class with the following members is created for each member element of a type. The class implements the standard System.Collections.IEnumerable interface, so it can be used with the foreach statement.

In the descriptions below, "MemberType" stands for the type of the member element itself.

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberType Append()</td>
<td>Creates a new element and appends it to its parent.</td>
</tr>
<tr>
<td>MemberType At(int index)</td>
<td>Returns the member element specified by the index.</td>
</tr>
<tr>
<td>System.Collections.IEnumerable GetEnumerator()</td>
<td>Returns an object for iterating instances of the member element.</td>
</tr>
<tr>
<td>void Remove()</td>
<td>Deletes all occurrences of the element from its parent.</td>
</tr>
<tr>
<td>void RemoveAt(int index)</td>
<td>Deletes the occurrence of the element specified by the index.</td>
</tr>
</tbody>
</table>

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int Count</td>
<td>Returns the count of elements.</td>
</tr>
<tr>
<td>int EnumerationValue</td>
<td>Generated for enumeration types only. Sets or gets the element value using one of the constants generated for the possible values. Returns Invalid if the value does not match any of the enumerated values in the schema.</td>
</tr>
<tr>
<td>bool Exists</td>
<td>Returns true if at least one element exists.</td>
</tr>
<tr>
<td>MemberType First</td>
<td>Returns the first instance of the member element.</td>
</tr>
<tr>
<td>MemberType Last</td>
<td>Returns the last instance of the member element.</td>
</tr>
<tr>
<td>MemberType this[int index]</td>
<td>Returns the member element specified by the</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>index.</td>
<td>index.</td>
</tr>
<tr>
<td>MemberType Value</td>
<td>Sets or gets the element content (only generated if element can have mixed or simple content).</td>
</tr>
</tbody>
</table>
16.10 Reference to Generated Classes (Java)

This chapter includes a description of Java classes generated with MapForce from a DTD or XML schema (see Generating Code from XML Schemas or DTDs). You can integrate these classes into your code to read, modify, and write XML documents.

**Note:** The generated code may include other supporting classes, which are not listed here and are subject to modification.

### 16.10.1 com.altova.types.DateTime

This class enables you to process XML attributes or elements that have date and time types, such as xs:dateTime.

#### Constructors

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public DateTime()</strong></td>
<td>Initializes a new instance of the DateTime class to an empty value.</td>
</tr>
<tr>
<td><strong>public DateTime(DateTime newValue)</strong></td>
<td>Initializes a new instance of the DateTime class to the DateTime value supplied as argument.</td>
</tr>
<tr>
<td><strong>public DateTime(int newyear, int newmonth, int newday, int newhour, int newminute, int newsecond, double newpartsecond, int newoffsetTZ)</strong></td>
<td>Initializes a new instance of the DateTime class to the year, month, day, hour, minute, second, the fractional part of the second, and timezone supplied as arguments. The fractional part of the second newpartsecond must be between 0 and 1. The timezone offset newoffsetTZ can be either positive or negative and is expressed in minutes.</td>
</tr>
<tr>
<td><strong>public DateTime(int newyear, int newmonth, int newday, int newhour, int newminute, int newsecond, double newpartsecond)</strong></td>
<td>Initializes a new instance of the DateTime class to the year, month, day, hour, minute, second, and the fractional part of a second supplied as arguments.</td>
</tr>
<tr>
<td><strong>public DateTime(int newyear, int newmonth, int newday)</strong></td>
<td>Initializes a new instance of the DateTime class to the year, month, and day supplied as arguments.</td>
</tr>
<tr>
<td><strong>public DateTime(Calendar newValue)</strong></td>
<td>Initializes a new instance of the DateTime class to the java.util.Calendar value supplied as argument.</td>
</tr>
</tbody>
</table>
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>static</strong> <code>DateTime now()</code></td>
<td>Returns the current time as a <code>DateTime</code> object.</td>
</tr>
<tr>
<td><strong>static</strong> <code>DateTime parse(String s)</code></td>
<td>Returns a <code>DateTime</code> object parsed from the string value supplied as argument. For example, the following sample string values would be converted successfully to a <code>DateTime</code> object:</td>
</tr>
<tr>
<td><code>2015-11-24T12:54:47.969+01:00</code></td>
<td></td>
</tr>
<tr>
<td><code>2015-11-24T12:54:47</code></td>
<td></td>
</tr>
<tr>
<td><code>2015-11-24</code></td>
<td></td>
</tr>
<tr>
<td><code>int</code> <code>getDay()</code></td>
<td>Returns the day of the current <code>DateTime</code> instance.</td>
</tr>
<tr>
<td><code>int</code> <code>getHour()</code></td>
<td>Returns the hour of the current <code>DateTime</code> instance.</td>
</tr>
<tr>
<td><code>int</code> <code>getMillisecond()</code></td>
<td>Returns the millisecond of the current <code>DateTime</code> instance, as an integer value.</td>
</tr>
<tr>
<td><code>int</code> <code>getMinute()</code></td>
<td>Returns the minute of the current <code>DateTime</code> instance.</td>
</tr>
<tr>
<td><code>int</code> <code>getMonth()</code></td>
<td>Returns the month of the current <code>DateTime</code> instance.</td>
</tr>
<tr>
<td><code>double</code> <code>getPartSecond()</code></td>
<td>Returns the fractional part of the second of the current <code>DateTime</code> instance, as a <code>double</code> value. The return value is greater than zero and smaller than one, for example:</td>
</tr>
<tr>
<td><code>0.313</code></td>
<td></td>
</tr>
<tr>
<td><code>int</code> <code>getSecond()</code></td>
<td>Returns the second of the current <code>DateTime</code> instance.</td>
</tr>
<tr>
<td><code>int</code> <code>getTimezoneOffset()</code></td>
<td>Returns the timezone offset, in minutes, of the current <code>DateTime</code> instance. For example, the timezone &quot;UTC-01:00&quot; would be returned as:</td>
</tr>
<tr>
<td><code>-60</code></td>
<td></td>
</tr>
<tr>
<td><code>Calendar</code> <code>getValue()</code></td>
<td>Returns the current <code>DateTime</code> instance as a <code>java.util.Calendar</code> value.</td>
</tr>
<tr>
<td><code>int</code> <code>getWeekday()</code></td>
<td>Returns the day in week of the current <code>DateTime</code> instance. Values range from 0 through 6, where 0 is Monday (ISO-8601).</td>
</tr>
<tr>
<td><code>int</code> <code>getYear()</code></td>
<td>Returns the year of the current <code>DateTime</code> instance.</td>
</tr>
</tbody>
</table>
| `int` `hasTimezone()` | Returns information about the timezone of the current `DateTime` instance. Possible return values are:

- `CalendarBase.TZ_MISS` |
- `A timezone offset is not defined.` |
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CalendarBase.TZ.UTC</td>
<td>The timezone is UTC.</td>
</tr>
<tr>
<td>CalendarBase.TZ_OFFSET</td>
<td>A timezone offset has been defined.</td>
</tr>
<tr>
<td><code>void</code> setDay( int nDay )</td>
<td>Sets the day of the current DateTime instance to the value supplied as argument.</td>
</tr>
<tr>
<td><code>void</code> setHasTimezone( int nHasTZ )</td>
<td>Sets the timezone information of the current DateTime instance to the value supplied as argument. This method can be used to strip the timezone information or set the timezone to UTC (Coordinated Universal Time). Valid values for the nHasTZ argument: CalendarBase.TZ_MISMATCH Set the timezone offset to undefined. CalendarBase.TZ_UTC Set the timezone to UTC. CalendarBase.TZ_OFFSET If the current object has a timezone offset, leave it unchanged.</td>
</tr>
<tr>
<td><code>void</code> setHour( int nHour )</td>
<td>Sets the hour of the current DateTime instance to the value supplied as argument.</td>
</tr>
<tr>
<td><code>void</code> setMinute( int nMinute )</td>
<td>Sets the minute of the current DateTime instance to the value supplied as argument.</td>
</tr>
<tr>
<td><code>void</code> setMonth( int nMonth )</td>
<td>Sets the month of the current DateTime instance to the value supplied as argument.</td>
</tr>
<tr>
<td><code>void</code> setPartSecond( double nPartSecond )</td>
<td>Sets the fractional part of the second of the current DateTime instance to the value supplied as argument.</td>
</tr>
<tr>
<td><code>void</code> setSecond( int nSecond )</td>
<td>Sets the second of the current DateTime instance to the value supplied as argument.</td>
</tr>
<tr>
<td><code>void</code> setTimezoneOffset( int nOffsetTZ )</td>
<td>Sets the timezone offset of the current DateTime instance to the value supplied as argument. The value nOffsetTZ must be an integer (positive or negative) and must be expressed in minutes.</td>
</tr>
<tr>
<td><code>void</code> setYear( int nYear )</td>
<td>Sets the year of the current DateTime instance to the value supplied as argument.</td>
</tr>
<tr>
<td><code>String</code> toString()</td>
<td>Returns the string representation of the current DateTime instance, for example: 2015-11-24T15:50:56.968+01:00</td>
</tr>
</tbody>
</table>
Examples
Before using the following code listings in your program, ensure the Altova types are imported:

```java
import com.altova.types.*;
```

The following code listing illustrates various ways to create `DateTime` objects:

```java
protected static void DateTimeExample1() {
    // Initialize a new instance of the DateTime class to the current time
    DateTime dt = new DateTime(DateTime.now());
    System.out.println("DateTime created from current date and time: " +
                        dt.toString());

    // Initialize a new instance of the DateTime class by supplying the parts
    DateTime dt1 = new DateTime(2015, 11, 23, 14, 30, 24, .459);
    System.out.println("DateTime from parts (no timezone): " +
                       dt1.toString());

    // Initialize a new instance of the DateTime class by supplying the parts
    DateTime dt2 = new DateTime(2015, 11, 24, 14, 30, 24, .459, -60);
    System.out.println("DateTime from parts (with negative timezone): " +
                       dt2.toString());

    // Initialize a new instance of the DateTime class by parsing a string value
    DateTime dt3 = DateTime.parse("2015-11-24T12:54:47.969 +01:00");
    System.out.println("DateTime parsed from string: " + dt3.toString());
}
```

The following code listing illustrates getting values from `DateTime` objects:

```java
protected static void DateTimeExample2() {
    // Initialize a new instance of the DateTime class to the current time
    DateTime dt = new DateTime(DateTime.now());

    // Output the formatted year, month, and day of this DateTime instance
    String str1 = String.format("Year: %d; Month: %d; Day: %d;",
                                dt.getYear(), dt.getMonth(), dt.getDay());
    System.out.println(str1);

    // Output the formatted hour, minute, and second of this DateTime instance
    String str2 = String.format("Hour: %d; Minute: %d; Second: %d;",
                                dt.getHour(), dt.getMinute(), dt.getSecond());
    System.out.println(str2);
}
```
// Return the timezone (in minutes) of this DateTime instance
System.out.println("Timezone: " + dt.getTimezoneOffset());

// Get the DateTime as a java.util.Calendar value
java.util.Calendar dt_java = dt.getValue();
System.out.println("" + dt_java.toString());

// Return the day of week of this DateTime instance
System.out.println("Weekday: " + dt.getWeekday());

// Check whether the DateTime instance has a timezone defined
switch (dt.hasTimezone())
{
    case CalendarBase.TZ_MISSING:
        System.out.println("No timezone.");
        break;
    case CalendarBase.TZ_UTC:
        System.out.println("The timezone is UTC.");
        break;
    case CalendarBase.TZ_OFFSET:
        System.out.println("This object has a timezone.");
        break;
    default:
        System.out.println("Unable to determine whether a timezone is
defined.");
        break;
}

The following code listing illustrates changing the timezone offset of a DateTime object:

protected static void DateTimeExample3()
{
    // Create a new DateTime object with timezone -0100 UTC
    DateTime dt = new DateTime(2015, 11, 24, 14, 30, 24, .459, -60);
    // Output the value before the change
    System.out.println("Before: " + dt.toString());
    // Change the offset to +0100 UTC
    dt.setTimezoneOffset(60);
    // Output the value after the change
    System.out.println("After: " + dt.toString());
}
16.10.2  com.altova.types.Duration

This class enables you to process XML attributes or elements of type `xs:duration`.

**Constructors**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration(Duration newvalue)</td>
<td>Initializes a new instance of the <code>Duration</code> class to the <code>Duration</code> object supplied as argument.</td>
</tr>
<tr>
<td>Duration(int newyear, int newmonth, int newday, int newhour, int newminute, int newsecond, double newpartsecond, boolean newisnegative)</td>
<td>Initializes a new instance of the <code>Duration</code> class to a duration built from parts supplied as arguments.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static Duration getFromDayTime(int newday, int newhour, int newminute, int newsecond, double newpartsecond)</td>
<td>Returns a <code>Duration</code> object created from the number of days, hours, minutes, seconds, and fractional second parts supplied as argument.</td>
</tr>
<tr>
<td>static Duration getFromYearMonth(int newyear, int newmonth)</td>
<td>Returns a <code>Duration</code> object created from the number of years and months supplied as argument.</td>
</tr>
<tr>
<td>static Duration parse(String s)</td>
<td>Returns a <code>Duration</code> object created from the string supplied as argument. For example, the string <code>-P1Y1M1DT1H1M1.333S</code> can be used to create a negative duration of one year, one month, one day, one hour, one minute, one second, and 0.333 fractional parts of a second. To create a negative duration, append the minus sign (-) to the string.</td>
</tr>
</tbody>
</table>
| static Duration parse(String s, ParseType pt) | Returns a `Duration` object created from the string supplied as argument, using a specific parse format. The parse format can be any of the following:  
  - `ParseType.DAYTIME` Must be used when the string `s` consists of any of the following: days, hours, minutes, seconds, fractional second parts, for example `-P4DT4H4M4.774S`.  
  - `ParseType.DURATION` Must be used when the string `s` consists of any of the following: |
## Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int getDay()</td>
<td>Returns the number of days in the current Duration instance.</td>
</tr>
<tr>
<td>long getDayTimeValue()</td>
<td>Returns the day and time value (in milliseconds) of the current Duration instance. Years and months are ignored.</td>
</tr>
<tr>
<td>int getHour()</td>
<td>Returns the number of hours in the current Duration instance.</td>
</tr>
<tr>
<td>int getMillisecond()</td>
<td>Returns the number of milliseconds in the current Duration instance.</td>
</tr>
<tr>
<td>int getMinute()</td>
<td>Returns the number of minutes in the current Duration instance.</td>
</tr>
<tr>
<td>int getMonth()</td>
<td>Returns the number of months in the current Duration instance.</td>
</tr>
<tr>
<td>double getPartSecond()</td>
<td>Returns the number of fractional second parts in the current Duration instance.</td>
</tr>
<tr>
<td>int getSecond()</td>
<td>Returns the number of seconds in the current Duration instance.</td>
</tr>
<tr>
<td>int getYear()</td>
<td>Returns the number of years in the current Duration instance.</td>
</tr>
<tr>
<td>int getYearMonthValue()</td>
<td>Returns the year and month value (in months) of the current Duration instance. Days, hours, seconds, and milliseconds are ignored.</td>
</tr>
<tr>
<td>boolean isNegative()</td>
<td>Returns Boolean true if the current Duration instance is positive.</td>
</tr>
<tr>
<td>void setDayTimeValue(long l)</td>
<td>Sets the duration to the number of milliseconds supplied as argument, affecting only the day and time part of the duration.</td>
</tr>
<tr>
<td>void setNegative( boolean isNegative )</td>
<td>Converts the current Duration instance to a negative duration.</td>
</tr>
<tr>
<td>void</td>
<td>Sets the duration to the number of months supplied as</td>
</tr>
</tbody>
</table>

### ParseType.YEARMONTH

Must be used when the string s consists of any of the following: years, months. For example: P3Y2M.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setYearMonthValue(int l)</code></td>
<td>argument. Only the years and months part of the duration is affected.</td>
</tr>
<tr>
<td><code>String toString()</code></td>
<td>Returns the string representation of the current <code>Duration</code> instance, for example:</td>
</tr>
<tr>
<td></td>
<td>-P4DT4H4M4.774S</td>
</tr>
<tr>
<td><code>String toYearMonthString()</code></td>
<td>Returns the string representation of the <code>YearMonth</code> part of the current <code>Duration</code> instance, for example:</td>
</tr>
<tr>
<td></td>
<td>P1Y2M</td>
</tr>
</tbody>
</table>

**Examples**

Before using the following code listings in your program, ensure the Altova types are imported:

```java
import com.altova.types.*;
import com.altova.types.Duration.ParseType;
```

The following code listing illustrates various ways to create `Duration` objects:

```java
protected static void ExampleDuration() {
    // Create a negative duration of 1 year, 1 month, 1 day, 1 hour, 1 minute, 1 second,
    // and 0.333 fractional second parts
    Duration dr = new Duration(1, 1, 1, 1, 1, 1, .333, true);

    // Create a duration from an existing Duration object
    Duration dr1 = new Duration(dr);

    // Create a duration of 4 days, 4 hours, 4 minutes, 4 seconds, .774 fractional second parts
    Duration dr2 = Duration.getFromDayTime(4, 4, 4, 4, .774);

    // Create a duration of 3 years and 2 months
    Duration dr3 = Duration.getFromYearMonth(3, 2);

    // Create a duration from a string
    Duration dr4 = Duration.parse("-P4DT4H4M4.774S");

    // Create a duration from a string, using specific parse formats
    Duration dr5 = Duration.parse("-P1Y1M1DT1H1M1.333S", ParseType.DURATION);
    Duration dr6 = Duration.parse("P3Y2M", ParseType.YEARMONTH);
    Duration dr7 = Duration.parse("-P4DT4H4M4.774S", ParseType.DAYTIME);
}
```

The following code listing illustrates getting and setting the value of `Duration` objects:
```java
protected static void DurationExample2() {
    // Create a duration of 1 year, 2 month, 3 days, 4 hours, 5 minutes, 6 seconds,
    // and 333 milliseconds
    Duration dr = new Duration(1, 2, 3, 4, 5, 6, .333, false);
    // Output the number of days in this duration
    System.out.println(dr.getDay());

    // Create a positive duration of one year and 333 milliseconds
    Duration dr1 = new Duration(1, 0, 0, 0, 0, 0, .333, false);
    // Output the day and time value in milliseconds
    System.out.println(dr1.getDayTimeValue());

    // Create a positive duration of 1 year, 1 month, 1 day, 1 hour, 1 minute, 1 second,
    // and 333 milliseconds
    Duration dr2 = new Duration(1, 1, 1, 1, 1, 1, .333, false);
    // Output the year and month value in months
    System.out.println(dr2.getYearMonthValue());

    // Create a positive duration of 1 year and 1 month
    Duration dr3 = new Duration(1, 1, 0, 0, 0, 0, 0, false);
    // Output the value
    System.out.println("The duration is now: " + dr3.toString());
    // Set the DayTime part of duration to 1000 milliseconds
    dr3.setDayTimeValue(1000);
    // Output the value
    System.out.println("The duration is now: " + dr3.toString());
    // Set the YearMonth part of duration to 1 month
    dr3.setYearMonthValue(1);
    // Output the value
    System.out.println("The duration is now: " + dr3.toString());
    // Output the year and month part of the duration
    System.out.println("The YearMonth part of the duration is: " +
                        dr3.toYearMonthString());
}
```

### 16.10.3 com.altova.xml.meta.Attribute

This class enables you to access schema information about classes generated from attributes. Note that this class is not meant to provide dynamic information about particular instances of an attribute in an XML document. Instead, it enables you to obtain programmatically information about a particular attribute defined in the XML schema.

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleType getDataType()</td>
<td>Returns the type of the attribute content.</td>
</tr>
</tbody>
</table>
### 16.10.4 `com.altova.xml.meta.ComplexType`

This class enables you to access schema information about classes generated from complex types. Note that this class is not meant to provide dynamic information about particular instances of a complex type in an XML document. Instead, it enables you to obtain programmatically information about a particular complex type defined in the XML schema.

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Attribute findAttribute(String localName, String namespaceURI)</code></td>
<td>Finds the attribute with the specified local name and namespace URI.</td>
</tr>
<tr>
<td><code>Element findElement(String localName, String namespaceURI)</code></td>
<td>Finds the element with the specified local name and namespace URI.</td>
</tr>
<tr>
<td><code>Attribute[] GetAttributes()</code></td>
<td>Returns a list of all attributes.</td>
</tr>
<tr>
<td><code>ComplexType getBaseType()</code></td>
<td>Returns the base type of this type.</td>
</tr>
<tr>
<td><code>SimpleType getContentType()</code></td>
<td>Returns the simple type of the content.</td>
</tr>
<tr>
<td><code>Element[] GetElements()</code></td>
<td>Returns a list of all elements.</td>
</tr>
<tr>
<td><code>String getLocalName()</code></td>
<td>Returns the local name of the type.</td>
</tr>
<tr>
<td><code>String getNamespaceURI()</code></td>
<td>Returns the namespace URI of the type.</td>
</tr>
</tbody>
</table>

### 16.10.5 `com.altova.xml.meta.Element`

This class enables you to access information about classes generated from schema elements. Note that this class is not meant to provide dynamic information about particular instances of an element in an XML document. Instead, it enables you to obtain programmatically information about a particular element defined in the XML schema.
Reference to Generated Classes (Java)

16.10.6 com.altova.xml.meta.SimpleType

This class enables you to access schema information about classes generated from simple types. Note that this class is not meant to provide dynamic information about particular instances of simple types in an XML document. Instead, it enables you to obtain programmatically information about a particular simple type defined in the XML schema.

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleType getBaseType()</td>
<td>Returns the base type of this type.</td>
</tr>
<tr>
<td>String[] getEnumerations()</td>
<td>Returns an array of all enumeration facets.</td>
</tr>
<tr>
<td>int getFractionDigits()</td>
<td>Returns the value of this facet.</td>
</tr>
<tr>
<td>int getLength()</td>
<td>Returns the value of this facet.</td>
</tr>
<tr>
<td>String getLocalName()</td>
<td>Returns the local name of the type.</td>
</tr>
<tr>
<td>String getMaxExclusive()</td>
<td>Returns the value of this facet.</td>
</tr>
<tr>
<td>String getMaxInclusive()</td>
<td>Returns the value of this facet.</td>
</tr>
<tr>
<td>int getMaxLength()</td>
<td>Returns the value of this facet.</td>
</tr>
<tr>
<td>int getMinExclusive()</td>
<td>Returns the value of this facet.</td>
</tr>
<tr>
<td>String getMinInclusive()</td>
<td>Returns the value of this facet.</td>
</tr>
<tr>
<td>int getMinLength()</td>
<td>Returns the value of this facet.</td>
</tr>
<tr>
<td>String getNamespaceURI()</td>
<td>Returns the namespace URI of the type.</td>
</tr>
</tbody>
</table>
16.10.7 com.[YourSchema].[Doc]

When code is generated from an XML Schema, the generated code provides a document class with the same name as the schema. This class contains all possible root elements as members, as well as the members listed below. Note that, in the method names below, "Doc" stands for the name of the generated document class itself.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static Doc createDocument()</td>
<td>Creates a new, empty XML document.</td>
</tr>
<tr>
<td>static Doc loadFromBinary(byte[] xml)</td>
<td>Loads an XML document from a byte array.</td>
</tr>
<tr>
<td>static Doc loadFromFile(String fileName)</td>
<td>Loads an XML document from a file.</td>
</tr>
<tr>
<td>static Doc loadFromString(String xml)</td>
<td>Loads an XML document from a string.</td>
</tr>
<tr>
<td>byte[] saveToBinary(boolean prettyPrint)</td>
<td>Saves an XML document to a byte array, with optional &quot;pretty-print&quot; formatting.</td>
</tr>
<tr>
<td>byte[] saveToBinary(boolean prettyPrint, String encoding)</td>
<td>Saves an XML document to a byte array, with optional &quot;pretty-print&quot; formatting, with the specified encoding.</td>
</tr>
<tr>
<td>byte[] saveToBinary(boolean prettyPrint)</td>
<td>Saves an XML document to a byte array, with optional &quot;pretty-print&quot; formatting, with the specified encoding.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>prettyPrint, String encoding, <strong>boolean</strong> bigEndian, <strong>boolean</strong> writeBOM)</td>
<td>Byte order and Unicode byte-order mark can be specified for Unicode encodings.</td>
</tr>
<tr>
<td><strong>void</strong> saveToFile(String fileName, <strong>boolean</strong> prettyPrint)</td>
<td>Saves an XML document to a file, with optional &quot;pretty-print&quot; formatting.</td>
</tr>
<tr>
<td><strong>void</strong> saveToFile(String fileName, <strong>boolean</strong> prettyPrint, <strong>boolean</strong> omitXmlDecl)</td>
<td>Saves an XML document to a byte array, with optional &quot;pretty-print&quot; formatting, with UTF-8 encoding. When omitXmlDecl is true, the XML declaration will not be written.</td>
</tr>
<tr>
<td><strong>void</strong> saveToFile(String fileName, <strong>boolean</strong> prettyPrint, <strong>boolean</strong> omitXmlDecl, String encoding)</td>
<td>Saves an XML document to a byte array, with optional &quot;pretty-print&quot; formatting, with the specified encoding. When omitXmlDecl is true, the XML declaration will not be written. Byte order and Unicode byte-order mark can be specified for Unicode encodings.</td>
</tr>
<tr>
<td><strong>void</strong> saveToFile(String fileName, <strong>boolean</strong> prettyPrint, <strong>boolean</strong> omitXmlDecl, String encoding, <strong>boolean</strong> bBigEndian, <strong>boolean</strong> bBOM)</td>
<td>Saves an XML document to a byte array, with optional &quot;pretty-print&quot; formatting, with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.</td>
</tr>
<tr>
<td><strong>void</strong> saveToFile(String fileName, <strong>boolean</strong> prettyPrint, String encoding)</td>
<td>Saves an XML document to a file, with optional &quot;pretty-print&quot; formatting.</td>
</tr>
<tr>
<td><strong>void</strong> saveToFile(String fileName, <strong>boolean</strong> prettyPrint, String encoding, <strong>boolean</strong> bBigEndian, <strong>boolean</strong> bBOM)</td>
<td>Saves an XML document to a byte array, with optional &quot;pretty-print&quot; formatting, with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.</td>
</tr>
<tr>
<td><strong>String</strong> saveToString( <strong>boolean</strong> prettyPrint)</td>
<td>Saves an XML document to a string, with optional &quot;pretty-print&quot; formatting.</td>
</tr>
<tr>
<td><strong>String</strong> saveToString( <strong>boolean</strong> prettyPrint, <strong>boolean</strong> omitXmlDecl)</td>
<td>Saves an XML document to a string, with optional &quot;pretty-print&quot; formatting. When omitXmlDecl is true, the XML declaration will not be written.</td>
</tr>
<tr>
<td><strong>void</strong> setSchemaLocation(String schemaLocation)</td>
<td>Adds an xsi:schemaLocation or xsi:noNamespaceSchemaLocation attribute to the root element. A root element must already exist.</td>
</tr>
</tbody>
</table>
16.10.8 com.[YourSchema].[YourSchemaType].MemberAttribute

When code is generated from an XML schema, a class is created for each member attribute of a type. In the descriptions below, "AttributeType" stands for the type of the member attribute itself.

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean exists()</td>
<td>Returns true if the attribute exists.</td>
</tr>
<tr>
<td>int getEnumerationValue()</td>
<td>Generated for enumeration types only. Returns one of the constants generated for the possible values, or <strong>Invalid</strong> if the value does not match any of the enumerated values in the schema.</td>
</tr>
<tr>
<td>com.altova.xml.meta.Attribute getInfo()</td>
<td>Returns an object for querying schema information (see <a href="#">com.altova.xml.meta.Attribute</a>).</td>
</tr>
<tr>
<td>AttributeType getValue()</td>
<td>Gets the attribute value.</td>
</tr>
<tr>
<td>void remove()</td>
<td>Removes the attribute from its parent element.</td>
</tr>
<tr>
<td>void setEnumerationValue(int)</td>
<td>Generated for enumeration types only. Pass one of the constants generated for the possible values to this method to set the value.</td>
</tr>
<tr>
<td>void setValue(AttributeType value)</td>
<td>Sets the attribute value.</td>
</tr>
</tbody>
</table>

16.10.9 com.[YourSchema].[YourSchemaType].MemberElement

When code is generated from an XML schema, a class with the following members is created for each member element of a type. In the descriptions below, "MemberType" stands for the type of the member element itself.

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberType append()</td>
<td>Creates a new element and appends it to its parent.</td>
</tr>
<tr>
<td>MemberType at(int index)</td>
<td>Returns the instance of the member element at the specified index.</td>
</tr>
<tr>
<td>int count()</td>
<td>Returns the count of elements.</td>
</tr>
<tr>
<td>boolean exists()</td>
<td>Returns true if at least one element exists.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MemberType first()</td>
<td>Returns the first instance of the member element.</td>
</tr>
<tr>
<td><code>int getEnumerationValue()</code></td>
<td>Generated for enumeration types only. Returns one of the constants generated for the possible values, or <code>Invalid</code> if the value does not match any of the enumerated values in the schema.</td>
</tr>
<tr>
<td>MemberType getValue()</td>
<td>Gets the element content (only generated if element can have simple or mixed content).</td>
</tr>
<tr>
<td>java.util.Iterator iterator()</td>
<td>Returns an object for iterating instances of the member element.</td>
</tr>
<tr>
<td>MemberType last()</td>
<td>Returns the last instance of the member element.</td>
</tr>
<tr>
<td>void remove()</td>
<td>Deletes all occurrences of the element from its parent.</td>
</tr>
<tr>
<td>void removeAt(int index)</td>
<td>Deletes the occurrence of the element specified by the index.</td>
</tr>
<tr>
<td>void setEnumerationValue(int index)</td>
<td>Generated for enumeration types only. Pass one of the constants generated for the possible values to this method to set the value.</td>
</tr>
<tr>
<td>void setValue(MemberType value)</td>
<td>Sets the element content (only generated if element can have simple or mixed content).</td>
</tr>
</tbody>
</table>
16.11 Code Generation Tips

Resolving "Out of memory" exceptions during Java compilation

Complex mappings with large schemas can produce a large amount of code, which might cause a java.lang.OutOfMemory exception during compilation using Ant. To rectify this:

- Add the environment variable **ANT_OPTS**, which sets specific Ant options such as the memory to be allocated to the compiler, and set its value as shown below.

![Edit User Variable](image)

- To make sure that the compiler and the generated code run in the same process as Ant, change the **fork** attribute, in `build.xml`, to false.

You may need to customize the values depending on the amount of memory in your machine and the size of the project you are working with. For more details, see your Java VM documentation.

When running the ant jar command, you may get an error message similar to "[...] archive contains more than 65535 entities". To prevent this, it is recommended that you use Ant 1.9 or later, and, in the `build.xml` file, add `zip64mode="as-needed"` to the `<jar>` element.

Reserving method names

When customizing code generation using the supplied SPL files, it might be necessary to reserve names to avoid collisions with other symbols. To do this:

1. Navigate to subdirectory corresponding to the programming language of the `spl` subdirectory of the program installation directory e.g. C:\Program Files\Altova\MapForce2019\spl\java\.
2. Open the `settings.spl` file and insert a new line into the reserve section e.g. reserve "myReservedWord".
3. Regenerate the program code.
16.12 Code Generator Options

To view or change the MapForce settings applicable to code generation:

- On the **Tools** menu, click **Options**, and then click **Generation**.

The available settings are as follows.

| **C++ Settings** | Defines the specific compiler settings for the C++ environment, namely:  
|                  | - The XML library (MSXML, Xerces 3.x)  
|                  | - Whether static or dynamic libraries must be generated  
|                  | - Whether code must be generated with or without MFC support |


| **Wrapper Classes** | Allows you to generate wrapper classes for XML schemas, see [Generating Code from XML Schemas or DTDs](#). These wrapper classes can be used by custom code that includes the code generated by MapForce. |

| **Server Execution File** | These options are applicable when you compile mappings to MapForce Server execution files. They do not affect generation of |
| C#, C++, or Java code. For more information, see [Compiling Mappings to MapForce Server Execution Files](#). |
16.13 SPL (Spy Programming Language)

This section gives an overview of SPL (Spy Programming Language), the code generator's template language.

It is assumed that you have prior programming experience, and are familiar with operators, functions, variables and classes, as well as the basics of object-oriented programming - which is used heavily in SPL.

The templates used by MapForce are supplied in the ...\MapForce\spl folder. You can use these files as an aid to help you in developing your own templates.

How code generator works
Inputs to the code generator are the template files (.spl) and the object model provided by MapForce. The template files contain SPL instructions for creating files, reading information from the object model and performing calculations, interspersed with literal code fragments in the target programming language.

The template file is interpreted by the code generator and outputs .cpp, .java, .cs source code files, project files, or any other type of file depending on the template. The source code can then be compiled into an executable file that accesses XML data described by the schema file.

SPL files have access to a wide variety of information that is collated from the source schemas. Please note that an SPL file is not tied to a specific schema, but allows access to all schemas! Make sure you write your SPL files generically, avoid structures etc. which apply to specific schemas!

Example: Creating a new file in SPL
This is a very basic SPL file. It creates a file named test.cpp, and places the include statement within it. The close command completes the template.

```
[create "test.cpp"]
#include "stdafx.h"
[close]
```

16.13.1 Basic SPL structure

An SPL file contains literal text to output, interspersed with code generator instructions.

Code generator instructions are enclosed in square brackets `[` and `]`. Multiple statements can be included in a bracket pair. Additional statements have to be separated by a new line or a colon `:`

Valid examples are:

```
[$x = 42
```
Adding text to files

Text not enclosed by [ and ], is written directly to the current output file. If there is no current output file, the text is ignored (see Using files how to create an output file).

To output literal square brackets, escape them with a backslash: \[ and \]; to output a backslash use \\.

Comments

Comments inside an instruction block always begin with a ' character, and terminate on the next line, or at a block close character ].

16.13.2 Declarations

The following statements are evaluated while parsing the SPL template file. They are not affected by flow control statements like conditions, loops or subroutines, and are always evaluated exactly once.

These keywords, like all keywords in SPL, are not case sensitive.

Remember that all of these declarations must be inside a block delimited by square brackets.

map ... to ...

map mapname key to value [, key to value ]...

This statement adds information to a map. See below for specific uses.

map schemanativetype schematype to typespec

The specified built-in XML Schema type will be mapped to the specified native type or class, using the specified formatter. This setting applies only to code generation for version 2007r3 and higher. Typespec is a native type or class name, followed by a comma, followed by the formatter class instance.

Example:
map sitemap:navtype "double" to "double,Altova::DoubleFormatter"

map type ... to ...

map type schematype to classname

The specified built-in XML Schema type will be mapped to the specified class. This setting applies only to code generation for version 2007 or lower.

Example:

map type "float" to "CSchemaFloat"

default ... is ...

default setting is value

This statement allows you to affect how class and member names are derived from the XML Schema. Note that the setting names are case sensitive.

Example:

default "InvalidCharReplacement" is "_"

<table>
<thead>
<tr>
<th>Setting name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ValidFirstCharSet</td>
<td>Allowed characters for starting an identifier</td>
</tr>
<tr>
<td>ValidCharSet</td>
<td>Allowed characters for other characters in an identifier</td>
</tr>
<tr>
<td>InvalidCharReplacement</td>
<td>The character that will replace all characters in names that are not in the ValidCharSet</td>
</tr>
<tr>
<td>AnonTypePrefix</td>
<td>Prefix for names of anonymous types*</td>
</tr>
<tr>
<td>AnonTypeSuffix</td>
<td>Suffix for names of anonymous types*</td>
</tr>
<tr>
<td>ClassNamePrefix</td>
<td>Prefix for generated class names</td>
</tr>
<tr>
<td>ClassNameSuffix</td>
<td>Suffix for generated class names</td>
</tr>
<tr>
<td>EnumerationPrefix</td>
<td>Prefix for symbolic constants declared for enumeration values</td>
</tr>
<tr>
<td>EnumerationUpperCase</td>
<td>&quot;on&quot; to convert the enumeration constant names to upper case</td>
</tr>
<tr>
<td>FallbackName</td>
<td>If a name consists only of characters that are not in ValidCharSet, use this one</td>
</tr>
</tbody>
</table>
* Names of anonymous types are built from AnonTypePrefix + element name + AnonTypeSuffix

```
reserve
```

**reserve**  
`reserve word`

Adds the specified word to the list of reserved words. This ensures that it will never be generated as a class or member name.

Example:

```
reserve "while"
```

**include**

includes the specified file as SPL source. This allows you to split your template into multiple files for easier editing and handling.

```
include filename
```

Example:

```
include "Module.cpp"
```

### 16.13.3 Variables

Any non-trivial SPL file will require variables. Some variables are predefined by the code generator, and new variables may be created simply by assigning values to them.

The `$` character is used when **declaring** or **using** a variable, a variable name is always prefixed by `$`. Variable names are **case sensitive**.

Variables types:

- integer - also used as boolean, where 0 is false and everything else is true
- string
- object - provided by MapForce
- iterator - see **foreach** statement

Variable types are declared by first assignment:

```
[$x = 0]
```

x is now an integer.
Strings
String constants are always enclosed in double quotes, like in the example above. \n and \t inside double quotes are interpreted as newline and tab, " is a literal double quote, and \ is a backslash. String constants can also span multiple lines.

String concatenation uses the & character:

```plaintext
[$BasePath = $outputpath & "/" & $JavaPackageDir]
```

Objects
Objects represent the information contained in the XML schemas, database structures, text files and mappings. Objects have properties, which can be accessed using the . operator. It is not possible to create new objects in SPL (they are predefined by the code generator, derived from the input mapping), but it is possible to assign objects to variables.

Example:

```plaintext
class [=$class.Name]
```

This example outputs the word "class", followed by a space and the value of the Name property of the $class object.

16.13.4 Predefined variables

After a Schema file is analyzed by the code generator, the objects in the table below exist in the Template Engine.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$schematype</td>
<td>integer</td>
<td>1 for DTD, 2 for XML Schema</td>
</tr>
<tr>
<td>$TheLibrary</td>
<td>Library</td>
<td>The library derived from the XML Schema or DTD</td>
</tr>
<tr>
<td>$module</td>
<td>string</td>
<td>Name of the source Schema without extension</td>
</tr>
<tr>
<td>$outputpath</td>
<td>string</td>
<td>The output path specified by the user, or the default output path</td>
</tr>
</tbody>
</table>

For C++ generation only:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$domtype</td>
<td>integer</td>
<td>1 for MSXML, 2 for Xerces</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>$libtype</td>
<td>integer</td>
<td>1 for static LIB, 2 for DLL</td>
</tr>
<tr>
<td>$mfc</td>
<td>boolean</td>
<td>True if MFC support is enabled</td>
</tr>
<tr>
<td>$VSVersion</td>
<td>integer</td>
<td>Specifies the Visual Studio version. Valid values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0      No Visual Studio project</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2008   Visual Studio 2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2010   Visual Studio 2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2013   Visual Studio 2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2015   Visual Studio 2015</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2017   Visual Studio 2017</td>
</tr>
</tbody>
</table>

For C# generation only:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$VSVersion</td>
<td>integer</td>
<td>Specifies the Visual Studio version. Valid values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0      No Visual Studio project</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2008   Visual Studio 2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2010   Visual Studio 2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2013   Visual Studio 2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2015   Visual Studio 2015</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2017   Visual Studio 2017</td>
</tr>
</tbody>
</table>

16.13.5 Creating output files

These statements are used to create output files from the code generation. Remember that all of these statements must be inside a block delimited by square brackets.

```plaintext
create
```

`create filename`

creates a new file. The file has to be closed with the `close` statement. All following output is written to the specified file.

Example:
[create $outputpath & "/" & $JavaPackageDir & "/" & $application.Name & ".java"]
package [=JavaPackageName];

public class [=application.Name]Application {
...
}
[close]

**close**
closes the current output file.

**$variable**
writes the value of the specified variable to the current output file.

Example:

```sql
[$x = 20+3]
The result of your calculation is [=x] - so have a nice day!
```

The file output will be:

The result of your calculation is 23 - so have a nice day!

**write**

**write string**
writes the string to the current output file.

Example:

```sql
[write "C" & $name]
```

This can also be written as:

```sql
C[=$name]
```

**filecopy ... to ...**
**filecopy source to target**

copies the source file to the target file, without any interpretation.

Example:

```
filecopy "java/mapforce/mapforce.png" to $outputpath & "/" & $JavaPackageDir & "mapforce.png"
```

### 16.13.6 Operators

Operators in SPL work like in most other programming languages.

List of SPL operators in descending precedence order:

- `. Access object property`
- `( ) Expression grouping`
- `true boolean constant "true"`
- `false boolean constant "false"`
- `& String concatenation`
- `- Sign for negative number`
- `not Logical negation`
- `* Multiply`
- `/ Divide`
- `% Modulo`
- `+ Add`
- `- Subtract`
- `<= Less than or equal`
- `< Less than`
- `>= Greater than or equal`
- `>` Greater than`
- `=` Equal`
- `<>` Not equal`
- `and Logical conjunction (with short circuit evaluation)`
- `or Logical disjunction (with short circuit evaluation)`
- `=` Assignment`

### 16.13.7 Conditions

SPL allows you to use standard "if" statements. The syntax is as follows:
if condition
    statements
else
    statements
endif

or, without else:

if condition
    statements
endif

Note: There are no round brackets enclosing the condition.

As in any other programming language, conditions are constructed with logical and comparison operators.

Example:

[if $namespace.ContainsPublicClasses and $namespace.Prefix <> ""]
    whatever you want ['inserts whatever you want, in the resulting file']
[endif]

Switch

SPL also contains a multiple choice statement.

Syntax:

switch $variable
    case X:
        statements
    case Y:
    case Z:
        statements
    default:
        statements
endswitch

The case labels must be constants or variables.

The switch statement in SPL does not fall through the cases (as in C), so there is no need for a "break" statement.

16.13.8 Collections and foreach

Collections and iterators

A collection contains multiple objects - like a ordinary array. Iterators solve the problem of storing and incrementing array indexes when accessing objects.
Syntax:

```plaintext
foreach iterator in collection
    statements
next
```

Example:

```plaintext
[foreach $class in $classes
    if not $class.IsInternal
        ]
    class [=$class.Name];
[ endif
next]
```

Example 2:

```plaintext
[foreach $i in 1 To 3
    Write "// Step " & $i & "\n"
    ' Do some work
next]
```

In the first line:

- `$classes` is the **global object** of all generated types. It is a collection of single class objects.
- **Foreach** steps through all the items in `$classes`, and executes the code following the instruction, up to the `next` statement, for each of them.

In each iteration, `$class` is assigned to the next class object. You simply work with the class object instead of using, `classes[i]->Name()`, as you would in C++.

All collection iterators have the following additional properties:

- **Index** The current index, starting with 0
- **IsFirst** true if the current object is the first of the collection (index is 0)
- **IsLast** true if the current object is the last of the collection
- **Current** The current object (this is implicit if not specified and can be left out)

Example:

```plaintext
[foreach $enum in $facet.Enumeration
    if not $enum.IsFirst
        ], [
    endif
]"[=$enum.Value][]"]
next]
```
16.13.9 Subroutines

Code generator supports subroutines in the form of procedures or functions.

Features:

- By-value and by-reference passing of values
- Local/global parameters (local within subroutines)
- Local variables
- Recursive invocation (subroutines may call themselves)

16.13.9.1 Subroutine declaration

Subroutines

Syntax example:

```
Sub SimpleSub()
    ... lines of code
EndSub
```

- **Sub** is the keyword that denotes the procedure.
- **SimpleSub** is the name assigned to the subroutine.
- Round **parenthesis** can contain a parameter list.
- The code block of a subroutine starts immediately after the closing parameter parenthesis.
- **EndSub** denotes the end of the code block.

**Note:** Recursive or cascaded subroutine **declaration** is not permitted, i.e. a subroutine may not contain another subroutine.

Parameters

Parameters can also be passed by procedures using the following syntax:

- All parameters must be variables
- Variables must be prefixed by the $ character
- Local variables are defined in a subroutine
- Global variables are declared explicitly, outside of subroutines
- Multiple parameters are separated by the comma character "," within round parentheses
- Parameters can pass values

Parameters - passing values

Parameters can be passed in two ways, by value and by reference, using the keywords **ByVal** and **ByRef** respectively.

Syntax:
ByVal specifies that the parameter is passed by value. Note that most objects can only be passed by reference.

ByRef specifies that the parameter is passed by reference. This is the default if neither ByVal nor ByRef is specified.

**Function return values**

To return a value from a subroutine, use the `return` statement. Such a function can be called from within an expression.

Example:

```plaintext
' define a function
[Sub MakeQualifiedName( ByVal $namespacePrefix, ByVal $localName )
  if $namespacePrefix = ""
    return $localName
  else
    return $namespacePrefix & ":" & $localName
  endif
EndSub]
```

16.13.9.2 **Subroutine invocation**

Use `call` to invoke a subroutine, followed by the procedure name and parameters, if any.

```plaintext
Call SimpleSub()
```

or

```plaintext
Call CompleteSub( "FirstParameter", $ParamByValue, $ParamByRef )
```

**Function invocation**

To invoke a function (any subroutine that contains a `return` statement), simply use its name inside an expression. Do not use the `call` statement to call functions. Example:

```plaintext
$QName = MakeQualifiedName($namespace, "entry")
```
16.13.9.3  Subroutine example

The following example shows subroutine declaration and invocation.

```plaintext
create $outputpath & $module & "output.txt"

' define sub SimpleSub()
Sub SimpleSub()
]SimpleSub() called
[endsub

' execute sub SimpleSub()
Call SimpleSub()

$ParamByValue = "Original Value"
]ParamByValue = [$ParamByValue]
[$ParamByRef = "Original Value"
]ParamByRef = [$ParamByRef]

' define sub CompleteSub()
[Sub CompleteSub( $param, ByVal $paramByValue, ByRef $paramByRef )
]CompleteSub called.
    param = [$param]
    paramByValue = [$paramByValue]
    paramByRef = [$paramByRef]
[$ParamByRef = "Local Variable"
$paramByValue = "new value"
$paramByRef = "new value"
] Set values inside Sub
[$ParamByRef = "Local Variable"
$paramByValue = "new value"
$paramByRef = "new value"
]CompleteSub finished.
[endsub

' run sub CompleteSub()
Call CompleteSub( "FirstParameter", $ParamByValue, $ParamByRef )
]
ParamByValue=[$ParamByValue]
ParamByRef=[$ParamByRef]
[ Close
```

16.13.10 Built in Types

The section describes the properties of the built-in types used in the predefined variables which describe the parsed schema.
16.13.10.1 Library

This object represents the whole library generated from the XML Schema or DTD.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SchemaNamespaces</td>
<td>Namespace collection</td>
<td>Namespaces in this library</td>
</tr>
<tr>
<td>SchemaFilename</td>
<td>string</td>
<td>Name of the XSD or DTD file this library is derived from</td>
</tr>
<tr>
<td>SchemaType</td>
<td>integer</td>
<td>1 for DTD, 2 for XML Schema</td>
</tr>
<tr>
<td>Guid</td>
<td>string</td>
<td>A globally unique ID</td>
</tr>
<tr>
<td>CodeName</td>
<td>string</td>
<td>Generated library name (derived from schema file name)</td>
</tr>
</tbody>
</table>

16.13.10.2 Namespace

One namespace object per XML Schema namespace is generated. Schema components that are not in any namespace are contained in a special namespace object with an empty NamespaceURI. Note that for DTD, namespaces are also derived from attributes whose names begin with "xmlns".

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeName</td>
<td>string</td>
<td>Name for generated code (derived from prefix)</td>
</tr>
<tr>
<td>LocalName</td>
<td>string</td>
<td>Namespace prefix</td>
</tr>
<tr>
<td>NamespaceURI</td>
<td>string</td>
<td>Namespace URI</td>
</tr>
<tr>
<td>Types</td>
<td>Type collection</td>
<td>All types contained in this namespace</td>
</tr>
<tr>
<td>Library</td>
<td>Library</td>
<td>Library containing this namespace</td>
</tr>
</tbody>
</table>

16.13.10.3 Type

This object represents a complex or simple type. It is used to generate a class in the target language. There is one additional type per library that represents the document, which has all possible root elements as members.

Anonymous types have an empty LocalName.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeName</td>
<td>string</td>
<td>Name for generated code (derived from local</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LocalName</td>
<td>string</td>
<td>Original name in the schema</td>
</tr>
<tr>
<td>Namespace</td>
<td>Namespace</td>
<td>Namespace containing this type</td>
</tr>
<tr>
<td>Attributes</td>
<td>Member (collection)</td>
<td>Attributes contained in this type*</td>
</tr>
<tr>
<td>Elements</td>
<td>Member (collection)</td>
<td>Child elements contained in this type</td>
</tr>
<tr>
<td>IsSimpleType</td>
<td>boolean</td>
<td>True for simple types, false for complex types</td>
</tr>
<tr>
<td>IsDerived</td>
<td>boolean</td>
<td>True if this type is derived from another type, which is also represented by a Type object</td>
</tr>
<tr>
<td>IsDerivedByExtension</td>
<td>boolean</td>
<td>True if this type is derived by extension</td>
</tr>
<tr>
<td>IsDerivedByRestriction</td>
<td>boolean</td>
<td>True if this type is derived by restriction</td>
</tr>
<tr>
<td>IsDerivedByUnion</td>
<td>boolean</td>
<td>True if this type is derived by union</td>
</tr>
<tr>
<td>IsDerivedByList</td>
<td>boolean</td>
<td>True if this type is derived by list</td>
</tr>
<tr>
<td>BaseType</td>
<td>Type</td>
<td>The base type of this type (if IsDerived is true)</td>
</tr>
<tr>
<td>IsDocumentRootType</td>
<td>boolean</td>
<td>True if this type represents the document itself</td>
</tr>
<tr>
<td>Library</td>
<td>Library</td>
<td>Library containing this type</td>
</tr>
<tr>
<td>IsFinal</td>
<td>boolean</td>
<td>True if declared as final in the schema</td>
</tr>
<tr>
<td>IsMixed</td>
<td>boolean</td>
<td>True if this type can have mixed content</td>
</tr>
<tr>
<td>IsAbstract</td>
<td>boolean</td>
<td>True if this type is declared as abstract</td>
</tr>
<tr>
<td>IsGlobal</td>
<td>boolean</td>
<td>True if this type is declared globally in the schema</td>
</tr>
<tr>
<td>IsAnonymous</td>
<td>boolean</td>
<td>True if this type is declared locally in an element</td>
</tr>
</tbody>
</table>

For simple types only:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsNativeBound</td>
<td>boolean</td>
<td>True if native type binding exists</td>
</tr>
<tr>
<td>NativeBinding</td>
<td>NativeBinding</td>
<td>Native binding for this type</td>
</tr>
<tr>
<td>Facets</td>
<td>Facets</td>
<td>Facets of this type</td>
</tr>
<tr>
<td>Whitespace</td>
<td>string</td>
<td>Shortcut to the Whitespace facet</td>
</tr>
</tbody>
</table>
* Complex types with text content (these are types with mixed content and complexType with simpleContent) have an additional unnamed attribute member that represents the text content.

16.13.10.4  **Member**

This object represents an attribute or element in the XML Schema. It is used to create class members of types.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeName</td>
<td>string</td>
<td>Name for generated code (derived from local name or parent declaration)</td>
</tr>
<tr>
<td>LocalName</td>
<td>string</td>
<td>Original name in the schema. Empty for the special member representing text content of complex types.</td>
</tr>
<tr>
<td>NamespaceURI</td>
<td>string</td>
<td>The namespace URI of this Element/Attribute within XML instance documents/streams.</td>
</tr>
<tr>
<td>DeclaringType</td>
<td>Type</td>
<td>Type originally declaring the member (equal to ContainingType for non-inherited members)</td>
</tr>
<tr>
<td>ContainingType</td>
<td>Type</td>
<td>Type where this is a member of</td>
</tr>
<tr>
<td>DataType</td>
<td>Type</td>
<td>Data type of this member's content</td>
</tr>
<tr>
<td>Library</td>
<td>Library</td>
<td>Library containing this member's DataType</td>
</tr>
<tr>
<td>IsAttribute</td>
<td>boolean</td>
<td>True for attributes, false for elements</td>
</tr>
<tr>
<td>IsOptional</td>
<td>boolean</td>
<td>True if minOccurs = 0 or optional attribute</td>
</tr>
<tr>
<td>IsRequired</td>
<td>boolean</td>
<td>True if minOccurs &gt; 0 or required attribute</td>
</tr>
<tr>
<td>IsFixed</td>
<td>boolean</td>
<td>True for fixed attributes, value is in Default property</td>
</tr>
<tr>
<td>IsDefault</td>
<td>boolean</td>
<td>True for attributes with default value, value is in Default property</td>
</tr>
<tr>
<td>IsNillable</td>
<td>boolean</td>
<td>True for nillable elements</td>
</tr>
<tr>
<td>IsUseQualified</td>
<td>boolean</td>
<td>True if NamespaceURI is not empty</td>
</tr>
<tr>
<td>MinOccurs</td>
<td>integer</td>
<td>minOccurs, as in schema. 1 for required attributes</td>
</tr>
<tr>
<td>MaxOccurs</td>
<td>integer</td>
<td>maxOccurs, as in schema. 0 for prohibited attributes, -1 for unbounded</td>
</tr>
<tr>
<td>Default</td>
<td>string</td>
<td>Default value</td>
</tr>
</tbody>
</table>
16.13.10.5 **NativeBinding**

This object represents the binding of a simple type to a native type in the target programming language, as specified by the "schemanativetype" map.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ValueType</td>
<td>string</td>
<td>Native type</td>
</tr>
<tr>
<td>ValueHandler</td>
<td>string</td>
<td>Formatter class instance</td>
</tr>
</tbody>
</table>

16.13.10.6 **Facets**

This object represents all facets of a simple type. Inherited facets are merged with the explicitly declared facets. If a Length facet is in effect, MinLength and MaxLength are set to the same value.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeclaringType</td>
<td>Type</td>
<td>Type facets are declared on</td>
</tr>
<tr>
<td>Whitespace</td>
<td>string</td>
<td>&quot;preserve&quot;, &quot;collapse&quot; or &quot;replace&quot;</td>
</tr>
<tr>
<td>MinLength</td>
<td>integer</td>
<td>Facet value</td>
</tr>
<tr>
<td>MaxLength</td>
<td>integer</td>
<td>Facet value</td>
</tr>
<tr>
<td>MinInclusive</td>
<td>integer</td>
<td>Facet value</td>
</tr>
<tr>
<td>MinExclusive</td>
<td>integer</td>
<td>Facet value</td>
</tr>
<tr>
<td>MaxInclusive</td>
<td>integer</td>
<td>Facet value</td>
</tr>
<tr>
<td>MaxExclusive</td>
<td>integer</td>
<td>Facet value</td>
</tr>
<tr>
<td>TotalDigits</td>
<td>integer</td>
<td>Facet value</td>
</tr>
<tr>
<td>FractionDigits</td>
<td>integer</td>
<td>Facet value</td>
</tr>
<tr>
<td>List</td>
<td>Facet collection</td>
<td>All facets as list</td>
</tr>
</tbody>
</table>

**Facet**

This object represents a single facet with its computed value effective for a specific type.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LocalName</td>
<td>string</td>
<td>Facet name</td>
</tr>
<tr>
<td>NamespaceURI</td>
<td>string</td>
<td>Facet namespace</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FacetType</td>
<td>string</td>
<td>one of &quot;normalization&quot;, &quot;lexicalspace&quot;, &quot;valuespace-length&quot;, &quot;valuespace-enum&quot; or &quot;valuespace-range&quot;</td>
</tr>
<tr>
<td>DeclaringType</td>
<td>Type</td>
<td>Type this facet is declared on</td>
</tr>
<tr>
<td>FacetCheckerName</td>
<td>string</td>
<td>Name of facet checker (from schemafacet map)</td>
</tr>
<tr>
<td>FacetValue</td>
<td>string or integer</td>
<td>Actual value of this facet</td>
</tr>
</tbody>
</table>
17 The MapForce API

The COM-based API of MapForce enables clients to access the functionality of MapForce from a custom code or application, and automate a wide range of tasks.

The MapForce COM API follows the common specifications for automation servers as set out by Microsoft. MapForce is automatically registered as a COM server object during installation. Once the COM server object is registered, you can invoke it from within applications and scripting languages that have programming support for COM calls. This makes it possible to access the MapForce API not only from development environments using .NET, C++ and Visual Basic, but also from scripting languages like JScript and VBScript.

**Note:** If you use the MapForce API to create an application that you intend to distribute to other clients, MapForce must be installed on each client computer. Also, your custom integration code must be deployed to (or your application installed on) each client computer.
17.1 Overview

This overview of the MapForce API provides you with the object model for the API and a description of the most important API concepts. The following topics are covered:

- Accessing the API
- The Object Model
- Error Handling
- Examples

17.1.1 Accessing the API

To access the MapForce COM API, a new instance of the `Application` object must be created in your application (or script). Once this object is created, you can interact with MapForce by invoking its methods and properties as required (for example, create a new document, open an existing document, generate mapping code, etc).

Prerequisites

To make the MapForce COM object available in your Visual Studio project, add a reference to the MapForce type library (.tlb) file. The following instructions are applicable to Visual Studio 2013, but are similar in other Visual Studio versions:

1. On the Project menu, click Add Reference.
2. Click Browse and select the `MapForce.tlb` file located in the MapForce installation folder.

A sample MapForce API client in C# is available at: (My) Documents\Altova\MapForce2019\MapForceExamples\API\C#.

In Java, the MapForce API is available through Java-COM bridge libraries. These libraries are available in the MapForce installation folder: C:\Program Files (x86)\Altova\MapForce2019\JavaAPI (note this path is valid when 32-bit MapForce runs on 64-bit Windows, otherwise adjust the path accordingly).

- `AltovaAutomation.dll`: a JNI wrapper for Altova automation servers
- `AltovaAutomation.jar`: Java classes to access Altova automation servers
- `MapForceAPI.jar`: Java classes that wrap the MapForce automation interface
- `MapForceAPI_JavaDoc.zip`: a Javadoc file containing help documentation for the Java API

To allow access to the MapForce automation server directly from Java code, the libraries above must be in the Java classpath.

A sample MapForce API client in Java is available at: (My) Documents\Altova\MapForce2019\MapForceExamples\API\Java.

In scripting languages such as JScript or VBScript, the MapForce COM object is accessible through the Microsoft Windows Script Host (see https://msdn.microsoft.com/en-us/library/9bbdkx3k.aspx). Such scripts can be written with a text editor, and do not need
compilation, since they are executed by the Windows Script Host packaged with Windows. (To check that the Windows Script Host is running, type wscript.exe /? at the command prompt). A sample MapForce API client in JScript is available at: (My) Documents\Altova\MapForce2019\MapForceExamples\API\JScript.

**Note:** For 32-bit MapForce, the registered name, or programmatic identifier (ProgId) of the COM object is MapForce.Application. For 64-bit MapForce, the name is MapForce_x64.Application.

### Guidelines

The following guidelines should be considered in your client code:

- Do not hold references to objects in memory longer than you need them. If a user interacts between two calls of your client, then there is no guarantee that these references are still valid.
- Be aware that if your client code crashes, instances of MapForce may still remain in the system. For details on how to avoid error messages, see [Error handling](#).
- Free references explicitly, if using languages such as C++.

### Creating the Application object

The syntax to create the starting Application object depends on the programming language, as shown in the examples below:

#### Visual Basic

```vbnet
dim objMapForce as MapForceLib.Application = new MapForceLib.Application
```

#### VBA

```vba'
create a new instance of MapForce.
dim objMapForce as application
set objMapForce = createobject("MapForce.Application")
```

#### VBScript

```vbs
' Access a running instance, or create a new instance of MapForce.
set objMapForce = getobject("MapForce.Application");
```

#### C#

```csharp
// Create a new instance of MapForce via its automation interface.
MapForceLib.Application objMapForce = new MapForceLib.Application();
```
17.1.2 The Object Model

The starting point for every application which uses the MapForce API is the `Application` object. All other interfaces are accessed through the `Application` object as the starting point.

The object model of the MapForce API can be represented as follows (each indentation level indicates a child–parent relationship with the level directly above):

```
Application
  Options
  Project
    ProjectItem
  Documents
    Document
      MapForceView
      Mapping
        Component
        Datapoint
        Components
        Connection
      Mappings
      ErrorMarkers
        ErrorMarker
      AppOutputLines
        AppOutputLine
        AppOutputLines
          ...
        AppOutputLineSymbol
```

For information about creating an instance the `Application` object, see Accessing the API. For reference to the objects exposed by the API, see Object Reference.
17.1.3 Error Handling

The MapForce API returns errors in two different ways. Every API method returns an HRESULT. This return value informs the caller about any malfunctions during the execution of the method. If the call was successful, the return value is equal to S_OK. C/C++ programmers generally use HRESULT to detect errors.

Visual Basic, scripting languages, and other high-level development environments do not give the programmer access to the returning HRESULT of a COM call. They use the second error-raising mechanism supported by the MapForce API, the IErrorInfo interface. If an error occurs, the API creates a new object that implements the IErrorInfo interface. The development environment takes this interface and fills its own error-handling mechanism with the provided information.

The following text describes how to deal with errors raised from the MapForce API in different development environments.

Visual Basic
A common way to handle errors in Visual Basic is to define an error handler. This error handler can be set with the On Error statement. Usually the handler displays an error message and does some cleanup to avoid spare references and any kind of resource leaks. Visual Basic fills its own Err object with the information from the IErrorInfo interface.

```vba
Sub Validate()
    'place variable declarations here

    'set error handler
    On Error GoTo ErrorHandler

    'if generation fails, program execution continues at ErrorHandler:
    objMapForce.ActiveDocument.GenerateXSLT()

    'additional code comes here

    'exit
    Exit Sub
End Sub

ErrorHandler:
    MsgBox("Error: " & (Err.Number - vbObjectError) & Chr(13) & "Description: " & Err.Description)
End Sub
```

JavaScript
The Microsoft implementation of JavaScript (JScript) provides a try-catch mechanism to deal with errors raised from COM calls. It is very similar to the VisualBasic approach, in that you also declare an error object containing the necessary information.
function Generate()
{
    // please insert variable declarations here
    try
    {
        objMapForce.ActiveDocument.GenerateXSLT();
    }
    catch (Error)
    {
        sError = Error.description;
        nErrorCode = Error.number & 0xffff;
        return false;
    }

    return true;
}

C/C++
C/C++ gives you easy access to the HRESULT of the COM call and to the IErrorInterface.

HRESULT hr;

    // Call GenerateXSLT() from the MapForce API
    if (FAILED(hr = ipDocument->GenerateXSLT()))
    {
        IErrorInfo *ipErrorInfo = Null;

        if (SUCCEEDED(::GetErrorInfo(0, &ipErrorInfo)))
        {
            BSTR bstrDescr;
            ipErrorInfo->GetDescription(&bstrDescr);

            // handle Error information
            wprintf(L"Error message:\t%s\n", bstrDescr);
            ::SysFreeString(bstrDescr);

            // release Error info
            ipErrorInfo->Release();
        }
    }

17.1.4 Examples

Programming languages differ in the way they support COM access. The following examples for C#, Java, and JScript will help you get started. The code listings in this section are available at C:/Users/<username>/Documents/Altova/MapForce2019/MapForceExamples/API.
17.1.4.1  Example C# Project

After you install MapForce, an example MapForce API client project for C# is available in the directory `C:/Users/<username>/Documents/Altova/MapForce2019/MapForceExamples/API`.

You can compile and run the project with Visual Studio 2008 or later. To compile and run the example, open the solution .sln file in Visual Studio and run **Debug | Start Debugging** or **Debug | Start Without Debugging**.

**Note:** If you have a 64-bit operating system and are using a 32-bit installation of MapForce, add the `x86` platform in the solution's Configuration Manager and build the sample using this configuration. A new x86 platform (for the active solution in Visual Studio) can be created in the New Solution Platform dialog (**Build | Configuration Manager | Active solution platform | <New...>**).

When you run the example, a Windows form is displayed, containing buttons that invoke basic MapForce operations:

- Start MapForce
- Create a new mapping design
- Open the CompletePO.mfd file from the `...\MapForceExamples` folder (note that you may need to adjust the path to point to the `\MapForceExamples` folder on your machine)
- Generate C# code in a temp directory
- Shut down MapForce

![Form1](image)

**Code listing**

The listing is commented for ease of understanding. The code essentially consists of a series of handlers for the buttons in the user interface shown above.

```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
```
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // An instance of MapForce accessed via its automation interface.
        MapForceLib.Application MapForce;

        // Location of examples installed with MapForce
        String strExamplesFolder;

        private void Form1_Load(object sender, EventArgs e)
        {
        }

        // handler for the "Start MapForce" button
        private void StartMapForce_Click(object sender, EventArgs e)
        {
            if (MapForce == null)
            {
                Cursor.Current = Cursors.WaitCursor;

                // if we have no MapForce instance, we create one and make it visible.
                MapForce = new MapForceLib.Application();
                MapForce.Visible = true;

                // locate examples installed with MapForce.
                int majorVersionYear = MapForce.MajorVersion + 1998;
                strExamplesFolder = Environment.GetEnvironmentVariable("USERPROFILE") + "\My Documents\Altova\MapForce" + Convert.ToString(majorVersionYear) + "\MapForceExamples\";

                Cursor.Current = Cursors.Default;
            }
            else
            {
                // if we have already an MapForce instance running we toggle its visibility flag.
            }
        }

        // handler for the "Open CompletePO.mfd" button
        private void openCompletePO_Click(object sender, EventArgs e)
        {
            if (MapForce == null)
                StartMapForce_Click(null, null);

            // Open one of the sample files installed with the product.
            MapForce.OpenDocument(strExamplesFolder + "CompletePO.mfd");
private void newMapping_Click(object sender, EventArgs e)
{
    if (MapForce == null)
    {
        StartMapForce_Click(null, null);

        // Create a new mapping
        MapForce.NewMapping();
    }
}

private void shutdownMapForce_Click(object sender, EventArgs e)
{
    if (MapForce != null)
    {
        // allow shut-down of MapForce by releasing UI
        MapForce.Visible = false;

        // explicitly release COM object
        try
        {
            while (System.Runtime.InteropServices.Marshal.ReleaseComObject(MapForce) > 0) ;
        }
        finally
        {
            // avoid later access to this object.
            MapForce = null;
        }
    }
}

private void generateCppCode_Click(object sender, EventArgs e)
{
    if (MapForce == null)
    {
        listBoxMessages.Items.Add("start MapForce first.");
        // COM errors get returned to C# as exceptions. We use a try/catch block to handle them.
        try
        {

            listBoxMessages.Items.Add("Active document " + doc.Name);
            doc.GenerateCHashCode();
        }
        catch (Exception ex)
        {
            // The COM call was not successful.
            // Probably no application instance has been started or no document is open.
            MessageBox.Show("COM error: " + ex.Message);
        }
    }
}
delegate void addListBoxItem_delegate(string sText);
// called from the UI thread
private void addListBoxItem(string sText)
{
    listBoxMessages.Items.Add(sText);
}
// wrapper method to allow to call UI controls methods from a worker thread
void syncWithUIthread(Control ctrl, addListBoxItem_delegate methodToInvoke, String sText)
{
    // Control.Invoke: Executes on the UI thread, but calling thread
    // waits for completion before continuing.
    // Control.BeginInvoke: Executes on the UI thread, and calling
    // thread doesn't wait for completion.
    if (ctrl.InvokeRequired)
        ctrl.BeginInvoke(methodToInvoke, new Object[] { sText });
}
// event handler for OnDocumentOpened event
{
    String sText = "";
    if (i_ipDocument.Name.Length > 0)
        sText = "Document " + i_ipDocument.Name + " was opened!";
    else
        sText = "A new mapping was created.";
    // we need to synchronize the calling thread with the UI thread
    // because
    // the COM events are triggered from a working thread
    addListBoxItem_delegate methodToInvoke = new addListBoxItem_delegate(addListBoxItem);
    // call syncWithUIthread with the following arguments:
    // 1 - listBoxMessages - list box control to display messages from
    COM events
    // 2 - methodToInvoke - a C# delegate which points to the method
    which will be called from the UI thread
    // 3 - sText - the text to be displayed in the list box
    syncWithUIthread(listBoxMessages, methodToInvoke, sText);
}
private void checkBoxEventOnOff_CheckedChanged(object sender, EventArgs e)
{
    if (MapForce != null)
    {
        if (checkBoxEventOnOff.Checked)
        else
    }
}
17.1.4.2  Example Java Project

After you install MapForce, an example MapForce API client project for Java is available in the directory \C:/Users/<username>/Documents/Altova/MapForce2019/MapForceExamples/API.

You can test the Java example directly from the command line, using the batch file BuildAndRun.bat, or you can compile and run the example project from within Eclipse. See below for instructions on how to use these procedures.

File list
The Java examples folder contains all the files required to run the example project. These files are listed below:

- AltovaAutomation.dll: Java-COM bridge: DLL part
- AltovaAutomation.jar: Java-COM bridge: Java library part
- MapForceAPI.jar: Java classes of the MapForce API
- RunMapForce.java: Java example source code
- BuildAndRun.bat: Batch file to compile and run example code from the command line prompt. Expects folder where Java Virtual Machine resides as parameter.
- .classpath: Eclipse project helper file
- .project: Eclipse project file
- MapForceAPI_Javadoc.zip: Javadoc file containing help documentation for the Java API

What the example does
The example starts up MapForce and performs a few operations, including opening and closing documents. When done, MapForce stays open. You must close it manually.

Running the example from the command line
To run the example from the command line, open a command prompt window, go to the Java folder of the API Examples folder (see above for location), and then type:

    buildAndRun.bat "<Path-to-the-Java-bin-folder>"

The Java binary folder must be that of a Java Development Kit (JDK) 7 or later installation on your computer.

Press the Return key. The Java source in RunMapForce.java will be compiled and then executed.

Loading the example in Eclipse
Open Eclipse and use the File | Import... | General | Existing Projects into Workspace command to add the Eclipse project file (.project) located in the Java folder of the API Examples folder (see above for location). The project RunMapForce will then appear in your Package Explorer or Navigator.
Select the project and then the command **Run as Java Application** to execute the example.

**Note:** You can select a class name or method of the Java API and press F1 to get help for that class or method.

**Java source code listing**

The Java source code in the example file `RunMapForce.java` is listed below with comments.

```java
// access general JAVA-COM bridge classes
import java.util.Iterator;

import com.altova.automation.libs.*;

// access XMLSpy Java-COM bridge
import com.altova.automation.MapForce.*;

/**
 * A simple example that starts XMLSpy COM server and performs a few operations on it.
 * Feel free to extend.
 */
public class RunMapForce
{
    public static void main(String[] args)
    {
        // an instance of the application.
        Application mapforce = null;

        // instead of COM error handling use Java exception mechanism.
        try
        {
            // Start MapForce as COM server.
            mapforce = new Application();
            // COM servers start up invisible so we make it visible
            mapforce.setVisible(true);

            // The following lines attach to the application events using a default implementation
            // for the events and override one of its methods.
            // If you want to override all document events it is better to derive your listener class
            // from DocumentEvents and implement all methods of this interface.
            mapforce.addListener(new ApplicationEventsDefaultHandler()
            {
                @Override
                public void onDocumentOpened(Document i_ipDoc) throws AutomationException
                {
                    String name = i_ipDoc.getName();
                    if (name.length() > 0)
                    {
                        // do something
                    }
                }
            });

            // do something
        }
        catch (Exception e)
        {
            // handle exception
        }
    }
}
```
System.out.println("Document " + name + " was opened.");
else
    System.out.println("A new mapping was created.");
}
})

// Locate samples installed with the product.
int majorVersionYear = mapforce.getMajorVersion() + 1998;
String strExamplesFolder = System.getenv("USERPROFILE") + "\Documents\Altova\MapForce" + Integer.toString(majorVersionYear) + "\MapForceExamples\";
// create a new MapForce mapping and generate c++ code
Document newDoc = mapforce.newMapping();
ErrorMarkers err1 = newDoc.generateCodeEx(ENUMProgrammingLanguage.eCpp);
display(err1);
// open CompletePO.mfd and generate c++ code
Document doc = mapforce.openDocument(strExamplesFolder + "CompletePO.mfd");  
ErrorMarkers err2 = doc.generateCodeEx(ENUMProgrammingLanguage.eCpp);
display(err2);
doc.close();
doc = null;
System.out.println("Watch MapForce!");
}
catch (AutomationException e)
{
    // e.printStackTrace();
}
finally
{
    // Make sure that MapForce can shut down properly.
    if (mapforce != null)
        mapforce.dispose();

    // Since the COM server was made visible and still is visible, it will keep running
    // and needs to be closed manually.
    System.out.println("Now close MapForce!");
}

public static void display(ErrorMarkers err) throws AutomationException
{
    Iterator<ErrorMarker> itr = err.iterator();

    if (err.getCount() == 0)
        System.out.println("Code generation completed successfully.\n");

    while (itr.hasNext())
    {
        String sError = "";
        Object element = itr.next();
### JScript Examples

After you install MapForce, an example MapForce API client project for JScript is available in the directory `C:/Users/<username>/Documents/Altova/MapForce2019/MapForceExamples/API`.

The example files can be run in one of two ways:

- **From the command line:**
  Open a command prompt window and type the name of one of the example scripts (for example, `Start.js`). The Windows Scripting Host that is packaged with Windows will execute the script.

- **From Windows Explorer:**
  In Windows Explorer, browse for the JScript file and double-click it. The Windows Scripting Host that is packaged with Windows will execute the script. After the script is executed, the command console gets closed automatically.

The following example files are included:

- **Start.js**
  Start Mapforce registered as an automation server or connect to a running instance (see [Start Application](#)).

- **DocumentAccess.js**
  Shows how to open, iterate and close documents (see [Simple Document Access](#)).

- **GenerateCode.js**
  Shows how to invoke code generation using JScript (see [Generate Code](#)).

- **Readme.txt**
  Provides basic help to run the scripts.

This documentation additionally includes a few extra JScript code listings:

- [Example: Code Generation](#)
- [Example: Mapping Execution](#)
- [Example: Project Support](#)

#### 17.1.4.3.1 Start Application

The JScript code listing below starts the application and shuts it down. If an instance of the application is already running, the running instance will be returned.

**Note:** For 32-bit MapForce, the registered name, or programmatic identifier (ProgId) of the COM
The code listed above is available as a sample file (see JScript Examples). To run the script, start it from a command prompt window or from Windows Explorer.

17.1.4.3.2 Simple Document Access

The JScript listing below shows how to open documents, set a document as the active document, iterate through the open documents, and close documents.
```javascript
if (typeof objMapForce) == "undefined")
{
    try {
obMapForce = WScript.GetObject("MapForce_x64.Application")
    } catch (err)
    {
        WScript.Echo("Can't access or create MapForce.Application");
        WScript.Quit();
    }
}

// if newly started, the application will start without its UI visible. Set it
// to visible.
objMapForce.Visible = true;

// *********************** code snippet for "Simple Document Access"
**************************

// Locate examples via USERPROFILE shell variable. The path needs to be
// adapted to major release versions.
objWshShell = WScript.CreateObject("WScript.Shell");
majorVersionYear = objMapForce.MajorVersion + 1998
strExampleFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\Documents\Altova\MapForce" + majorVersionYear + "\MapForceExamples\"


// *********************** code snippet for "Simple Document Access"
**************************

// *********************** code snippet for "Iteration"
***************************

// go through all open documents using a JScript Enumerator
for (var iterDocs = new Enumerator(objMapForce.Documents); !iterDocs.atEnd(); iterDocs.moveNext())
{
    objName = iterDocs.item().Name;
    WScript.Echo("Document name: " + objName);
}

// go through all open documents using index-based access to the document
collection
for (i = objMapForce.Documents.Count; i > 0; i--)
    objMapForce.Documents.Item(i).Close();

// *********************** code snippet for "Iteration"
***************************

//objMapForce.Visible = false; // will shutdown application if it has no
more COM connections
```

The MapForce API

Overview 1509

© 2019 Altova GmbH
Altova MapForce 2019 Enterprise Edition
The code listed above is available as a sample file (see JScript Examples). To run the script, start it from a command prompt window or from Windows Explorer.

17.1.4.3.3 Generate Code

The JScript listing below shows how to open documents, set a document as the active document, iterate through the open documents, and generate C++ code.
The code listed above is available as a sample file (see JScript Examples). To run the script, start it from a command prompt window or from Windows Explorer.

## 17.1.4.3.4 Example: Code Generation

The following JScript example shows how to load an existing document and generate different kinds of mapping code for it.

```javascript
// ------------------- begin JScript example ---------------------
// Generate Code for existing mapping.
```
// works with Windows scripting host.

// ------------------ helper function ------------------
function Exit(strErrorText)
{
    WScript.Echo(strErrorText);
    WScript.Quit(-1);
}

function ERROR(strText, objErr)
{
    if (objErr != null)
    {
        Exit("ERROR: (" + (objErr.number & 0xffff) + ")" + objErr.description + " - " + strText);
    }
    else
    {
        Exit("ERROR: " + strText);
    }
}

// ----------------------------------
// ----------------- MAIN ------------------

// ----- create the Shell and FileSystemObject of the windows scripting
try
{
    objWshShell = WScript.CreateObject("WScript.Shell");
    objFSO = WScript.CreateObject("Scripting.FileSystemObject");
}
catch(err)
{
    Exit("Can't create WScript.Shell object");
}

// ----- open MapForce or access running instance and make it visible
try
{
    objMapForce.Visible = true; // remove this line to perform background processing
}
catch(err)
{
    WScript.Echo("Can't access or create MapForce.Application");
}

// ----- open an existing mapping. adapt this to your needs!
objMapForce.OpenDocument(objFSO.GetAbsolutePathName("Test.mfd");

// ----- access the mapping to have access to the code generation methods
var objDoc = objMapForce.ActiveDocument;

// ----- set the code generation output properties and call the code generation methods.
// ----- adapt the output directories to your needs
try
{
    // ----- code generation uses some of these options
    var objOptions = objMapForce.Options;

    // ----- generate XSLT -----
    objOptions.XSLTDefaultOutputDirectory = "C:\test\TestCOMServer\XSLT";
The following JScript example shows how to load an existing document with a simple mapping, access its components, set input- and output-instance file names and execute the mapping.

```javascript
/*
This sample file performs the following operations:

Load existing MapForce mapping document.
Find source and target component.
Set input and output instance filenames.
Execute the transformation.

Works with Windows scripting host.
*/

// ---- general helpers ------------------------------

function Exit( message )
{
    WScript.Echo( message );
    WScript.Quit(-1);
}

function ERROR( message, err )
{
    if( err != null )
        Exit( "ERROR: (" + (err.number & 0xffff) + ") " + err.description + " -
```
" + message );
    else
        Exit( "ERROR: " + message );
}

// ---- MapForce constants -----------------------
var eComponentUsageKind_Unknown     = 0;
var eComponentUsageKind_Instance    = 1;
var eComponentUsageKind_Input       = 2;
var eComponentUsageKind_Output      = 3;

// ---- MapForce helpers -----------------------

// Searches in the specified mapping for a component by name and returns it. // If not found, throws an error.
function FindComponent( mapping, component_name )
{
    var components = mapping.Components;
    for( var i = 0 ; i < components.Count ; ++i )
    {
        var component = components.Item( i + 1 );
        if( component.Name == component_name )
            return component;
    }
    throw new Error( "Cannot find component with name " + component_name );
}

// Browses components in a mapping and returns the first one found acting as // source component (i.e. having connections on its right side).
function GetFirstSourceComponent( mapping )
{
    var components = mapping.Components;
    for( var i = 0 ; i < components.Count ; ++i )
    {
        var component = components.Item( i + 1 );
        if( component.UsageKind == eComponentUsageKind_Instance &&
            component.HasOutgoingConnections )
        {
            return component;
        }
    }
    throw new Error( "Cannot find a source component" );
}

// Browses components in a mapping and returns the first one found acting as // target component (i.e. having connections on its left side).
function GetFirstTargetComponent( mapping )
```javascript
{
    var components = mapping.Components;
    for ( var i = 0 ; i < components.Count ; ++i )
    {
        var component = components.Item( i + 1 );
        if ( component.UsageKind == eComponentUsageKind_Instance && component.HasIncomingConnections )
        {
            return component;
        }
    }
    throw new Error( "Cannot find a target component" );
}

function IndentTextLines( s )
{
    return "\t" + s.replace( /\n/g, "\n\t" );
}

function GetAppoutputLineFullText( oAppoutputLine )
{
    var s = oAppoutputLine.GetLineText();
    var oAppoutputChildLines = oAppoutputLine.ChildLines;
    var i;
    for ( i = 0 ; i < oAppoutputChildLines.Count ; ++i )
    {
        oAppoutputChildLine = oAppoutputChildLines.Item( i + 1 );
        sChilds = GetAppoutputLineFullText( oAppoutputChildLine );
        s += "\n" + IndentTextLines( sChilds );
    }
    return s;
}

// Create a nicely formatted string from AppOutputLines
function GetResultMessagesString( oAppoutputLines )
{
    var sl = "Transformation result messages:\n";
    var oAppoutputLine;
    var i;
    for ( i = 0 ; i < oAppoutputLines.Count ; ++i )
    {
        oAppoutputLine = oAppoutputLines.Item( i + 1 );
        sl += GetAppoutputLineFullText( oAppoutputLine );
        sl += "\n";
    }
    return( sl );
}
```
// ---- MAIN -------------------------------------

var wshShell;
var fso;
var mapforce;

// create the Shell and FileSystemObject of the windows scripting system
try
{
    wshShell = WScript.CreateObject( "WScript.Shell" );
    fso = WScript.CreateObject( "Scripting.FileSystemObject" );
}
catch( err )
{
    ERROR( "Can't create windows scripting objects", err );
}

// open MapForce or access currently running instance
try
{
    mapforce = WScript.GetObject( "", "MapForce.Application" );
}
catch( err )
{
    ERROR( "Can't access or create MapForce.Application", err );
}

try
{
    // Make MapForce UI visible. This is an API requirement for output
generation.
    mapforce.Visible = true;

    // open an existing mapping.
    // **** adjust the examples path to your needs ! **************
    var sMapForceExamplesPath = fso.BuildPath(
        wshShell.SpecialFolders( "MyDocuments" ),
        "Altova\MapForce2019\MapForceExamples" );
    var sDocFilename = fso.BuildPath( sMapForceExamplesPath,
        "PersonList.mfd" );
    var doc = mapforce.OpenDocument( sDocFilename );

    // Find existing components by name in the main mapping.
    // Note, the names of components may not be unique as a schema component's
    // name
    // is derived from its schema file name.
    var source_component = FindComponent( doc.MainMapping, "Employees" );
    var target_component = FindComponent( doc.MainMapping, "PersonList" );
    // If you do not know the names of the components for some reason, you
could
    // use the following functions instead of FindComponent.
    //var source_component = GetFirstSourceComponent( doc.MainMapping );
    //var target_component = GetFirstTargetComponent( doc.MainMapping );

    // specify the desired input and output files.
    source_component.InputInstanceFile = fso.BuildPath( sMapForceExamplesPath,
        "Employees.xml" );
target_component.OutputInstanceFile =
fso.BuildPath( sMapForceExamplesPath, "test_transformation_results.xml" );

// Perform the transformation.
// You can use doc.GenerateOutput() if you do not need result messages.
// If you have a mapping with more than one target component and you want
// to execute the transformation only for one specific target component,
// call target_component.GenerateOutput() instead.
var result_messages = doc.GenerateOutputEx();

var summary_info =
    "Transformation performed from " + source_component.InputInstanceFile + 
    " to " + target_component.OutputInstanceFile + 
    GetResultMessagesString( result_messages );
WScript.Echo( summary_info );
}
catch( err )
{
    ERROR( "Failure", err );
}

17.1.4.3.6 Example: Project Support

The following JScript example shows how to use the MapForce API to automate tasks pertaining
to MapForce projects. Before running the example, make sure to edit the variable strSamplePath
so that it points to the MapForceExamples folder of your MapForce installation.

To successfully run all operations in this example below, you will need the Enterprise version of
MapForce. If you have the Professional version running, comment out the lines that insert the
WebService project.

// //////////// global variables /////////////////
var objMapForce = null;
var objWshShell = null;
var objFSO = null;

// !!! adapt the following path to your needs. !!!
var strSamplePath = "C:\Users\<username>\Documents\Altova\MapForce2019\MapForceExamples\Tutorial\";

// ///////////////// Helpers /////////////////

function Exit(strErrorText)
{
    WScript.Echo(strErrorText);
    WScript.Quit(-1);
}

function ERROR(strText, objErr)
{
    if (objErr != null)
function CreateGlobalObjects ()
{
    // the Shell and FileSystemObject of the windows scripting host often
    // always useful
    try
    {
        objWshShell = WScript.CreateObject("WScript.Shell");
        objFSO = WScript.CreateObject("Scripting.FileSystemObject");
    }
    catch(err)
    { Exit("Can't create WScript.Shell object"); }

    // create the MapForce connection
    // if there is a running instance of MapForce (that never had a connection)
    // use it
    // otherwise, we automatically create a new instance
    try
    {
    }
    catch(err)
    {
        { Exit("Can't access or create MapForce.Application"); } } }

// ----------------------------------------------------------
// print project tree items and their properties recursively.
// ----------------------------------------------------------
function PrintProjectTree( objProjectItemIter, strTab )
{
    while ( ! objProjectItemIter.atEnd() )
    {
        // get current project item
        objItem = objProjectItemIter.item();

        try
        {
            // ----- print common properties
            strGlobalText += strTab + "[" + objItem.Kind + "]" + objItem.Name + "\n";

            // ----- print code generation properties, if available
            try
            {
                if ( objItem.CodeGenSettings_UseDefault )
                    strGlobalText += strTab + " Use default code generation settings\n";
                else
                    strGlobalText += strTab + " Use default code generation settings\n";
            }
            catch(err)
            { Exit("Can't access or create MapForce.Application"); } }
    }
}

Exit ("ERROR: (" + (objErr.number & 0xffff) + ")" + objErr.description + " - " + strText);
else
    Exit ("ERROR: " + strText); }

strGlobalText += strTab + " code generation language is " + objItem.CodeGenSettings_Language + " output folder is " + objItem.CodeGenSettings_OutputFolder + "\n"
}
catch( err ) {} 

// ----- print WSDL settings, if available
try
{
strGlobalText += strTab + " WSDL File is " + objItem.WSDLFile + " Qualified Name is " + objItem.QualifiedName + "\n";
}
catch( err ) {}

catch( ex )
{ strGlobalText += strTab + "[" + objItem.Kind + "]\n" }

// ---- recurse
PrintProjectTree( new Enumerator( objItem ), strTab + ' ' );

objProjectItemIter.moveNext();
}

// ----------------------------------------------------------
// Load example project installed with MapForce.
// ----------------------------------------------------------
function LoadSampleProject()
{
  // close open project
  objProject = objMapForce.ActiveProject;
  if ( objProject != null )
    objProject.Close();

  // open sample project and iterate through it.
  // sump properties of all project items

  objProject = objMapForce.OpenProject(strSamplePath + "MapForceExamples.mfp");
  strGlobalText = ';
  PrintProjectTree( new Enumerator (objProject), ' ' )
  WScript.Echo( strGlobalText );

  objProject.Close();
}

// ----------------------------------------------------------
// Create a new project with some folders, mappings and a
// Web service project.
// ----------------------------------------------------------
function CreateNewProject()
{
try
{
    // create new project and specify file to store it.
    objProject = objMapForce.NewProject(strSamplePath + "Sample.mfp");

    // create a simple folder structure
    objProject.CreateFolder( "New Folder 1" );
    objFolder1 = objProject.Item(0);
    objFolder1.CreateFolder( "New Folder 2" );
    objFolder2 = ( new Enumerator( objFolder1 ) ).item();   // an alternative to Item(0)

    // add two different mappings to folder structure
    objFolder1.AddFile( strSamplePath + "DB_Altova_SQLXML.mfd" );
    objFolder2.AddActiveFile();

    // override code generation settings for this folder
    objFolder2.CodeGenSettings_UseDefault = false;
    objFolder2.CodeGenSettings_OutputFolder = strSamplePath + "SampleOutput"
    objFolder2.CodeGenSettings_Language = 1;      //C++

    // insert Web service project based on a wsdl file from the installed examples
    objProject.InsertWebService( strSamplePath + "TimeService/TimeService.wsdl",
        TimeService",
        "TimeServiceSoap",
        true );
    objProject.Save();
    if ( ! objProject.Saved )
    WScript.Echo("problem occurred when saving project");

    // dump project tree
    strGlobalText = '';
    PrintProjectTree( new Enumerator (objProject), ' ' )
    WScript.Echo( strGlobalText );
} catch (err)
{
    ERROR("while creating new project", err );
}

// ----------------------------------------------------------
// Generate code for a project's sub-tree. Mix default code
// generation parameters and overloaded parameters.
// ----------------------------------------------------------
function GenerateCodeForNewProject()
{
    // since the Web service project contains only initial mappings,
    // we generate code only for our custom folder.
    // code generation parameters from project are used for Folder1,
    // whereas Folder2 provides overwritten values.
    objFolder = objProject.Item(0);
}
objFolder1.GenerateCode();

// /////////////////////// MAIN //////////////////////////////
CreateGlobalObjects();
objMapForce.Visible = true;
LoadSampleProject();
CreateNewProject();
GenerateCodeForNewProject();

// uncomment to shut down application when script ends
// objMapForce.Visible = false;
17.2 Object Reference

This section provides reference to the objects of the MapForce COM API. The objects are described in a generic manner, since the API may be used with virtually any language that supports calling a COM object. For language-specific examples, see Examples.

The API contains two main sections, each describing the interfaces and the enumeration types used in the API, respectively. The enumeration values contain both the string name and a numeric value. If your scripting environment does not support enumerations, use the number-values instead.

In .NET, for every interface of the MapForce COM automation interface, a .NET class exists with the same name. Also, COM types will be converted to the appropriate .NET type. For example, a type such as Long in the COM API would appear as System.Int32 in .NET.

In Java, note the following syntax variations:

- **Classes and class names.** For every interface of the MapForce automation interface, a Java class exists with the name of the interface.
- **Method names.** Method names on the Java interface are the same as used on the COM interfaces, but start with a small letter to conform to Java naming conventions. To access COM properties, Java methods that prefix the property name with get and set can be used. If a property does not support write-access, no setter method is available. For example, for the Name property of the Document interface, the Java methods getName and setName are available.
- **Enumerations.** For every enumeration defined in the automation interface, a Java enumeration is defined with the same name and values.
- **Events and event handlers.** For every interface in the automation interface that supports events, a Java interface with the same name plus 'Event' is available. To simplify the overloading of single events, a Java class with default implementations for all events is provided. The name of this Java class is the name of the event interface plus 'DefaultHandler'. For example:

```java
Application // Java class to access the application
ApplicationEvents // Events interface for the application
ApplicationEventsDefaultHandler // Default handler for "ApplicationEvents"
```

17.2.1 Interfaces

17.2.1.1 Application

The Application interface is the interface to a MapForce application object. It represents the main access point for the MapForce application itself. This interface is the starting point to do any further operations with MapForce or to retrieve or create other MapForce related automation objects. For information about creating an instance the Application object, see Accessing the API.

Properties to navigate the object model:
• Application
• Parent
• Options
• Project
• Documents

Application status:
• Visible
• Name
• Quit
• Status
• WindowHandle

MapForce designs:
• NewDocument
• OpenDocument
• OpenURL
• ActiveDocument

MapForce projects:
• NewProject
• OpenProject
• ActiveProject

MapForce code generation:
• HighlightSerializedMarker

Global resources:
• GlobalResourceConfig
• GlobalResourceFile

Version information:
• Edition
• IsAPISupported
• MajorVersion
• MinorVersion

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveDocument</td>
<td>Read-only. Returns the automation object of the currently active document. This property returns the same as Documents.ActiveDocument.</td>
</tr>
<tr>
<td>ActiveProject</td>
<td>Read-only.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Application</strong></td>
<td>Returns the automation object of the currently active project.</td>
</tr>
<tr>
<td><strong>Documents</strong></td>
<td>Read-only. Retrieves the application's top-level object.</td>
</tr>
<tr>
<td><strong>GlobalResourceConfig</strong></td>
<td>Gets or sets the name of the active global resource configuration file. By default, the file is called <strong>GlobalResources.xml</strong>. The configuration file can be renamed and saved to any location. You can therefore have multiple Global Resources XML files. However, only one of these Global Resources XML files can be active, per application, at one time, and only the definitions contained in this file will be available to the application.</td>
</tr>
<tr>
<td><strong>GlobalResourceFile</strong></td>
<td>Gets or sets the global resource definition file. By default, the file is called <strong>GlobalResources.xml</strong>.</td>
</tr>
<tr>
<td><strong>IsAPISupported</strong></td>
<td>Read-only. Returns true if the API is supported in this version of MapForce.</td>
</tr>
<tr>
<td><strong>MajorVersion</strong></td>
<td>Read-only. Gets the major version number of MapForce. The version is calculated starting from 1998, and is incremented by one every year. For example, the major version is &quot;18&quot; for the release 2016.</td>
</tr>
<tr>
<td><strong>MinorVersion</strong></td>
<td>Read-only. The minor version number of the product, e.g. 2 for 2006 R2 SP1.</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>Read-only. The name of the application.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Read-only. This property gives access to options that configure the generation of code.</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Read-only. The parent object according to the object model.</td>
</tr>
<tr>
<td><strong>ServicePackVersion</strong></td>
<td>Read-only. The service pack version number of the product, e.g. 1</td>
</tr>
</tbody>
</table>
### Name | Description
---|---
for 2016 R2 SP1. |  

**Status**  
Read-only.  
The status of the application. It is one of the values of the ENUMApplicationStatus enumeration.

**Visible**  
**True** if MapForce is displayed on the screen (though it might be covered by other applications or be iconized).  
**False** if MapForce is hidden. The default value for MapForce when automatically started due to a request from the automation server Application is false. In all other cases, the property is initialized to **true**.  
An application instance that is visible is said to be controlled by the user (and possibly by clients connected via the automation interface). It will only shut down due to an explicit user request. To shut down an application instance, set its visibility to false and clear all references to this instance within your program. The application instance will shut down automatically when no further COM clients are holding references to it.

**WindowHandle**  
Read-only.  
Retrieves the application's Window Handle.

### Methods

| Name | Description |
---|---|
**HighlightSerializedMarker**  
Use this method to highlight a location in a mapping file that has been previously serialized. If the corresponding document has not already been loaded, it will be loaded first. See Document.GenerateCodeEx for a method to retrieve a serialized marker.

**NewDocument**  
Creates a new empty document. The newly opened document becomes the ActiveDocument. This method is a shortened form of Documents.NewDocument.

**NewProject**  
Creates a new empty project. The current project is closed. The new project is accessible under ActiveProject.

**NewWebServiceProject**  
Creates a new empty Web Service project. The new project is accessible under ActiveProject. This method is available in MapForce Enterprise Edition only.

**OpenDocument**  
Loads a previously saved document file and continues working on it. The newly opened document becomes the ActiveDocument. This method is a shorter form of
### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documents.OpenDocument</td>
<td></td>
</tr>
<tr>
<td>OpenProject</td>
<td>Opens an existing Mapforce project (*.mfp). The current project is closed. The newly opened project is accessible under ActiveProject.</td>
</tr>
<tr>
<td>OpenURL</td>
<td>Loads a previously saved document file from an URL location. Allows user name and password to be supplied.</td>
</tr>
<tr>
<td>Quit</td>
<td>Disconnects from MapForce to allow the application to shutdown. Calling this method is optional since MapForce keeps track of all external COM connections and automatically recognizes a disconnection. For more information on automatic shutdown see the Visible property.</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnDocumentOpened</td>
<td>This event is triggered when an existing or new document is opened. The corresponding close event is Document.OnDocumentClosed.</td>
</tr>
<tr>
<td>OnProjectOpened</td>
<td>This event is triggered when an existing or new project is loaded into the application. The corresponding close event is Project.OnProjectClosed.</td>
</tr>
<tr>
<td>OnShutdown</td>
<td>This event is triggered when the application is shutting down.</td>
</tr>
</tbody>
</table>

### 17.2.1.1.1 Properties

#### ActiveDocument

Returns the automation object of the currently active document. This property returns the same as Documents.ActiveDocument.

**Signature**

ActiveDocument : Document
Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.1.2  ActiveProject

Returns the automation object of the currently active project.

Signature

```plaintext
ActiveProject : Project
```

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.1.3  Application

Retrieves the application’s top-level object.

Signature

```plaintext
Application : Application
```

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.1.1.4  Documents

Returns a collection of all currently open documents.

Signature

<table>
<thead>
<tr>
<th>Documents : Documents</th>
</tr>
</thead>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.1.1.5  Edition


Signature

<table>
<thead>
<tr>
<th>Edition : String</th>
</tr>
</thead>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.1.1.6  GlobalResourceConfig

Gets or sets the name of the active global resource configuration file. By default, the file is called GlobalResources.xml.

The configuration file can be renamed and saved to any location. You can therefore have multiple Global Resources XML files. However, only one of these Global Resources XML File can be active, per application, at one time, and only the definitions contained in this file will be available to the application.

Signature

<table>
<thead>
<tr>
<th>GlobalResourceConfig : String</th>
</tr>
</thead>
</table>
### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

#### 17.2.1.1.7 GlobalResourceFile

Gets or sets the global resource definition file. By default, the file is called GlobalResources.xml.

**Signature**

```csharp
GlobalResourceFile : String
```

### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

#### 17.2.1.1.8 IsAPISupported

Returns true if the API is supported in this version of MapForce.

**Signature**

```csharp
IsAPISupported : Boolean
```

### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

#### 17.2.1.1.9 MajorVersion

Gets the major version number of MapForce. The version is calculated starting from 1998, and is incremented by one every year. For example, the major version is "18" for the release 2016.
**Signature**

| MajorVersion : Long |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

**17.2.1.1.10 MinorVersion**

The minor version number of the product, e.g. 2 for 2006 R2 SP1.

**Signature**

| MinorVersion : Long |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

**17.2.1.1.11 Name**

The name of the application.

**Signature**

| Name : String |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.1.12  **Options**

This property gives access to options that configure the generation of code.

**Signature**

| Options : Options |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.1.13  **Parent**

The parent object according to the object model.

**Signature**

| Parent : Application |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.1.14  **ServicePack Version**

The service pack version number of the product, e.g. 1 for 2016 R2 SP1.

**Signature**

| ServicePackVersion : Long |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
</tbody>
</table>
17.2.1.1.15  Status

The status of the application. It is one of the values of the `ENUMApplicationStatus` enumeration.

**Signature**

```
Status : ENUMApplicationStatus
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.1.16  Visible

*True* if MapForce is displayed on the screen (though it might be covered by other applications or be iconized).

*False* if MapForce is hidden. The default value for MapForce when automatically started due to a request from the automation server `Application` is false. In all other cases, the property is initialized to `true`.

An application instance that is visible is said to be controlled by the user (and possibly by clients connected via the automation interface). It will only shut down due to an explicit user request. To shut down an application instance, set its visibility to false and clear all references to this instance within your program. The application instance will shut down automatically when no further COM clients are holding references to it.

**Signature**

```
Visible : Boolean
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.1.1.17  WindowHandle

Retrieves the application's Window Handle.

**Signature**

```
WindowHandle : Long
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.1.2  Methods

17.2.1.1.2.1  HighlightSerializedMarker

Use this method to highlight a location in a mapping file that has been previously serialized. If the corresponding document has not already been loaded, it will be loaded first. See `Document.GenerateCodeEx` for a method to retrieve a serialized marker.

**Signature**

```
HighlightSerializedMarker(in i_strSerializedMarker: String) : Void
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_strSerializedMarker</td>
<td>String</td>
<td>The <code>ErrorMarker</code> object to highlight. Use <code>ErrorMaker.Serialized</code> to obtain this value.</td>
</tr>
</tbody>
</table>

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1007</td>
<td>The string passed in <code>i_strSerializedMarker</code> is not recognized as a</td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>serialized MapForce marker.</td>
</tr>
<tr>
<td>1008</td>
<td>The marker points to a location that is no longer valid.</td>
</tr>
</tbody>
</table>

### 17.2.1.2.2 NewDocument

Creates a new empty document. The newly opened document becomes the `ActiveDocument`. This method is a shortened form of `Documents.NewDocument`.

#### Signature

```
NewDocument() : Document
```

#### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

### 17.2.1.2.3 NewProject

Creates a new empty project. The current project is closed. The new project is accessible under `ActiveProject`.

#### Signature

```
NewProject() : Project
```

#### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

### 17.2.1.2.4 NewWebServiceProject

Creates a new empty Web Service project. The new project is accessible under `ActiveProject`. This method is available in MapForce Enterprise Edition only.
**Signature**

```
NewWebServiceProject() : Project
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1004</td>
<td>Error creating new project.</td>
</tr>
<tr>
<td>1005</td>
<td>Wrong edition of MapForce.</td>
</tr>
</tbody>
</table>

### 17.2.1.1.2.5 OpenDocument

Loads a previously saved document file and continues working on it. The newly opened document becomes the `ActiveDocument`. This method is a shorter form of `Documents.OpenDocument`.

**Signature**

```
OpenDocument(in i_strFileName:String) : Document
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_strFileName</td>
<td>String</td>
<td>The path of the document to open.</td>
</tr>
</tbody>
</table>

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

### 17.2.1.1.2.6 OpenProject

Opens an existing Mapforce project (*.mfp). The current project is closed. The newly opened project is accessible under `ActiveProject`.

**Signature**

```
OpenProject(in i_strFileName:String) : Project
```
Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_strFileName</td>
<td>String</td>
<td>The path of the project to open.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1002</td>
<td>The supplied filename is not valid.</td>
</tr>
</tbody>
</table>

17.2.1.1.2.7  OpenURL

Loads a previously saved document file from an URL location. Allows user name and password to be supplied.

Signature

```
OpenURL(in strURL: String, in strUser: String, in strPassword: String) : Void
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strURL</td>
<td>String</td>
<td>The URL from which the document should be loaded.</td>
</tr>
<tr>
<td>strUser</td>
<td>String</td>
<td>The username required to access the URL.</td>
</tr>
<tr>
<td>strPassword</td>
<td>String</td>
<td>The password required to access the URL.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1002</td>
<td>The supplied URL is not valid.</td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>1006</td>
<td>Error while opening the URL file.</td>
</tr>
</tbody>
</table>

### 17.2.1.1.2.8 Quit

Disconnects from MapForce to allow the application to shutdown. Calling this method is optional since MapForce keeps track of all external COM connections and automatically recognizes a disconnection. For more information on automatic shutdown see the Visible property.

**Signature**

```csharp
Quit() : Void
```

### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

### 17.2.1.1.3 Events

#### 17.2.1.1.3.1 OnDocumentOpened

This event is triggered when an existing or new document is opened. The corresponding close event is `Document.OnDocumentClosed`.

**Signature**

```csharp
OnDocumentOpened(in Document): Void
```

#### 17.2.1.1.3.2 OnProjectOpened

This event is triggered when an existing or new project is loaded into the application. The corresponding close event is `Project.OnProjectClosed`.

**Signature**

```csharp
OnProjectOpened(in Project): Void
```
17.2.1.3.3  OnShutdown

This event is triggered when the application is shutting down.

Signature

OnShutdown(): Void

17.2.1.2  AppOutputLine

Represents a message line. In contrast to ErrorMarker, its structure is more detailed and can contain a collection of child lines, therefore forming a tree of message lines.

Properties to navigate the object model:

- Application
- Parent

Line access:

- GetLineSeverity
- GetLineSymbol
- GetLineText
- GetLineTextEx
- GetLineTextWithChildren
- GetLineTextWithChildrenEx

A single AppOutputLine consists of one or more sub-lines. Sub-line access:

- GetLineCount

A sub-line consists of one or more cells. Cell access:

- GetCellCountInLine
- GetCellIcon
- GetCellSymbol
- GetCellText
- GetCellTextDecoration
- GetIsCellText

Below an AppOutputLine there can be zero, one, or more child lines which themselves are of type AppOutputLine, which thus form a tree structure.

Child lines access:

- ChildLines
### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Read-only. Retrieves the application's top-level object.</td>
</tr>
<tr>
<td>ChildLines</td>
<td>Read-only. Returns a collection of the current line's direct child lines.</td>
</tr>
<tr>
<td>Parent</td>
<td>Read-only. The parent object according to the object model.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetCellCountInLine</td>
<td>Gets the number of cells in the sub-line indicated by nLine in the current AppOutputLine.</td>
</tr>
<tr>
<td>GetCellIcon</td>
<td>Gets the icon of the cell indicated by nCell in the current AppOutputLine's sub-line indicated by nLine</td>
</tr>
<tr>
<td>GetCellSymbol</td>
<td>Gets the symbol of the cell indicated by nCell in the current AppOutputLine's sub-line indicated by nLine</td>
</tr>
<tr>
<td>GetCellText</td>
<td>Gets the text of the cell indicated by nCell in the current AppOutputLine's sub-line indicated by nLine</td>
</tr>
<tr>
<td>GetCellTextDecoration</td>
<td>Gets the decoration of the text cell indicated by nCell in the current AppOutputLine's sub-line indicated by nLine. It can be one of the ENUMAppOutputLine_TextDecoration values.</td>
</tr>
<tr>
<td>GetIsCellText</td>
<td>Returns true if the cell indicated by nCell in the current AppOutputLine's sub-line indicated by nLine is a text cell.</td>
</tr>
<tr>
<td>GetLineCount</td>
<td>Gets the number of sub-lines the current line consists of.</td>
</tr>
<tr>
<td>GetLineSeverity</td>
<td>Gets the severity of the line. It can be one of the ENUMAppOutputLine_Severity values.</td>
</tr>
<tr>
<td>GetLineSymbol</td>
<td>Gets the symbol assigned to the whole line.</td>
</tr>
<tr>
<td>GetLineText</td>
<td>Gets the contents of the line as text.</td>
</tr>
<tr>
<td>GetLineTextEx</td>
<td>Gets the contents of the line as text using the specified part and line separators.</td>
</tr>
<tr>
<td>GetLineTextWithChildren</td>
<td>Gets the contents of the line including all child and descendant lines as text.</td>
</tr>
</tbody>
</table>
### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetLineTextWithChildrenEx</td>
<td>Gets the contents of the line including all child and descendant lines as text using the specified part, line, tab and item separators.</td>
</tr>
</tbody>
</table>

### 17.2.1.2.1 Properties

#### 17.2.1.2.1.1 Application

Retrieves the application's top-level object.

**Signature**

| Application : Application |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4101</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

#### 17.2.1.2.1.2 ChildLines

Returns a collection of the current line's direct child lines.

**Signature**

| ChildLines : AppOutputLines |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4101</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.2.1.3  Parent

The parent object according to the object model.

Signature

| Parent : AppOutputLines |

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4101</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.2.2  Methods

17.2.1.2.2.1  GetCellCountInLine

Gets the number of cells in the sub-line indicated by \texttt{nLine} in the current \texttt{AppOutputLine}.

Signature

| GetCellCountInLine(in nLine:Long) : Long |

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nLine</td>
<td>Long</td>
<td>Specifies the zero-based index of the line.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4101</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.2.2.2 GetCellIcon (obsolete)

Gets the icon of the cell indicated by \texttt{nCell} in the current \texttt{AppOutputLine}'s sub-line indicated by \texttt{nLine}.

**Signature**

\[
\text{GetCellIcon}(\text{nLine:Long, in nCell:Long}) : \text{Long}
\]

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nLine</td>
<td>Long</td>
<td></td>
</tr>
<tr>
<td>nCell</td>
<td>Long</td>
<td></td>
</tr>
</tbody>
</table>

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4101</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.2.2.3 GetCellSymbol

Gets the symbol of the cell indicated by \texttt{nCell} in the current \texttt{AppOutputLine}'s sub-line indicated by \texttt{nLine}.

**Signature**

\[
\text{GetCellSymbol}(\text{nLine:Long, in nCell:Long}) : \text{AppOutputLineSymbol}
\]

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nLine</td>
<td>Long</td>
<td>Specifies the zero-based index of the line.</td>
</tr>
<tr>
<td>nCell</td>
<td>Long</td>
<td>Specifies the zero-based index of the cell.</td>
</tr>
</tbody>
</table>
Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4101</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.2.2.4  GetCellText

Gets the text of the cell indicated by \texttt{nCell} in the current \texttt{AppOutputLine}'s sub-line indicated by \texttt{nLine}.

Signature

\texttt{GetCellText(in nLine:Long, in nCell:Long) : String}

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{nLine}</td>
<td>\texttt{Long}</td>
<td>Specifies the zero-based index of the line.</td>
</tr>
<tr>
<td>\texttt{nCell}</td>
<td>\texttt{Long}</td>
<td>Specifies the zero-based index of the cell.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4101</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.2.2.5  GetCellTextDecoration

Gets the decoration of the text cell indicated by \texttt{nCell} in the current \texttt{AppOutputLine}'s sub-line indicated by \texttt{nLine}. It can be one of the \texttt{ENUMAppOutputLine_TextDecoration} values.

Signature

\texttt{GetCellTextDecoration(in nLine:Long, in nCell:Long) : Long}
Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nLine</td>
<td>Long</td>
<td>Specifies the zero-based index of the line.</td>
</tr>
<tr>
<td>nCell</td>
<td>Long</td>
<td>Specifies the zero-based index of the cell.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4101</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.2.2.6  GetIsCellText

Returns true if the cell indicated by \text{nCell} in the current \text{AppOutputLine}'s sub-line indicated by \text{nLine} is a text cell.

Signature

\text{GetIsCellText(in nLine:Long, in nCell:Long) : Boolean}

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nLine</td>
<td>Long</td>
<td>Specifies the zero-based index of the line.</td>
</tr>
<tr>
<td>nCell</td>
<td>Long</td>
<td>Specifies the zero-based index of the cell.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4101</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.2.2.7  \textit{GetLineCount}

Gets the number of sub-lines the current line consists of.

\textbf{Signature}

\begin{verbatim}
GetLineCount() : Long
\end{verbatim}

\textbf{Errors}

\begin{center}
\begin{tabular}{|l|l|}
\hline
\textbf{Error code} & \textbf{Description} \\
\hline
4100 & The object is no longer valid. \\
4101 & Invalid address for the return parameter was specified. \\
\hline
\end{tabular}
\end{center}

17.2.1.2.2.8  \textit{GetLineSeverity}

Gets the severity of the line. It can be one of the \texttt{ENUMAppOutputLine_Severity} values.

\textbf{Signature}

\begin{verbatim}
GetLineSeverity() : Long
\end{verbatim}

\textbf{Errors}

\begin{center}
\begin{tabular}{|l|l|}
\hline
\textbf{Error code} & \textbf{Description} \\
\hline
4100 & The object is no longer valid. \\
4101 & Invalid address for the return parameter was specified. \\
\hline
\end{tabular}
\end{center}

17.2.1.2.2.9  \textit{GetLineSymbol}

Gets the symbol assigned to the whole line.

\textbf{Signature}

\begin{verbatim}
GetLineSymbol() : AppOutputLineSymbol
\end{verbatim}

\textbf{Errors}

\begin{center}
\begin{tabular}{|l|l|}
\hline
\textbf{Error code} & \textbf{Description} \\
\hline
4100 & The object is no longer valid. \\
\hline
\end{tabular}
\end{center}
### 17.2.1.2.2.10 GetLineText

**Summary**

Gets the contents of the line as text.

**Signature**

```
GetLineText() : String
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4101</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The object is no longer valid.</td>
</tr>
</tbody>
</table>

### 17.2.1.2.2.11 GetLineTextEx

**Summary**

Gets the contents of the line as text using the specified part and line separators.

**Signature**

```
GetLineTextEx(in psTextPartSeperator: String, in psLineSeperator: String) : String
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psTextPartSeperator</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>psLineSeperator</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4101</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.2.2.12 GetLineTextWithChildren

Gets the contents of the line including all child and descendant lines as text.

**Signature**

```plaintext
GetLineTextWithChildren() : String
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4101</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.2.2.13 GetLineTextWithChildrenEx

Gets the contents of the line including all child and descendant lines as text using the specified part, line, tab and item separators.

**Signature**

```plaintext
GetLineTextWithChildrenEx(in psPartSep:String, in psLineSep:String, in psTabSep:String, in psItemSep:String) : String
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psPartSep</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>psLineSep</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>psTabSep</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>psItemSep</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4101</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.3  **AppOutputLines**

Represents a collection of AppOutputLine message lines.

Properties to navigate the object model:

- Application
- Parent

Iterating through the collection:

- Count
- Item

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Read-only. Retrieves the application's top-level object.</td>
</tr>
<tr>
<td>Count</td>
<td>Read-only. Retrieves the number of lines in the collection.</td>
</tr>
<tr>
<td>Item</td>
<td>Read-only. Retrieves the line at index ( n ) from the collection. Indices start with 1.</td>
</tr>
<tr>
<td>Parent</td>
<td>Read-only. The parent object according to the object model.</td>
</tr>
</tbody>
</table>

17.2.1.3.1  **Properties**

17.2.1.3.1.1  **Application**

Retrieves the application's top-level object.

**Signature**

```
Application : Application
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>4001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.3.1.2  **Count**

Retrieves the number of lines in the collection.

**Signature**

```
Count : Integer
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.3.1.3  **Item**

Retrieves the line at index $n$ from the collection. Indices start with 1.

**Signature**

```
Item this(in n:Integer) : AppOutputLine
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.3.1.4  **Parent**

The parent object according to the object model.

**Signature**

```
Parent : AppOutputLine
```
Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.4  **AppOutputLineSymbol**

An **AppOutputLineSymbol** represents a link in an **AppOutputLine message line** which can be clicked in the MapForce Messages window. It is applied to a cell of an **AppOutputLine** or to the whole line itself.

Properties to navigate the object model:

- Application
- Parent

Access to **AppOutputLineSymbol** methods:

- GetSymbolHREF
- GetSymbolID
- IsSymbolHREF

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Read-only. Retrieves the application's top-level object.</td>
</tr>
<tr>
<td>Parent</td>
<td>Read-only. The parent object according to the object model.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetSymbolHREF</td>
<td>If the symbol is of type URL, returns the URL as a string.</td>
</tr>
<tr>
<td>GetSymbolID</td>
<td>Gets the ID of the symbol.</td>
</tr>
<tr>
<td>IsSymbolHREF</td>
<td>Returns true if the symbol is of kind URL.</td>
</tr>
</tbody>
</table>
17.2.1.4.1 Properties

17.2.1.4.1.1 Application

Retrieves the application's top-level object.

Signature

| Application : Application |

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.4.1.2 Parent

The parent object according to the object model.

Signature

| Parent : Application |

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.4.2 Methods

17.2.1.4.2.1 GetSymbolHREF

If the symbol is of type URL, returns the URL as a string.

Signature

| GetSymbolHREF() : String |
Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.4.2.2  GetSymbolID

Gets the ID of the symbol.

Signature

```plaintext
GetSymbolID() : Long
```

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.4.2.3  IsSymbolHREF

Returns true if the symbol is of kind URL.

Signature

```plaintext
IsSymbolHREF() : Boolean
```

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.5  Component

A Component represents a MapForce component.

To navigate the control, use the Application and Parent properties.
Component properties:

- HasIncomingConnections
- HasOutgoingConnections
- CanChangeInputInstanceFile
- CanChangeOutputInstanceFile
- ComponentName
- ID
- IsParameterInputRequired
- IsParameterSequence
- Name
- Preview
- Schema
- SubType
- Type

Instance related properties:

- InputInstanceFile
- OutputInstanceFile

Datapoints:

- GetRootDatapoint

Execution:

- GenerateOutput

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Read-only. Retrieves the application's top-level object.</td>
</tr>
<tr>
<td>CanChangeInputInstanceFile</td>
<td>Read-only. Indicates if the input instance file name can be changed.</td>
</tr>
<tr>
<td></td>
<td>Returns false if the component has a filename node and this node has a connection on its left (input) side, otherwise returns true. If the component does not have a filename node, false is returned.</td>
</tr>
<tr>
<td>CanChangeOutputInstanceFile</td>
<td>Read-only. Indicates if the output instance file name can be changed.</td>
</tr>
<tr>
<td></td>
<td>Returns false if the component has a filename node and this node has a connection on its left (input) side, otherwise returns true. If the component does not have a filename node, false is returned.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ComponentName</strong></td>
<td>Gets or sets the component's name.</td>
</tr>
<tr>
<td><strong>HasIncomingConnections</strong></td>
<td>Read-only. Indicates if the component has any incoming connections (on its left side) not including the filename node. An incoming connection on the filename node does not have any effect on the returned value.</td>
</tr>
<tr>
<td><strong>HasOutgoingConnections</strong></td>
<td>Read-only. Indicates if the component has any outgoing connections (on its right side).</td>
</tr>
<tr>
<td><strong>ID</strong></td>
<td>Read-only. Retrieves the component ID.</td>
</tr>
<tr>
<td><strong>InputInstanceFile</strong></td>
<td>Gets or sets the component's input instance file.</td>
</tr>
<tr>
<td><strong>IsParameterInputRequired</strong></td>
<td>Gets or sets, if the input parameter component requires an ingoing connection on the function call component of the user-defined function this input parameter component is in. This property works only for components, which are input parameter components.</td>
</tr>
<tr>
<td><strong>IsParameterSequence</strong></td>
<td>Gets or sets, if the input or output parameter component supports sequences. This property works only for components, which are input or output parameter components.</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>Read-only. Gets the component's name.</td>
</tr>
<tr>
<td><strong>OutputInstanceFile</strong></td>
<td>Gets or sets the component's output instance file.</td>
</tr>
<tr>
<td></td>
<td>Trying to access the <code>OutputInstanceFile</code> of a component via the API does not return any data if the “File” connector of the component has been connected to another item in the mapping.</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Read-only. The parent object according to the object model.</td>
</tr>
<tr>
<td><strong>Preview</strong></td>
<td>Gets or sets if the component is the current preview component.</td>
</tr>
<tr>
<td></td>
<td>This property works only for components, which are target components in the document's main mapping. Only one target component in the main mapping can be the preview component at any time.</td>
</tr>
<tr>
<td></td>
<td>When setting this property, it is only possible to set it to true. This then will also implicitly set the <code>Preview</code> property of all other components to false.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>If there is just a single target component in the main mapping, it is also the preview component.</td>
</tr>
<tr>
<td><strong>Schema</strong></td>
<td>Read-only. Retrieves the component's schema file name.</td>
</tr>
<tr>
<td><strong>SubType</strong></td>
<td>Read-only. Retrieves the component's subtype.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Read-only. Retrieves the component's type.</td>
</tr>
<tr>
<td><strong>UsageKind</strong></td>
<td>Read-only. Retrieves the component's usage kind.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GenerateOutput</strong></td>
<td>Generates the output file(s) defined in the mapping for the current component only, using a MapForce internal mapping language. The name(s) of the output file(s) are defined as property of the current component which is the output item in the mapping for this generation process.</td>
</tr>
<tr>
<td><strong>GetRootDatapoint</strong></td>
<td>Gets a root datapoint on the left (input) or right (output) side of a component. To access children and descendants, the Datapoint object provides further methods.</td>
</tr>
</tbody>
</table>

### 17.2.1.5.1 Properties

### 17.2.1.5.1.1 Application

Retrieves the application's top-level object.

**Signature**

| Application : Application |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
</tbody>
</table>
### 17.2.1.5.1.2 CanChangeInputInstanceFile

Indicates if the input instance file name can be changed.

Returns false if the component has a filename node and this node has a connection on its left (input) side, otherwise returns true. If the component does not have a filename node, false is returned.

**Signature**

```plaintext
CanChangeInputInstanceFile : Boolean
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

### 17.2.1.5.1.3 CanChangeOutputInstanceFile

Indicates if the output instance file name can be changed.

Returns false if the component has a filename node and this node has a connection on its left (input) side, otherwise returns true. If the component does not have a filename node, false is returned.

**Signature**

```plaintext
CanChangeOutputInstanceFile : Boolean
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.5.1.4 ComponentName

Gets or sets the component's name.

**Signature**

| ComponentName : String |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1246</td>
<td>The component does not support setting its name.</td>
</tr>
<tr>
<td>1247</td>
<td>Invalid component name.</td>
</tr>
</tbody>
</table>

17.2.1.5.1.5 HasIncomingConnections

Indicates if the component has any incoming connections (on its left side) not including the filename node. An incoming connection on the filename node does not have any effect on the returned value.

**Signature**

| HasIncomingConnections : Boolean |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.5.1.6 HasOutgoingConnections

Indicates if the component has any outgoing connections (on its right side).

**Signature**

| HasOutgoingConnections : Boolean |
Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.5.1.7  **ID**

Retrieves the component ID.

**Signature**

| ID : Long |

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.5.1.8  **InputInstanceFile**

Gets or sets the component's input instance file.

**Signature**

| InputInstanceFile : String |

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.5.1.9  **IsParameterInputRequired**

Gets or sets, if the input parameter component requires an ingoing connection on the function call component of the user-defined function this input parameter component is in. This property works only for components, which are input parameter components.
### Signature

**IsParameterInputRequired** : Boolean

### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1232</td>
<td>This operation works only for an input parameter component.</td>
</tr>
<tr>
<td>1240</td>
<td>Changing the document not allowed. It is read-only.</td>
</tr>
</tbody>
</table>

### 17.2.1.5.1.10 **IsParameterSequence**

Gets or sets, if the input or output parameter component supports sequences. This property works only for components, which are input or output parameter components.

### Signature

**IsParameterSequence** : Boolean

### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1233</td>
<td>This operation works only for an input or output parameter component.</td>
</tr>
<tr>
<td>1240</td>
<td>Changing the document not allowed. It is read-only.</td>
</tr>
</tbody>
</table>

### 17.2.1.5.1.11 **Name**

Gets the component's name.

### Signature

**Name** : String
17.2.1.5.1.12  OutputInstanceFile

Gets or sets the component's output instance file.

Trying to access the OutputInstanceFile of a component via the API does not return any data if the “File” connector of the component has been connected to another item in the mapping.

**Signature**

```
OutputInstanceFile : String
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.5.1.13  Parent

The parent object according to the object model.

**Signature**

```
Parent : Mapping
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.5.1.14  Preview

Gets or sets if the component is the current preview component.

This property works only for components, which are target components in the document's main mapping. Only one target component in the main mapping can be the preview component at any time.

When setting this property, it is only possible to set it to true. This then will also implicitly set the Preview property of all other components to false.

If there is just a single target component in the main mapping, it is also the preview component.

**Signature**

```
Preview : Boolean
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1234</td>
<td>Only a target component in the main mapping can be set as preview component.</td>
</tr>
<tr>
<td>1235</td>
<td>A component cannot be set as non-preview component. Set another component as preview component instead.</td>
</tr>
</tbody>
</table>

17.2.1.5.1.15  Schema

Retrieves the component's schema file name.

**Signature**

```
Schema : String
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.5.1.16  **SubType**

Retrieves the component's subtype.

**Signature**

| SubType | ENUMComponentSubType |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.5.1.17  **Type**

Retrieves the component's type.

**Signature**

| Type | ENUMComponentType |

17.2.1.5.1.18  **UsageKind**

Retrieves the component's usage kind.

**Signature**

| UsageKind | ENUMComponentUsageKind |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.5.2  Methods

17.2.1.5.2.1  GenerateOutput

Generates the output file(s) defined in the mapping for the current component only, using a MapForce internal mapping language. The name(s) of the output file(s) are defined as property of the current component which is the output item in the mapping for this generation process.

Signature

```plaintext
GenerateOutput(out pbError:Boolean) : AppOutputLines
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbError</td>
<td>Boolean</td>
<td>This is an output-only parameter. You will receive a value only if the calling language supports output parameters. If not, the value you pass here will remain unchanged when the function has finished.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1248</td>
<td>Generating output is only supported when the graphical user interface is visible.</td>
</tr>
</tbody>
</table>

17.2.1.5.2.2  GetRootDatapoint

Gets a root datapoint on the left (input) or right (output) side of a component. To access children and descendants, the Datapoint object provides further methods.

Signature

```plaintext
GetRootDatapoint(in side:ENUMComponentDatapointSide, in strNamespace:String, in strLocalName:String, in strParameterName:String) : Datapoint
```
### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>side</strong></td>
<td>ENUMComponentDatapointSide</td>
<td>The side parameter indicates if an input, or output, datapoint of a component is to be retrieved.</td>
</tr>
<tr>
<td><strong>strNamespace</strong></td>
<td>String</td>
<td>The specified namespace and local name, indicate the specific name of the node whose datapoint is to be retrieved. For components with structural information such as schema components, you will have to provide the namespace together with the local name, or you can just pass an empty string for the namespace. File-based components like the schema component contain a special node on their root, the filename node. There, GetRootDatapoint can only find the filename node. You will have to pass namespace &quot;<a href="http://www.altova.com/mapforce">http://www.altova.com/mapforce</a>&quot; and local name &quot;FileInstance&quot; to retrieve a datapoint of this node.</td>
</tr>
<tr>
<td><strong>strLocalName</strong></td>
<td>String</td>
<td>See above.</td>
</tr>
<tr>
<td><strong>strParameterName</strong></td>
<td>String</td>
<td>The specified parameter name should be an empty string unless the component in question is a function call component. Since a user-defined function might contain input or output parameters of the same structure, the function call component calling this user-defined function can have more than one root node with an identical namespace and local name.</td>
</tr>
</tbody>
</table>
They will then differ only by their parameter names, which are in fact the names of the according parameter components in the user-defined function mapping itself.

It is not mandatory to specify the parameter name, though. In that case, the method will return the first root datapoint matching the specified namespace and local name.

### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1248</td>
<td>Datapoint not found.</td>
</tr>
</tbody>
</table>

### 17.2.1.6 Components

Represents a collection of Component objects.

To navigate the object model, use the following properties:

- Application
- Parent

To iterate through the collection:

- Count
- Item

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Read-only. Retrieves the application's top-level object.</td>
</tr>
<tr>
<td>Count</td>
<td>Read-only. Retrieves the number of components in the collection.</td>
</tr>
</tbody>
</table>
### 17.2.1.6.1 Properties

#### 17.2.1.6.1.1 Application

Retrieves the application's top-level object.

**Signature**

```
Application : Application
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

#### 17.2.1.6.1.2 Count

Retrieves the number of components in the collection.

**Signature**

```
Count : Integer
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.6.1.3  Item

Retrieves the component at index \( n \) from the collection. Indices start with 1.

**Signature**

```
Item this(in n:Integer) : Component
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.6.1.4  Parent

The parent object according to the object model.

**Signature**

```
Parent : Mapping
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.7  Connection

A `Connection` object represents a connector between two components.

To navigate the object model, use the following properties:

- Application
- Parent

To get or set the connection's type, use `ConnectionType`. 
Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Read-only. Retrieves the application's top-level object.</td>
</tr>
<tr>
<td>ConnectionType</td>
<td>Gets or sets the connection's type.</td>
</tr>
<tr>
<td>Parent</td>
<td>Read-only. The parent object according to the object model.</td>
</tr>
</tbody>
</table>

17.2.1.7.1 Properties

17.2.1.7.1.1 Application

Retrieves the application's top-level object.

Signature

Application : Application

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2100</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>2101</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.7.1.2 ConnectionType

Gets or sets the connection's type.

Signature

ConnectionType : ENUMConnectionType

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2100</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>2101</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
### 17.2.1.7.1.3 Parent

The parent object according to the object model.

**Signature**

Parent : Mapping

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2100</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>2101</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

### 17.2.1.8 Datapoint

A Datapoint object represents an input or output icon of a component.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Read-only. Retrieves the application's top-level object.</td>
</tr>
<tr>
<td>Parent</td>
<td>Read-only. The parent object according to the object model.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetChild</td>
<td>Scans for a direct child datapoint of the current datapoint, by namespace and local name.</td>
</tr>
</tbody>
</table>

If a schema component has elements that contain mixed content, each displays an additional child node, the so-called text() node. To retrieve a datapoint of a text() node, you will have to pass an empty string in strNamespace as well as "#text" in strLocalName and...
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eSearchDatapointElement in searchFlags.</td>
<td></td>
</tr>
</tbody>
</table>

### 17.2.1.8.1 Properties

#### 17.2.1.8.1.1 Application

Retrieves the application's top-level object.

**Signature**

| Application : Application |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>2001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

#### 17.2.1.8.1.2 Parent

The parent object according to the object model.

**Signature**

| Parent : Component |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>2001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.8.2  Methods

17.2.1.8.2.1  GetChild

Scans for a direct child datapoint of the current datapoint, by namespace and local name.

If a schema component has elements that contain mixed content, each displays an additional child node, the so-called text() node. To retrieve a datapoint of a text() node, you will have to pass an empty string in strNamespace as well as "#text" in strLocalName and eSearchDatapointElement in searchFlags.

Signature

GetChild(in strNamespace:String, in strLocalName:String, in searchFlags:ENUMSearchDatapointFlags) : Datapoint

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strNamespace</td>
<td>String</td>
<td>The namespace of the direct child datapoint.</td>
</tr>
<tr>
<td>strLocalName</td>
<td>String</td>
<td>The name of the direct child datapoint.</td>
</tr>
<tr>
<td>searchFlags</td>
<td>ENUMSearchDatapointFlags</td>
<td>Search flags can be passed as combination of values (combined using binary OR) of the ENUMSearchDatapointFlags enumeration.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>2001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>2002</td>
<td>Datapoint not found.</td>
</tr>
</tbody>
</table>

17.2.1.9  Document

A Document object represents a MapForce document (a loaded MFD file). A document contains a main mapping and zero or more local user-defined-function mappings.
To navigate the object model, use the following properties:

- Application
- Parent

For file handling, use:

- Activate
- Close
- FullName
- Name
- Path
- Saved
- Save
- SaveAs

For mapping handling, use:

- MainMapping
- Mappings
- CreateUserDefinedFunction

For component handling, use:

- FindComponentByID

For code generation, use:

- OutputSettings_ApplicationName
- JavaSettings_BasePackageName
- GenerateHashCode
- GenerateCodeEx
- GenerateCppCode
- GenerateJavaCode
- GenerateXQuery
- GenerateXSLT
- GenerateXSLT2
- HighlightSerializedMarker

For mapping execution, use:

- GenerateOutput
- GenerateOutputEx

View access:

- MapForceView

Obsolete:

- OutputSettings_Encoding
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application</strong></td>
<td>Read-only. Retrieves the application's top-level object.</td>
</tr>
<tr>
<td><strong>FullName</strong></td>
<td>Path and name of the document file.</td>
</tr>
<tr>
<td><strong>JavaSettings_BasePackageName</strong></td>
<td>Sets or retrieves the base package name used when generating Java code. In the MapForce graphical user interface, this setting is available in the <strong>Mapping Settings</strong> dialog box (right-click the mapping and select <strong>Mapping Settings</strong> from the context menu).</td>
</tr>
<tr>
<td><strong>MainMapping</strong></td>
<td>Read-only. Retrieves the main mapping of the document.</td>
</tr>
<tr>
<td><strong>MapForceView</strong></td>
<td>Read-only. This property gives access to functionality specific to the MapForce view.</td>
</tr>
<tr>
<td><strong>Mappings</strong></td>
<td>Read-only. Returns a collection of the mappings contained in the document.</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>Read-only. Name of the document file without file path.</td>
</tr>
<tr>
<td><strong>OutputSettings_ApplicationName</strong></td>
<td>Sets or retrieves the application name available in the <strong>Mapping Settings</strong> dialog box (To display this dialog box in MapForce, right-click the mapping and select <strong>Mapping Settings</strong> from the context menu).</td>
</tr>
<tr>
<td><strong>OutputSettings_Encoding</strong></td>
<td>This property is no longer supported. Mapping output encoding settings do no longer exist. Components have individual output encoding settings.</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Read-only. The parent object according to the object model.</td>
</tr>
<tr>
<td><strong>Path</strong></td>
<td>Read-only. Path of the document file without name.</td>
</tr>
<tr>
<td><strong>Saved</strong></td>
<td>Read-only. <strong>True</strong> if the document was not modified since the last save operation, <strong>false</strong> otherwise.</td>
</tr>
</tbody>
</table>

## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Activate</strong></td>
<td>Makes this document the active document.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the document without saving.</td>
</tr>
<tr>
<td>CreateUserDefinedFunction</td>
<td>Creates a user defined function in the current document.</td>
</tr>
<tr>
<td>FindComponentByID</td>
<td>Searches in the whole document, also all its mappings, for the component with the specified id.</td>
</tr>
<tr>
<td>GenerateCHashCode</td>
<td>Generates C# code that will perform the mapping. Uses the properties defined in Application.Options to configure code generation.</td>
</tr>
<tr>
<td>GenerateCodeEx</td>
<td>Generates code that will perform the mapping. The parameter <code>i_nLanguage</code> specifies the target language. The method returns an object that can be used to enumerate all messages created by the code generator. These are the same messages that get displayed in the Messages window of MapForce.</td>
</tr>
<tr>
<td>GenerateCppCode</td>
<td>Generates C++ code that will perform the mapping. Uses the properties defined in Application.Options to configure code generation.</td>
</tr>
<tr>
<td>GenerateJavaCode</td>
<td>Generates Java code that will perform the mapping. Uses the properties defined in Application.Options to configure code generation.</td>
</tr>
<tr>
<td>GenerateOutput</td>
<td>Generates all output files defined in the mapping using a MapForce internal mapping language. The names of the output files are defined as properties of the output items in the mapping.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: This method can only be used when the MapForce (running as a COM server) main window is visible, or is embedded with a graphical user interface. If the method is called while MapForce is not visible, then an error will occur.</td>
</tr>
<tr>
<td>GenerateOutputEx</td>
<td>Generates all output files defined in the mapping using a MapForce internal mapping language. The names of the output files are defined as properties of the output items in the mapping. This method is identical to <code>GenerateOutput</code> except for its return value containing the resulting messages, warnings and errors arranged as trees of <code>AppOutputLines</code>.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: This method can only be used when the MapForce (running as a COM server) main window is visible, or is embedded with a graphical user interface. If the method is called while MapForce is not visible, then an error will occur.</td>
</tr>
<tr>
<td>GenerateXQuery</td>
<td>Generates mapping code as XQuery. Uses the properties defined in Application.Options to configure code generation.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GenerateXSLT</td>
<td>Generates mapping code as XSLT. Uses the properties defined in Application.Options to configure code generation.</td>
</tr>
<tr>
<td>GenerateXSLT2</td>
<td>Generates mapping code as XSLT2. Uses the properties defined in Application.Options to configure code generation.</td>
</tr>
<tr>
<td>HighlightSerializedMarker</td>
<td>Use this method to highlight a location in a mapping file that has been previously serialized. If the corresponding document is not already loaded, it will be loaded first. See GenerateCodeEx for a method to retrieve a serialized marker.</td>
</tr>
<tr>
<td>Save</td>
<td>Saves the document to the file defined by Document.FullName.</td>
</tr>
<tr>
<td>SaveAs</td>
<td>Saves the document to the specified file name, and sets Document.FullName to this value if the save operation was successful.</td>
</tr>
</tbody>
</table>

**Events**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnDocumentClosed</td>
<td>This event is triggered when a document is closed. The document object passed into the event handler should not be accessed. The corresponding open event is Application.OnDocumentOpened.</td>
</tr>
<tr>
<td>OnModifiedFlagChanged</td>
<td>This event is triggered when a document's modification status changes.</td>
</tr>
</tbody>
</table>

**17.2.1.9.1 Properties**

**17.2.1.9.1.1 Application**

Retrieves the application's top-level object.

**Signature**

| Application : Application |
Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.9.1.2  **FullName**

Path and name of the document file.

**Signature**

```java
FullName : String
```

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.9.1.3  **JavaSettings_BasePackageName**

Sets or retrieves the base package name used when generating Java code. In the MapForce graphical user interface, this setting is available in the **Mapping Settings** dialog box (right-click the mapping and select **Mapping Settings** from the context menu).

**Signature**

```java
JavaSettings_BasePackageName : String
```

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.9.1.4  MainMapping

Retrieves the main mapping of the document.

**Signature**

MainMapping : Mapping

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.9.1.5  MapForceView

This property gives access to functionality specific to the MapForce view.

**Signature**

MapForceView : MapForceView

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.9.1.6  Mappings

Returns a collection of the mappings contained in the document.

**Signature**

Mappings : Mappings

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
</tbody>
</table>
### Error code

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

#### 17.2.1.9.1.7 Name

Name of the document file without file path.

**Signature**

```plaintext
Name : String
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

#### 17.2.1.9.1.8 OutputSettings_ApplicationName

Sets or retrieves the application name available in the Mapping Settings dialog box (To display this dialog box in MapForce, right-click the mapping and select Mapping Settings from the context menu).

**Signature**

```plaintext
OutputSettings_ApplicationName : String
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

#### 17.2.1.9.1.9 OutputSettings_Encoding (obsolete)

This property is no longer supported. Mapping output encoding settings do no longer exist. Components have individual output encoding settings.

**Signature**

```plaintext
OutputSettings_Encoding : String
```
17.2.1.9.1.10  Parent

The parent object according to the object model.

**Signature**

| Parent : Documents |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.9.1.11  Path

Path of the document file without name.

**Signature**

| Path : String |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.9.1.12  Saved

*True* if the document was not modified since the last save operation, *false* otherwise.

**Signature**

| Saved : Boolean |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

**17.2.1.9.2 Methods**

**17.2.1.9.2.1 Activate**

Makes this document the active document.

**Signature**

```csharp
Activate() : Void
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

**17.2.1.9.2.2 Close**

Closes the document without saving.

**Signature**

```csharp
Close() : Void
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

**17.2.1.9.2.3 CreateUserDefinedFunction**

Creates a user defined function in the current document.

**Signature**

```csharp
CreateUserDefinedFunction(in strFunctionName: String, in strLibraryName: String, in strSyntax: String, in strDetails: String, in bInlinedUse: Boolean) : Mapping
```
Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strFunctionName</td>
<td>String</td>
<td>The name of the function.</td>
</tr>
<tr>
<td>strLibraryName</td>
<td>String</td>
<td>The name of the library to which this function belongs.</td>
</tr>
<tr>
<td>strSyntax</td>
<td>String</td>
<td>A string that describes the syntax of this function (this is for information purpose only).</td>
</tr>
<tr>
<td>strDetails</td>
<td>String</td>
<td>A description of this function.</td>
</tr>
<tr>
<td>bInlinedUse</td>
<td>Boolean</td>
<td>Boolean flag that specifies if the function has inlined use.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1208</td>
<td>Failed creating user-defined function.</td>
</tr>
<tr>
<td>1209</td>
<td>Changing the document not allowed. It is read-only.</td>
</tr>
</tbody>
</table>

17.2.1.9.2.4  FindComponentByID

Searches in the whole document, also all its mappings, for the component with the specified id.

Signature

FindComponentByID(in nID:Unsigned Long) : Component

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nID</td>
<td>Unsigned Long</td>
<td>The ID of the component to search for.</td>
</tr>
</tbody>
</table>
Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.9.2.5  GenerateCHashCode

Generates C# code that will perform the mapping. Uses the properties defined in Application.Options to configure code generation.

Signature

GenerateCHashCode() : Void

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1205</td>
<td>Error during code generation.</td>
</tr>
</tbody>
</table>

17.2.1.9.2.6  GenerateCodeEx

Generates code that will perform the mapping. The parameter i_nLanguage specifies the target language. The method returns an object that can be used to enumerate all messages created by the code generator. These are the same messages that get displayed in the Messages window of MapForce.

Signature

GenerateCodeEx(in i_nLanguage:ENUMProgrammingLanguage) : ErrorMarkers

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_nLanguage</td>
<td>ENUMProgrammingLanguage</td>
<td>Specifies the target code generation language.</td>
</tr>
</tbody>
</table>
Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1205</td>
<td>Error during code generation.</td>
</tr>
</tbody>
</table>

17.2.1.9.2.7  GenerateCppCode

Generates C++ code that will perform the mapping. Uses the properties defined in Application.Options to configure code generation.

Signature

```cpp
GenerateCppCode() : Void
```

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1205</td>
<td>Error during code generation.</td>
</tr>
</tbody>
</table>

17.2.1.9.2.8  GenerateJavaCode

Generates Java code that will perform the mapping. Uses the properties defined in Application.Options to configure code generation.

Signature

```java
GenerateJavaCode() : Void
```

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1205</td>
<td>Error during code generation.</td>
</tr>
</tbody>
</table>
17.2.1.9.2.9  GenerateOutput

Generates all output files defined in the mapping using a MapForce internal mapping language. The names of the output files are defined as properties of the output items in the mapping.

**Note**: This method can only be used when the MapForce (running as a COM server) main window is visible, or is embedded with a graphical user interface. If the method is called while MapForce is not visible, then an error will occur.

**Signature**

```
GenerateOutput() : Void
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1206</td>
<td>Error during execution of mapping algorithm.</td>
</tr>
<tr>
<td>1210</td>
<td>Generating output is only supported when the graphical user interface is visible.</td>
</tr>
</tbody>
</table>

17.2.1.9.2.10  GenerateOutputEx

Generates all output files defined in the mapping using a MapForce internal mapping language. The names of the output files are defined as properties of the output items in the mapping. This method is identical to GenerateOutput except for its return value containing the resulting messages, warnings and errors arranged as trees of AppOutputLines.

**Note**: This method can only be used when the MapForce (running as a COM server) main window is visible, or is embedded with a graphical user interface. If the method is called while MapForce is not visible, then an error will occur.

**Signature**

```
GenerateOutputEx() : AppOutputLines
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
### Error code Description

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1206</td>
<td>Error during execution of mapping algorithm.</td>
</tr>
<tr>
<td>1210</td>
<td>Generating output is only supported when the graphical user interface is visible.</td>
</tr>
</tbody>
</table>

#### 17.2.1.9.2.11  GenerateXQuery

Generates mapping code as XQuery. Uses the properties defined in `Application.Options` to configure code generation.

**Signature**

```
GenerateXQuery() : Void
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1204</td>
<td>Error during XSLT/XSLT2/XQuery code generation.</td>
</tr>
</tbody>
</table>

#### 17.2.1.9.2.12  GenerateXSLT

Generates mapping code as XSLT. Uses the properties defined in `Application.Options` to configure code generation.

**Signature**

```
GenerateXSLT() : Void
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1204</td>
<td>Error during XSLT/XSLT2/XQuery code generation.</td>
</tr>
</tbody>
</table>
17.2.1.9.2.13  GenerateXSLT2

Generates mapping code as XSLT2. Uses the properties defined in Application.Options to configure code generation.

**Signature**

```
GenerateXSLT2() : Void
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1204</td>
<td>Error during XSLT/XSLT2/XQuery code generation.</td>
</tr>
</tbody>
</table>

17.2.1.9.2.14  HighlightSerializedMarker

Use this method to highlight a location in a mapping file that has been previously serialized. If the corresponding document is not already loaded, it will be loaded first. See GenerateCodeEx for a method to retrieve a serialized marker.

**Signature**

```
HighlightSerializedMarker(in i_strSerializedMarker:String) : Void
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_strSerializedMarker</td>
<td>String</td>
<td>The ErrorMarker object to highlight. Use ErrorMaker.Serialized to obtain this value.</td>
</tr>
</tbody>
</table>

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1001</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1007</td>
<td>The string passed in i_strSerializedMarker is not recognized a</td>
</tr>
</tbody>
</table>
### 17.2.1.9.2.15  Save

Saves the document to the file defined by `Document.FullName`.

**Signature**

```plaintext
Save() : Void
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1008</td>
<td>The marker points to a location that is no longer valid.</td>
</tr>
</tbody>
</table>

### 17.2.1.9.2.16  SaveAs

Saves the document to the specified file name, and sets `Document.FullName` to this value if the save operation was successful.

**Signature**

```plaintext
SaveAs(in i_strFileName:String) : Void
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_strFileName</td>
<td>String</td>
<td>Specifies the path where to save the document.</td>
</tr>
</tbody>
</table>

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.9.3 Events

17.2.1.9.3.1 OnDocumentClosed

This event is triggered when a document is closed. The document object passed into the event handler should not be accessed. The corresponding open event is Application.OnDocumentOpened.

**Signature**

```csharp
OnDocumentClosed(in Document): Void
```

17.2.1.9.3.2 OnModifiedFlagChanged

This event is triggered when a document's modification status changes.

**Signature**

```csharp
OnModifiedFlagChanged(in Boolean): Void
```

17.2.1.10 Documents

Represents a collection of Document objects.

Properties to navigate the object model:

- Application
- Parent

Open and create mappings:

- OpenDocument
- NewDocument

Iterating through the collection:

- Count
- Item
- ActiveDocument

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveDocument</td>
<td>Read-only. Retrieves the active document. If no document is open,</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Application</td>
<td>Read-only. Retrieves the application's top-level object.</td>
</tr>
<tr>
<td>Count</td>
<td>Read-only. Retrieves the number of documents in the collection.</td>
</tr>
<tr>
<td>Item</td>
<td>Read-only. Retrieves the document at index $n$ from the collection. Indices start with 1.</td>
</tr>
<tr>
<td>Parent</td>
<td>Read-only. The parent object according to the object model.</td>
</tr>
</tbody>
</table>

## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewDocument</td>
<td>Creates a new document, adds it to the end of the collection, and makes it the active document.</td>
</tr>
<tr>
<td>OpenDocument</td>
<td>Opens an existing mapping document (*.mfd). Adds the newly opened document to the end of the collection and makes it the active document.</td>
</tr>
</tbody>
</table>

### 17.2.1.10.1 Properties

#### 17.2.1.10.1.1 ActiveDocument

Retrieves the active document. If no document is open, null is returned.

**Signature**

```plaintext
ActiveDocument : Document
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1600</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1601</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.10.1.2  Application

Retrieves the application's top-level object.

**Signature**

```plaintext
Application : Application
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1600</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1601</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.10.1.3  Count

Retrieves the number of documents in the collection.

**Signature**

```plaintext
Count : Integer
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1600</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1601</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.10.1.4  Item

Retrieves the document at index \( n \) from the collection. Indices start with 1.

**Signature**

```plaintext
Item this(in n:Integer) : Document
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1600</td>
<td>The object is no longer valid.</td>
</tr>
</tbody>
</table>
### Error code Description

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1601</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

#### 17.2.10.1.5 Parent

The parent object according to the object model.

**Signature**

Parent : Application

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1600</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1601</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

#### 17.2.10.2 Methods

#### 17.2.10.2.1 NewDocument

Creates a new document, adds it to the end of the collection, and makes it the active document.

**Signature**

NewDocument() : Document

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1600</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1601</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

#### 17.2.10.2.2 OpenDocument

Opens an existing mapping document (*.mfd). Adds the newly opened document to the end of the collection and makes it the active document.
Signature

OpenDocument(in strPath:String) : Document

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strPath</td>
<td>String</td>
<td>The path of the mapping file.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1600</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1601</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.11 ErrorMarker

Represents a simple message line. Unlike AppOutputLine objects, error markers do not have a hierarchical structure.

Properties to navigate the object model:

- Application
- Parent

Access to message information:

- DocumentFileName
- ErrorLevel
- Highlight
- Serialization
- Text

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Read-only. Retrieves the application's top-level object.</td>
</tr>
<tr>
<td>DocumentFileName</td>
<td>Read-only. Retrieves the name of the mapping file that the error marker is associated with.</td>
</tr>
<tr>
<td>ErrorLevel</td>
<td>Read-only. Retrieves the severity of the error.</td>
</tr>
</tbody>
</table>
### Properties

#### Application

Retrieves the application's top-level object.

**Signature**

```application
Application : Application
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1900</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1901</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.11.1.2  DocumentFileName

Retrieves the name of the mapping file that the error marker is associated with.

**Signature**

| DocumentFileName : String |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1900</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1901</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.11.1.3  ErrorLevel

Retrieves the severity of the error.

**Signature**

| ErrorLevel : ENUMCodeGenErrorLevel |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1900</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1901</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.11.1.4  Parent

The parent object according to the object model.

**Signature**

| Parent : ErrorMarkers |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1900</td>
<td>The object is no longer valid.</td>
</tr>
</tbody>
</table>
### 17.2.11.1.5 Serialization

Serialize error marker into a string. Use this string in calls to `Application.HighlightSerializedMarker` or `Document.HighlightSerializedMarker` to highlight the marked item in the mapping. The string can be persisted and used in other instantiations of MapForce or its Control.

#### Signature

<table>
<thead>
<tr>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serialization : String</td>
</tr>
</tbody>
</table>

#### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1900</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1901</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

### 17.2.11.1.6 Text

Retrieves the message text.

#### Signature

<table>
<thead>
<tr>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text : String</td>
</tr>
</tbody>
</table>

#### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1900</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1901</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

### 17.2.11.2 Methods

#### 17.2.11.2.1 Highlight

Highlights the item that the error marker is associated with. If the corresponding document is not open, it will be opened.
Signature

Highlight() : Void

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1900</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1901</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1008</td>
<td>The marker points to a location that is no longer valid.</td>
</tr>
</tbody>
</table>

17.2.1.12 ErrorMarkers

Represents a collection of ErrorMarker objects.

Properties to navigate the object model:

- Application
- Parent

Iterating through the collection:

- Count
- Item

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Read-only. Retrieves the application's top-level object.</td>
</tr>
<tr>
<td>Count</td>
<td>Read-only. Retrieves the number of error markers in the collection.</td>
</tr>
<tr>
<td>Item</td>
<td>Read-only. Retrieves the error marker at index n from the collection. Indices start with 1.</td>
</tr>
<tr>
<td>Parent</td>
<td>Read-only. The parent object according to the object model.</td>
</tr>
</tbody>
</table>
17.2.12.1 Properties

17.2.12.1.1 Application

Retrieves the application’s top-level object.

**Signature**

| Application | Application |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1800</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1801</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.12.1.2 Count

Retrieves the number of error markers in the collection.

**Signature**

| Count     | Integer |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1800</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1801</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.12.1.3 Item

Retrieves the error marker at index $n$ from the collection. Indices start with 1.

**Signature**

| Item this(in n:Integer) | ErrorMarker |
Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1800</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1801</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.12.1.4  Parent

The parent object according to the object model.

Signature

| Parent : Application |

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1800</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1801</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.13  MapForceView

Represents the current view in the MapForce Mapping tab for a document. A document has exactly one MapForceView which displays the currently active mapping.

Properties to navigate the object model:

- Application
- Parent

View activation and view properties:

- Active
- ShowItemTypes
- ShowLibraryInFunctionHeader
- HighlightMyConnections
- HighlightMyConnectionsRecursively

Mapping related properties:

- ActiveMapping
- ActiveMappingName

Adding items:
- InsertWSDLCall
- InsertXMLFile
- InsertXMLSchema
- InsertXMLSchemaWithSample

## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong></td>
<td>Use this property to query if the mapping view is the active view, or set this view to be the active one.</td>
</tr>
<tr>
<td><strong>ActiveMapping</strong></td>
<td>Gets or sets the currently active mapping in the document this MapForceView belongs to.</td>
</tr>
<tr>
<td><strong>ActiveMappingName</strong></td>
<td>Gets or sets the currently active mapping by name in the document this MapForceView belongs to.</td>
</tr>
<tr>
<td><strong>Application</strong></td>
<td>Read-only. Retrieves the application's top-level object.</td>
</tr>
<tr>
<td><strong>HighlightMyConnections</strong></td>
<td>This property defines whether connections from the selected item only should be highlighted.</td>
</tr>
<tr>
<td><strong>HighlightMyConnectionsRecursively</strong></td>
<td>This property defines if only the connections coming directly or indirectly from the selected item should be highlighted.</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Read-only. The parent object according to the object model.</td>
</tr>
<tr>
<td><strong>ShowItemTypes</strong></td>
<td>This property defines if types of items should be shown in the mapping diagram.</td>
</tr>
<tr>
<td><strong>ShowLibraryInFunctionHeader</strong></td>
<td>This property defines whether the name of the function library should be part of function names.</td>
</tr>
</tbody>
</table>

## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>InsertWSDLCall</strong></td>
<td>Adds a new WSDL call component to the mapping.</td>
</tr>
<tr>
<td><strong>InsertXMLFile</strong></td>
<td>MapForceView.InsertXMLFile is obsolete. Use Mapping.InsertXMLFile instead.</td>
</tr>
<tr>
<td><strong>InsertXMLSchema</strong></td>
<td>MapForceView.InsertXMLSchema is obsolete. Use Mapping.InsertXMLSchema instead.</td>
</tr>
<tr>
<td><strong>InsertXMLSchemaWithSample</strong></td>
<td>MapForceView.InsertXMLSchemaWithSample is obsolete. Use Mapping.InsertXMLFile instead. Notice, Mapping.InsertXMLFile does not require a parameter for passing the root element. The root</td>
</tr>
</tbody>
</table>
17.2.1.13.1  Properties

17.2.1.13.1.1  Active

Use this property to query if the mapping view is the active view, or set this view to be the active one.

**Signature**

<table>
<thead>
<tr>
<th>Active : Boolean</th>
</tr>
</thead>
</table>

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1300</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1301</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.13.1.2  ActiveMapping

Gets or sets the currently active mapping in the document this MapForceView belongs to.

**Signature**

<table>
<thead>
<tr>
<th>ActiveMapping : Mapping</th>
</tr>
</thead>
</table>

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1300</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1301</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.13.1.3  ActiveMappingName

Gets or sets the currently active mapping by name in the document this MapForceView belongs to.
**Signature**

```
ActiveMappingName : String
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1300</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1301</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

### 17.2.13.1.4 Application

Retrieves the application's top-level object.

**Signature**

```
Application : Application
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1300</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1301</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

### 17.2.13.1.5 HighlightMyConnections

This property defines whether connections from the selected item only should be highlighted.

**Signature**

```
HighlightMyConnections : Boolean
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1300</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1301</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

© 2019 Altova Gmbh

Altova MapForce 2019 Enterprise Edition
17.2.13.1.6  **HighlightMyConnectionsRecursively**

This property defines if only the connections coming directly or indirectly from the selected item should be highlighted.

**Signature**

<table>
<thead>
<tr>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>HighlightMyConnectionsRecursively : Boolean</td>
</tr>
</tbody>
</table>

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1300</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1301</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.13.1.7  **Parent**

The parent object according to the object model.

**Signature**

<table>
<thead>
<tr>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent : Document</td>
</tr>
</tbody>
</table>

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1300</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1301</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.13.1.8  **ShowItemTypes**

This property defines if types of items should be shown in the mapping diagram.

**Signature**

<table>
<thead>
<tr>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShowItemTypes : Boolean</td>
</tr>
</tbody>
</table>
Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1300</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1301</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.13.1.9  ShowLibraryInFunctionHeader

This property defines whether the name of the function library should be part of function names.

Signature

```plaintext
ShowLibraryInFunctionHeader : Boolean
```

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1300</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1301</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.13.2  Methods

17.2.13.2.1  InsertWSDLCall

Adds a new WSDL call component to the mapping.

Signature

```plaintext
InsertWSDLCall(in i_strWSDLFileName:String) : Void
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_strWSDLFileName</td>
<td>String</td>
<td>Specifies the path of the WSDL file to add to the mapping.</td>
</tr>
</tbody>
</table>
Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1300</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1301</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.13.2.2  InsertXMLFile (obsolete)


Signature

```php
InsertXMLFile(in i_strFileName:String, in i_strXMLRootName:String) : Void
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_strFileName</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>i_strXMLRootName</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

17.2.1.13.2.3  InsertXMLSchema (obsolete)


Signature

```php
InsertXMLSchema(in i_strSchemaFileName:String, in i_strXMLRootName:String) : Void
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_strSchemaFileName</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>i_strXMLRootName</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

17.2.1.13.2.4  InsertXMLSchemaWithSample (obsolete)

MapForceView.InsertXMLSchemaWithSample is obsolete. Use Mapping.InsertXMLFile instead. Notice, Mapping.InsertXMLFile does not require a parameter for passing the root element. The
Root element is automatically set as the XML file's root element name.

**Signature**

```
InsertXMLSchemaWithSample(in i_strSchemaFileName:String, in i_strXMLExampleFile:String, in i_strXMLRootName:String) : Void
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_strSchemaFileName</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>i_strXMLExampleFile</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>i_strXMLRootName</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

### 17.2.1.14 Mapping

A Mapping object represents a mapping in a document, either the main mapping, or a local user-defined-function mapping.

Properties to navigate the object model:

- Application
- Parent

Mapping properties:

- IsMainMapping
- Name

Components in the mapping:

- Components

Adding items:

- CreateConnection
- InsertFunctionCall
- InsertXMLFile
- InsertXMLSchema
- InsertXMLSchemaInputParameter
- InsertXMLSchemaOutputParameter

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Read-only. Retrieves the application's top-level object.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Components</strong></td>
<td>Read-only. Returns a collection of all components in the current mapping.</td>
</tr>
<tr>
<td><strong>IsMainMapping</strong></td>
<td>Read-only. Indicates if the current mapping is the main mapping of the document the mapping is in. True means it is the main mapping. False means it is a user-defined function (UDF).</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>Read-only. The name of the mapping or user defined-function (UDF).</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Read-only. The parent object according to the object model.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CreateConnection</strong></td>
<td>Creates a connection between the two supplied datapoints (DatapointFrom &amp; DatapointTo). It will fail to do so if the DatapointFrom is not an output-side datapoint, the DatapointTo is not an input-side datapoint, or a connection between these two datapoints already exists.</td>
</tr>
<tr>
<td><strong>InsertFunctionCall</strong></td>
<td>Inserts a function call component into the current mapping. The specified library and function names indicate the function or user-defined function to be called.</td>
</tr>
<tr>
<td><strong>InsertXMLFile</strong></td>
<td>Adds a new XML schema component to the mapping. The component's internal structure is determined by the schema referenced in the specified XML file (i_strFileName) or, if the XML file does not reference a schema file, by the separately specified schema file (i_strSchemaFileName). If the XML file has a schema file reference, then the parameter i_strSchemaFileName is ignored. The root element of the XML file will be used in the component. The specified XML file is used as the input sample to evaluate the mapping.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>InsertXMLSchema</td>
<td>Adds a new XML schema component to the mapping. The component’s internal structure is determined by the schema file specified in the first parameter. The second parameter defines the root element of this schema if there is more than one candidate. If the passed root element is an empty string and more candidates are available, a Select Root Element dialog box will pop up if MapForce is visible. If MapForce is invisible, no dialog box will pop up and only an error is returned. No XML input sample is assigned to this component.</td>
</tr>
<tr>
<td>InsertXMLSchemaInputParameter</td>
<td>Inserts an XML schema input parameter component into the current mapping. The current mapping has to be a user-defined function. Trying to insert it (the schema input parameter) into the main mapping will fail.</td>
</tr>
<tr>
<td>InsertXMLSchemaOutputParameter</td>
<td>Inserts an XML schema output parameter component into the current mapping. The current mapping has to be a user-defined function. Trying to insert it (the schema output parameter) into the main mapping will fail.</td>
</tr>
</tbody>
</table>

17.2.1.14.1 Properties

17.2.1.14.1.1 Application

Retrieves the application’s top-level object.

Signature

```
Application : Application
```

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.14.1.2  Components

Returns a collection of all components in the current mapping.

**Signature**

```
Components : Components
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.14.1.3  IsMainMapping

Indicates if the current mapping is the main mapping of the document the mapping is in.

- **True** means it is the main mapping.
- **False** means it is a user-defined function (UDF).

**Signature**

```
IsMainMapping : Boolean
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.14.1.4  Name

The name of the mapping or user defined-function (UDF).

**Signature**

```
Name : String
```
Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.14.1.5 Parent

The parent object according to the object model.

Signature

Parent : Document

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.14.2 Methods

17.2.14.2.1 CreateConnection

Creates a connection between the two supplied datapoints (DatapointFrom & DatapointTo).

It will fail to do so if the DatapointFrom is not an output-side datapoint, the DatapointTo is not an input-side datapoint, or a connection between these two datapoints already exists.

Signature

CreateConnection(in DatapointFrom:Datapoint, in DatapointTo:Datapoint) : Connection

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DatapointFrom</td>
<td>Datapoint</td>
<td>The datapoint from which the connection is to be created.</td>
</tr>
</tbody>
</table>
### DatapointTo

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DatapointTo</td>
<td>Datapoint</td>
<td>The destination datapoint.</td>
</tr>
</tbody>
</table>

### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1240</td>
<td>Changing the document not allowed. It is read-only.</td>
</tr>
<tr>
<td>1241</td>
<td>Failed creating the connection.</td>
</tr>
</tbody>
</table>

### 17.2.14.2.2 InsertFunctionCall

Inserts a function call component into the current mapping.

The specified library and function names indicate the function or user-defined function to be called.

#### Signature

```csharp
InsertFunctionCall(in strFunctionName:String, in strLibraryName:String) : Component
```

#### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strFunctionName</td>
<td>String</td>
<td>The name of the function to be inserted.</td>
</tr>
<tr>
<td>strLibraryName</td>
<td>String</td>
<td>The library name of the function to be inserted.</td>
</tr>
</tbody>
</table>

### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1240</td>
<td>Changing the document not allowed. It is read-only.</td>
</tr>
<tr>
<td>1242</td>
<td>Failed creating function call component.</td>
</tr>
</tbody>
</table>
17.2.1.14.2.3  InsertXMLFile

Adds a new XML schema component to the mapping.

The component's internal structure is determined by the schema referenced in the specified XML
file (i_strFileName) or, if the XML file does not reference a schema file, by the separately
specified schema file (i_strSchemaFileName).

If the XML file has a schema file reference, then the parameter i_strSchemaFileName is ignored.

The root element of the XML file will be used in the component.

The specified XML file is used as the input sample to evaluate the mapping.

Signature

```
InsertXMLFile(in i_strFileName:String, in i_strSchemaFileName:String) : Component
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_strFileName</td>
<td>String</td>
<td>The path of the instance XML file to add.</td>
</tr>
<tr>
<td>i_strSchemaFileName</td>
<td>String</td>
<td>The path of the XML Schema Definition file to add.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1240</td>
<td>Changing the document not allowed. It is read-only.</td>
</tr>
<tr>
<td>1244</td>
<td>Failed creating component.</td>
</tr>
</tbody>
</table>

17.2.1.14.2.4  InsertXMLSchema

Adds a new XML schema component to the mapping.

The component's internal structure is determined by the schema file specified in the first
parameter.

The second parameter defines the root element of this schema if there is more than one
candidate.

If the passed root element is an empty string and more candidates are available, a **Select Root Element** dialog box will pop up if MapForce is visible. If MapForce is invisible, no dialog box will pop up and only an error is returned.

No XML input sample is assigned to this component.

**Signature**

```
InsertXMLSchema(in i_strSchemaFileName:String, in i_strXMLRootName:String) : Component
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_strSchemaFileName</td>
<td>String</td>
<td>The path of the XML Schema Definition file to add.</td>
</tr>
<tr>
<td>i_strXMLRootName</td>
<td>String</td>
<td>The root element of the schema (applicable when the schema has more than one root element).</td>
</tr>
</tbody>
</table>

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1240</td>
<td>Changing the document not allowed. It is read-only.</td>
</tr>
<tr>
<td>1244</td>
<td>Failed creating component.</td>
</tr>
</tbody>
</table>

**17.2.14.2.5 InsertXMLSchemaInputParameter**

Inserts an XML schema input parameter component into the current mapping.

The current mapping has to be a user-defined function. Trying to insert it (the schema input parameter) into the main mapping will fail.

**Signature**

```
InsertXMLSchemaInputParameter(in strParamName:String, in strSchemaFileName:String, in strXMLRootElementName:String) : Component
```
Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strParamName</td>
<td>String</td>
<td>The name of the input parameter component to create.</td>
</tr>
<tr>
<td>strSchemaFileName</td>
<td>String</td>
<td>The path of the XML Schema Definition file to add.</td>
</tr>
<tr>
<td>strXMLRootElementName</td>
<td>String</td>
<td>The root element of the schema (applicable when the schema has more than one root element). If the passed root element is an empty string and more candidates are available, a Select Root Element dialog will pop up if MapForce is visible. If MapForce is invisible, no dialog box will pop up and only an error is returned.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1240</td>
<td>Changing the document not allowed. It is read-only.</td>
</tr>
<tr>
<td>1243</td>
<td>Failed creating parameter component.</td>
</tr>
<tr>
<td>1245</td>
<td>This operation is not supported for the main mapping.</td>
</tr>
</tbody>
</table>

17.2.1.14.2.6  InsertXMLSchemaOutputParameter

Inserts an XML schema output parameter component into the current mapping.

The current mapping has to be a user-defined function. Trying to insert it (the schema output parameter) into the main mapping will fail.

Signature

```
InsertXMLSchemaOutputParameter(in strParamName:String, in strSchemaFileName:String, in strXMLRootElementName:String) : Component
```
Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strParamName</td>
<td>String</td>
<td>The name of the output parameter component to create.</td>
</tr>
<tr>
<td>strSchemaFileName</td>
<td>String</td>
<td>The path of the XML Schema Definition file to add.</td>
</tr>
<tr>
<td>strXMLRootElementName</td>
<td>String</td>
<td>The root element of the schema (applicable when the schema has more than one root element). If the passed root element is an empty string and more candidates are available, a Select Root Element dialog will pop up if MapForce is visible. If MapForce is invisible, no dialog box will pop up and only an error is returned.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1240</td>
<td>Changing the document not allowed. It is read-only.</td>
</tr>
<tr>
<td>1243</td>
<td>Failed creating parameter component.</td>
</tr>
<tr>
<td>1245</td>
<td>This operation is not supported for the main mapping.</td>
</tr>
</tbody>
</table>

17.2.1.15 Mappings

Represents a collection of Mapping objects.

Properties to navigate the object model:

- Application
- Parent

Iterating through the collection:

- Count
- Item
Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Read-only. Retrieves the application's top-level object.</td>
</tr>
<tr>
<td>Count</td>
<td>Read-only. Retrieves the number of mappings in the collection.</td>
</tr>
<tr>
<td>Item</td>
<td>Read-only. Retrieves the mapping at index <code>n</code> from the collection. Indices start with 1.</td>
</tr>
<tr>
<td>Parent</td>
<td>Read-only. The parent object according to the object model.</td>
</tr>
</tbody>
</table>

17.2.1.15.1 Properties

17.2.1.15.1.1 Application

Retrieves the application's top-level object.

Signature

```
Application : Application
```

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.15.1.2 Count

Retrieves the number of mappings in the collection.

Signature

```
Count : Integer
```
17.2.15.1.3  Item

Retrieves the mapping at index \( n \) from the collection. Indices start with 1.

**Signature**

\[ \text{Item this(in n:Integer) : Mapping} \]

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.15.1.4  Parent

The parent object according to the object model.

**Signature**

\[ \text{Parent : Document} \]

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1201</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.16  Options

This object gives access to all MapForce options available in the **Tools | Options** dialog.

Properties to navigate the object model:
• Application
• Parent

General options:
• ShowLogoOnPrint
• ShowLogoOnStartup
• UseGradientBackground

Options for code generation:
• DefaultOutputEncoding
• DefaultOutputByteOrder
• DefaultOutputByteOrderMark
• XSLTDefaultOutputDirectory
• CodeDefaultOutputDirectory
• CPPSettings_DOMType
• CPPSettings_GenerateVC6ProjectFile
• CppSettings_GenerateVSProjectFile
• CPPSettings_LibraryType
• CPPSettings_UseMFC
• CSharpSettings_ProjectType

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Read-only. Retrives the application's top-level object.</td>
</tr>
<tr>
<td>CPPSettings_DOMType</td>
<td>Specifies the DOM type used by Document.GenerateCppCode.</td>
</tr>
<tr>
<td>CPPSettings_GenerateVC6ProjectFile</td>
<td>Specifies if VisualC++ 6.0 project files should be generated by Document.GenerateCppCode.</td>
</tr>
<tr>
<td>CppSettings_GenerateVSProjectFile</td>
<td>Specifies the version of Visual Studio in which project files should be generated by Document.GenerateCppCode.</td>
</tr>
<tr>
<td>CPPSettings_LibraryType</td>
<td>Specifies the library type used by Document.GenerateCppCode.</td>
</tr>
<tr>
<td>CPPSettings_UseMFC</td>
<td>Specifies if MFC support should be used by C++ code generated by Document.GenerateCppCode.</td>
</tr>
<tr>
<td>CSharpSettings_ProjectType</td>
<td>Specifies the type of C# project used by Document.GenerateCHashCode.</td>
</tr>
</tbody>
</table>
### Properties

#### Application

Retrieves the application's top-level object.

**Signature**

```
Application : Application
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1401</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

#### CodeDefaultOutputDirectory

## CodeDefaultOutputDirectory : String

### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1401</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

### CPPSettings_DOMType

17.2.16.1.3

Specifies the DOM type used by `Document.GenerateCppCode`.

#### Signature

`CPPSettings_DOMType : ENUMDOMType`

#### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1401</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1402</td>
<td>The parameter value is out of range.</td>
</tr>
<tr>
<td>1403</td>
<td>The parameter value is not available anymore.</td>
</tr>
</tbody>
</table>

### CPPSettings_GenerateVC6ProjectFile (obsolete)

17.2.16.1.4

Specifies if VisualC++ 6.0 project files should be generated by `Document.GenerateCppCode`.

#### Signature

`CPPSettings_GenerateVC6ProjectFile : Boolean`

#### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>The object is no longer valid.</td>
</tr>
</tbody>
</table>
### Error Code Descriptions

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1401</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1402</td>
<td>The parameter value is out of range.</td>
</tr>
<tr>
<td>1403</td>
<td>The parameter value is not available anymore.</td>
</tr>
</tbody>
</table>

#### 17.2.16.1.5 CppSettings_GenerateVSProjectFile

Specifies the version of Visual Studio in which project files should be generated by `Document.GenerateCppCode`.

**Signature**

```plaintext
CppSettings_GenerateVSProjectFile : ENUMProjectType
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1401</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1402</td>
<td>The parameter value is out of range.</td>
</tr>
<tr>
<td>1403</td>
<td>The parameter value is not available anymore.</td>
</tr>
</tbody>
</table>

#### 17.2.16.1.6 CPPSettings_LibraryType

Specifies the library type used by `Document.GenerateCppCode`.

**Signature**

```plaintext
CPPSettings_LibraryType : ENUMLibType
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1401</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.16.1.7  **CPPSettings_UseMFC**

Specifies if MFC support should be used by C++ code generated by `Document.GenerateCppCode`.

**Signature**

| CPPSettings_UseMFC : Boolean |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1401</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.16.1.8  **CSharpSettings_ProjectType**

Specifies the type of C# project used by `Document.GenerateCHashCode`.

**Signature**

| CSharpSettings_ProjectType : ENUMProjectType |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1401</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1402</td>
<td>The parameter value is out of range.</td>
</tr>
<tr>
<td>1403</td>
<td>The parameter value is not available anymore.</td>
</tr>
</tbody>
</table>

17.2.16.1.9  **DefaultOutputByteOrder**

Byte order for the file encoding used for output files.

**Signature**

| DefaultOutputByteOrder : String |
### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1401</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

#### 17.2.1.16.1.10 DefaultOutputByteOrderMark

Indicates if a byte order mark (BOM), is to be included in the file encoding of output files.

**Signature**

```java
DefaultOutputByteOrderMark : Boolean
```

### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1401</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

#### 17.2.1.16.1.11 DefaultOutputEncoding

File encoding used for output files.

**Signature**

```java
DefaultOutputEncoding : String
```

### Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1401</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.16.1.12  GenerateWrapperClasses

Indicates if wrapper classes are also to be generated when generating code.

**Signature**

| GenerateWrapperClasses : Boolean |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1401</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.16.1.13  JavaSettings_ApacheAxisVersion (obsolete)

This property is obsolete.

**Signature**

| JavaSettings_ApacheAxisVersion : ENUMApacheAxisVersion |

17.2.16.1.14  Parent

The parent object according to the object model.

**Signature**

| Parent : Application |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1401</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.16.1.15  ShowLogoOnPrint

Show or hide the MapForce logo on printed outputs.

**Signature**

| ShowLogoOnPrint : Boolean |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1401</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.16.1.16  ShowLogoOnStartup

Show or hide the MapForce logo on application startup.

**Signature**

| ShowLogoOnStartup : Boolean |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1401</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.16.1.17  UseGradientBackground

Set or retrieve the background color mode for a mapping window.

**Signature**

| UseGradientBackground : Boolean |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>The object is no longer valid.</td>
</tr>
</tbody>
</table>
### 17.2.16.1.18 XSLTDefaultOutputDirectory

Specifies the target directory where files generated by `Document.GenerateXSLT` are placed.

**Signature**

```
XSLTDefaultOutputDirectory : String
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1401</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

### 17.2.17 Project

A `Project` object represents a project and its tree of project items in MapForce.

**Properties to navigate the object model:**

- Application
- Parent

**File handling:**

- FullName
- Name
- Path
- Saved
- Save
- Close

**Project tree navigation:**

- Count
- Item
- _NewEnum

**Project tree manipulation:**

- AddActiveFile
- AddFile
- InsertWebService (Enterprise edition only)
- CreateFolder
Code generation:

- Output_Folder
- Output_Language
- Output_TextEncoding
- Java_BasePackageName
- GenerateCode
- GenerateCodeEx
- GenerateCodeIn
- GenerateCodeInEx

For examples of how to use the properties and methods listed above, see Example: Project Support. For operations with Web services, the MapForce Enterprise edition is required.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_NewEnum</td>
<td>Read-only. This property supports language-specific standard enumeration.</td>
</tr>
<tr>
<td>Application</td>
<td>Read-only. Retrieves the top-level application object.</td>
</tr>
<tr>
<td>Count</td>
<td>Read-only. Retrieves number of children of the project's root item.</td>
</tr>
<tr>
<td>FullName</td>
<td>Path and name of the project file.</td>
</tr>
<tr>
<td>Item</td>
<td>Read-only. Returns the child at n position of the project's root. The index is zero-based. The largest valid index is Count-1. For an alternative to visit all children, see _NewEnum.</td>
</tr>
<tr>
<td>Java_BasePackageName</td>
<td>Sets or gets the base package name of the Java packages that will be generated. This property is used only when generating Java code.</td>
</tr>
<tr>
<td>Name</td>
<td>Read-only. Name of the project file without file path.</td>
</tr>
<tr>
<td>Output_Folder</td>
<td>Sets or gets the default output folder used with GenerateCode and GenerateCodeIn. Project items can overwrite this value in their CodeGenSettings_OutputFolder property, when CodeGenSettings_UseDefault is set to false.</td>
</tr>
<tr>
<td>Output_Language</td>
<td>Sets or gets the default language for code generation when using GenerateCode. Project items can overwrite this value in their CodeGenSettings_OutputLanguage property, when CodeGenSettings_UseDefault is set to</td>
</tr>
</tbody>
</table>
### The MapForce API

#### Name Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output_TextEncoding</td>
<td>Sets or gets the text encoding used when generating XML-based code.</td>
</tr>
<tr>
<td>Parent</td>
<td>Read-only. The parent object according to the object model.</td>
</tr>
<tr>
<td>Path</td>
<td>Read-only. Path of the project file without name.</td>
</tr>
<tr>
<td>Saved</td>
<td>Read-only. <strong>True</strong> if the project was not modified since the last <strong>Save</strong> operation, <strong>false</strong> otherwise.</td>
</tr>
</tbody>
</table>

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddActiveFile</td>
<td>Adds the currently open document to the mapping folder of the project's root.</td>
</tr>
<tr>
<td>AddFile</td>
<td>Adds the specified document to the mapping folder of the project's root.</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the project without saving.</td>
</tr>
<tr>
<td>CreateFolder</td>
<td>Creates a new folder as a child of the project's root item.</td>
</tr>
<tr>
<td>GenerateCode</td>
<td>Generates code for all project items of the project. The code language and output location is determined by properties of the project and project items.</td>
</tr>
<tr>
<td>GenerateCodeEx</td>
<td>Generates code for all project items of the project. The code language and output location are determined by properties of the project and project items. An object that can be used to iterate through all messages issued by the code generation process is returned. These messages are the same as those shown in the Messages window of MapForce.</td>
</tr>
<tr>
<td>GenerateCodeIn</td>
<td>Generates code for all project items of the project in the specified language. The output location is determined by properties of the project and project items.</td>
</tr>
<tr>
<td>GenerateCodeInEx</td>
<td>Generates code for all project items of the project in the specified language. The output location is determined by properties of the project and project items. Returns an object that can be used to iterate through all messages issued by the code generation process. These messages are the same as those shown in the Messages window of MapForce.</td>
</tr>
</tbody>
</table>
Name | Description
---|---
InsertWebService | Inserts a new Web service project into the project's Web service folder. If \_bGenerateMappings is true, initial mapping documents for all ports get generated automatically.
Save | Saves the project to the file defined by FullName.

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnProjectClosed</td>
<td>This event is triggered when the project is closed. The project object passed into the event handler should not be accessed. The corresponding open event is Application.OnProjectOpened.</td>
</tr>
</tbody>
</table>

17.2.1.17.1 Properties

17.2.1.17.1._NewEnum

This property supports language-specific standard enumeration.

Signature

_NewEnum : IUnknown

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
</tbody>
</table>

Examples

```javascript
// JScript sample - enumeration of a project's project items.
function AllChildrenOfProjectRoot()
{
    objProject = objMapForce.ActiveProject;
    if (objProject != null)
    {
        for (objProjectIter = new Enumerator(objProject); !objProjectIter.atEnd(); objProjectIter.moveNext())
        {
            objProjectItem = objProjectIter.item();
```
// JScript sample - iterate all project items, depth first.
function IterateProjectItemsRec(objProjectItemIter)
{
    while (!objProjectItemIter.atEnd())
    {
        objProjectItem = objProjectItemIter.item();
        // do something with project item here
        IterateProjectItemsRec( new Enumerator(objProjectItem) );
        objProjectItemIter.moveToNext();
    }
}

function IterateAllProjectItems()
{
    objProject = objMapForce.ActiveProject;
    if ( objProject != null )
    {
        IterateProjectItemsRec( new Enumerator(objProject) );
    }
}

17.2.17.1.2  Application

Retrieves the top-level application object.

Signature

Application : Application

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1501</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.17.1.3 Count

Retrieves number of children of the project's root item. For examples, see Item or _NewEnum

**Signature**

| Count : Integer |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
</tbody>
</table>

17.2.1.17.1.4 FullName

Path and name of the project file.

**Signature**

| FullName : String |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1501</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.17.1.5 Item

Returns the child at n position of the project's root. The index is zero-based. The largest valid index is Count-1. For an alternative to visit all children, see _NewEnum.

**Signature**

| Item this(in n:Integer) : ProjectItem |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
</tbody>
</table>
Examples

```javascript
// ------------------------------------------
// JScript code snippet - enumerate children using Count and Item.
for( nItemIndex = 0; nItemIndex < objProject.Count; nItemIndex++ )
{
  objProjectItem = objProject.Item(nItemIndex);
  // do something with project item here
}
```

17.2.1.17.1.6  *Java_BasePackageName*

Sets or gets the base package name of the Java packages that will be generated. This property is used only when generating Java code.

**Signature**

```
Java_BasePackageName : String
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1501</td>
<td>Invalid package name specified. Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.17.1.7  *Name*

Name of the project file without file path.

**Signature**

```
Name : String
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1501</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.17.1.8  **Output_Folder**

Sets or gets the default output folder used with `GenerateCode` and `GenerateCodeIn`. Project items can overwrite this value in their `CodeGenSettings_OutputFolder` property, when `CodeGenSettings_UseDefault` is set to false.

**Signature**

```
Output_Folder : String
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1501</td>
<td>Invalid folder name specified. Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.17.1.9  **Output_Language**

Sets or gets the default language for code generation when using `GenerateCode`. Project items can overwrite this value in their `CodeGenSettings_OutputLanguage` property, when `CodeGenSettings_UseDefault` is set to false.

**Signature**

```
Output_Language : ENUMProgrammingLanguage
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1501</td>
<td>Invalid language specified. Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.17.1.10  **Output_TextEncoding**

Sets or gets the text encoding used when generating XML-based code.

**Signature**

```
Output_TextEncoding : String
```
Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1501</td>
<td>Invalid text encoding specified. Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.17.1.11  Parent

The parent object according to the object model.

Signature

Parent : Application

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1501</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.17.1.12  Path

Path of the project file without name.

Signature

Path : String

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1501</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
17.2.1.17.1.13  Saved

True if the project was not modified since the last Save operation, false otherwise.

**Signature**

| Saved : Boolean |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1501</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.17.2  Methods

17.2.1.17.2.1  AddActiveFile

Adds the currently open document to the mapping folder of the project's root.

**Signature**

| AddActiveFile() : ProjectItem |

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1501</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1503</td>
<td>No active document is available.</td>
</tr>
<tr>
<td>1504</td>
<td>Active documents needs to be given a path name before it can be added to the project.</td>
</tr>
<tr>
<td>1705</td>
<td>Mapping could not be assigned to project. Maybe it is already contained in the target folder.</td>
</tr>
</tbody>
</table>
17.2.1.17.2.2 AddFile

Adds the specified document to the mapping folder of the project's root.

**Signature**

```plaintext
AddFile(in i_strFileName:String) : ProjectItem
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_strFileName</td>
<td>String</td>
<td>Specifies the path of the document to add.</td>
</tr>
</tbody>
</table>

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1501</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1705</td>
<td>Mapping could not be assigned to project. The file does not exist or is not a MapForce mapping. Maybe the file is already assigned to the target folder.</td>
</tr>
</tbody>
</table>

17.2.1.17.2.3 Close

Closes the project without saving.

**Signature**

```plaintext
Close() : Void
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
</tbody>
</table>
17.2.1.17.2.4   CreateFolder

Creates a new folder as a child of the project's root item.

**Signature**

CreateFolder(in i_strFolderName:String) : ProjectItem

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_strFolderName</td>
<td>String</td>
<td>The name of the folder to create.</td>
</tr>
</tbody>
</table>

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1501</td>
<td>Invalid folder name or invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.17.2.5   GenerateCode

Generates code for all project items of the project. The code language and output location is determined by properties of the project and project items.

**Signature**

GenerateCode() : Void

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1706</td>
<td>Error during code generation.</td>
</tr>
</tbody>
</table>

17.2.1.17.2.6   GenerateCodeEx

Generates code for all project items of the project. The code language and output location are determined by properties of the project and project items. An object that can be used to iterate through all messages issued by the code generation process is returned. These messages are the same as those shown in the Messages window of MapForce.
Signature

GenerateCodeEx() : ErrorMarkers

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1501</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1706</td>
<td>Error during code generation.</td>
</tr>
</tbody>
</table>

17.2.1.17.2.7 GenerateCodeIn

Generates code for all project items of the project in the specified language. The output location is determined by properties of the project and project items.

Signature

GenerateCodeIn(in i_nLanguage:ENUMProgrammingLanguage) : Void

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_nLanguage</td>
<td>ENUMProgrammingLanguage</td>
<td>Specifies the programming language in which code should be generated.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1706</td>
<td>Error during code generation.</td>
</tr>
</tbody>
</table>

17.2.1.17.2.8 GenerateCodeInEx

Generates code for all project items of the project in the specified language. The output location is determined by properties of the project and project items. Returns an object that can be used to iterate through all messages issued by the code generation process. These messages are the same as those shown in the Messages window of MapForce.
Signature

GenerateCodeInEx(in i_nLanguage:ENUM ProgrammingLanguage) : ErrorMarkers

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_nLanguage</td>
<td>ENUM ProgrammingLanguage</td>
<td>Specifies the programming language in which code should be generated.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1501</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1706</td>
<td>Error during code generation.</td>
</tr>
</tbody>
</table>

17.2.1.17.2.9 InsertWebService

Inserts a new Web service project into the project's Web service folder. If i_bGenerateMappings is true, initial mapping documents for all ports get generated automatically.

Signature

InsertWebService(in i_strWSDLFile:String, in i_strService:String, in i_strPort:String, in i_bGenerateMappings:Boolean) : ProjectItem

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_strWSDLFile</td>
<td>String</td>
<td>Specifies the path of the WSDL file to add.</td>
</tr>
<tr>
<td>i_strService</td>
<td>String</td>
<td>Specifies the name of the Web service to add.</td>
</tr>
<tr>
<td>i_strPort</td>
<td>String</td>
<td>Specifies the port of the Web service to add.</td>
</tr>
<tr>
<td>i_bGenerateMappings</td>
<td>Boolean</td>
<td>If this parameter is true, initial mapping documents for all ports get generated</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>automatically.</td>
</tr>
</tbody>
</table>

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1501</td>
<td>WSDL file can not be found or is invalid. Service or port names are invalid. Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1503</td>
<td>Operation not supported by current edition.</td>
</tr>
</tbody>
</table>

17.2.1.17.2.10  **Save**

Saves the project to the file defined by FullName.

**Signature**

```csharp
Save() : Void
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1502</td>
<td>Can't save to file.</td>
</tr>
</tbody>
</table>

17.2.1.17.3  **Events**

17.2.1.17.3.1  **OnProjectClosed**

This event is triggered when the project is closed. The project object passed into the event handler should not be accessed. The corresponding open event is Application.OnProjectOpened.

**Signature**

```csharp
OnProjectClosed(in Project): Void
```
17.2.1.18 ProjectItem

A ProjectItem object represents one item in a project tree.

Properties to navigate the object model:

- Application
- Parent

Project tree navigation:

- Count
- Item
- _NewEnum

Project item properties:

- Kind
- Name
- WSDLFile (only available to Web service project items)
- QualifiedName (only available to Web service project items)

Project tree manipulation:

- AddActiveFile (only available to folder items)
- AddFile (only available to folder items)
- CreateFolder (only available to folder items)
- CreateMappingForProject (only available to Web service operations)
- Remove

Document access:

- Open (only available to mapping items and Web service operations)

Code-generation:

- CodeGenSettings_UseDefault
- CodeGenSettings_OutputFolder
- CodeGenSettings_Language
- GenerateCode
- GenerateCodeEx
- GenerateCodeIn
- GenerateCodeInEx

For examples of how to use the properties and methods listed above, see Example: Project Support. For operations with Web services, the MapForce Enterprise edition is required.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_NewEnum</td>
<td>Read-only.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Name</td>
<td>Retrieves or sets the name of a project item. The name of most items is read-only. Exceptions are user-created folders, the names of which can be altered after creation.</td>
</tr>
<tr>
<td>Kind</td>
<td>Read-only. Retrieves the kind of the project item. Availability of some properties and the applicability of certain methods is restricted to specific kinds of project items. The description of all methods and properties contains information about these restrictions.</td>
</tr>
<tr>
<td>Count</td>
<td>Read-only. Retrieves the number of children of this project item. See also Item. For examples, see Project.Item or Project._NewEnum.</td>
</tr>
<tr>
<td>Item</td>
<td>Read-only. Returns the child at n position of this project item. The index is zero-based. The largest valid index is ProjectItem.Count - 1. For an alternative to visit all children, see ProjectItem._NewEnum. For examples, see Project.Item or Project._NewEnum.</td>
</tr>
<tr>
<td>CodeGenSettings_UseDefault</td>
<td>Gets or sets whether output directory and code language are used as defined by either (a) the parent folders, or (b) the project root. This property is used with calls to GenerateCode, GenerateCodeIn, Project.GenerateCode and Project.GenerateCodeIn. If this property is set to false, the values of CodeGenSettings_OutputFolder and CodeGenSettings_Language are used to generate code for this project item.</td>
</tr>
<tr>
<td>CodeGenSettings_OutputFolder</td>
<td>Gets or sets the output directory to be used with GenerateCode, GenerateCodeIn, Project.GenerateCode or Project.GenerateCodeIn. This property is consulted only if CodeGenSettings_UseDefault is set to false.</td>
</tr>
<tr>
<td>CodeGenSettings_Language</td>
<td>Gets or sets the language to be used with GenerateCode or Project.GenerateCode. This property is consulted only if CodeGenSettings_UseDefault is set to false.</td>
</tr>
<tr>
<td>Application</td>
<td>Read-only. Retrieves the top-level application object.</td>
</tr>
<tr>
<td>Name</td>
<td>This property supports language specific standard enumeration. For examples, see Project.Item or Project._NewEnum.</td>
</tr>
</tbody>
</table>
### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Retrieves the project that this item is a child of. Has the same effect as <code>Application.ActiveProject</code>.</td>
</tr>
<tr>
<td>QualifiedName</td>
<td>Read-only. Retrieves the qualified name of a Web service item.</td>
</tr>
<tr>
<td>WSDLFile</td>
<td>Read-only. Retrieves the file name of the WSDL file defining the Web service that hosts the current project item.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddActiveFile</td>
<td>Adds the currently active document to this project item if it is a valid child. Otherwise, the document is added to the Mapping Folder of the project's root.</td>
</tr>
<tr>
<td>AddFile</td>
<td>Adds the specified document to this project item if it is a valid child. Otherwise, the document is added to the Mapping Folder of the project's root.</td>
</tr>
<tr>
<td>CreateFolder</td>
<td>Creates a new folder as a child of this project item.</td>
</tr>
<tr>
<td>CreateMappingForProject</td>
<td>Creates an initial mapping document for a Web service operation and saves it to <code>i_strFileName</code>. When using <code>Project.InsertWebService</code> you can set the <code>i_bGenerateMappings</code> flag to let MapForce automatically generate initial mappings for all ports.</td>
</tr>
<tr>
<td>GenerateCode</td>
<td>Generates code for this project item and its children. The code language and output location is determined by <code>CodeGenSettings.UseDefault</code>, <code>CodeGenSettings.Language</code> and <code>CodeGenSettings.OutputFolder</code>. Children of this project item can have their own property settings related to code-generation.</td>
</tr>
<tr>
<td>GenerateCodeEx</td>
<td>Generates code for this project item and its children. The code language and output location are determined by <code>CodeGenSettings.UseDefault</code>, <code>CodeGenSettings.Language</code> and <code>CodeGenSettings.OutputFolder</code>. Children of this project item can have their own property settings related to code-generation.</td>
</tr>
<tr>
<td>GenerateCodeIn</td>
<td>Generates code for the project item and its children in the specified language. The output location is determined by <code>CodeGenSettings.UseDefault</code> and <code>CodeGenSettings.OutputFolder</code>. Children of this project item can have their own property settings related to code-generation.</td>
</tr>
</tbody>
</table>
### GenerateCodeInEx
Generates code for the project item and its children in the specified language. The output location is determined by `CodeGenSettings_UseDefault` and `CodeGenSettings_OutputFolder`. Children of this project item can have their own property settings related to code-generation.

An object that can be used to iterate through all messages issued by the code generation process is returned. These messages are the same as those shown in the Messages window of MapForce.

### Open
Opens the project item as a document or makes the corresponding document the active one, if it is already open. The project item must be a MapForce mapping or, for Enterprise edition only, Web service operation.

### Remove
Remove this project item and all its children from the project tree.

## Events

### OnModifiedFlagChanged
Occurs when the `ProjectItem`'s modification status changes.

### OnProjectClosed
This event is triggered when the project is closed. The project object passed into the event handler should not be accessed. The corresponding open event is `Application.OnProjectOpened`.

## 17.2.18.1 Properties

### _NewEnum
This property supports language specific standard enumeration. For examples, see `Project.Item` or `Project._NewEnum`.

**Signature**

```
_NewEnum : IUnknown
```
Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
</tbody>
</table>

17.2.1.18.1.2 Application

Retrieves the top-level application object.

Signature

| Application          | Application |

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1701</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.18.1.3 CodeGenSettings_Language

Gets or sets the language to be used with GenerateCode or Project.GenerateCode. This property is consulted only if CodeGenSettings_UseDefault is set to false.

Signature

| CodeGenSettings_Language          | ENUMProgrammingLanguage |

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1701</td>
<td>Invalid language or invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.1.18.1.4 CodeGenSettings_OutputFolder

Gets or sets the output directory to be used with GenerateCode, GenerateCodeIn, Project.GenerateCode or Project.GenerateCodeIn. This property is consulted only if CodeGenSettings_UseDefault is set to false.
Signature

CodeGenSettings_OutputFolder : String

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1701</td>
<td>An invalid output folder or an invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.18.1.15  CodeGenSettings_UseDefault

Gets or sets whether output directory and code language are used as defined by either (a) the parent folders, or (b) the project root. This property is used with calls to GenerateCode, GenerateCodeIn, Project.GenerateCode and Project.GenerateCodeIn. If this property is set to false, the values of CodeGenSettings_OutputFolder and CodeGenSettings_Language are used to generate code for this project item.

Signature

CodeGenSettings_UseDefault : Boolean

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1701</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.18.1.6  Count

Retrieves the number of children of this project item. See also Item. For examples, see Project.Item or Project._NewEnum.
Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
</tbody>
</table>

17.2.18.1.7  Item

Returns the child at n position of this project item. The index is zero-based. The largest valid index is `ProjectItem.Count - 1`. For an alternative to visit all children, see `ProjectItem._NewEnum`. For examples, see `[Project.Item](#)` or `Project._NewEnum`.

Signature

```
Item this(in n:Integer) : ProjectItem
```

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
</tbody>
</table>

17.2.18.1.8  Kind

Retrieves the kind of the project item. Availability of some properties and the applicability of certain methods is restricted to specific kinds of project items. The description of all methods and properties contains information about these restrictions.

Signature

```
Kind : ENUMProjectItemType
```

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1701</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.18.1.9  Name

Retrieves or sets the name of a project item. The name of most items is read-only. Exceptions are user-created folders, the names of which can be altered after creation.
Signature

Name : String

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1701</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1702</td>
<td>Project item does not allow to alter its name.</td>
</tr>
</tbody>
</table>

17.2.18.1.10  Parent

Retrieves the project that this item is a child of. Has the same effect as Application.ActiveProject.

Signature

Parent : Project

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1701</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

17.2.18.1.11  QualifiedName

Retrieves the qualified name of a Web service item.

Signature

QualifiedName : String

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1701</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
### WSDLFile

Retrieves the file name of the WSDL file defining the Web service that hosts the current project item.

**Signature**

```plaintext
WSDLFile : String
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1701</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1702</td>
<td>The project item is not a part of a Web service.</td>
</tr>
</tbody>
</table>

### Methods

#### AddActiveFile

Adds the currently active document to this project item if it is a valid child. Otherwise, the document is added to the Mapping Folder of the project's root.

**Signature**

```plaintext
AddActiveFile() : ProjectItem
```

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1701</td>
<td>The file name is empty. Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1703</td>
<td>No active document is available.</td>
</tr>
<tr>
<td>1704</td>
<td>Active documents needs to be given a path name before it can be added to the project.</td>
</tr>
</tbody>
</table>
### Error code

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1705</td>
<td>Mapping could not be assigned to project. The file does not exist or is not a MapForce mapping. Maybe the file is already assigned to the target folder.</td>
</tr>
</tbody>
</table>

#### 17.2.1.18.2.2 AddFile

Adds the specified document to this project item if it is a valid child. Otherwise, the document is added to the Mapping Folder of the project's root.

**Signature**

AddFile(in i_strFilePath:String) : ProjectItem

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_strFilePath</td>
<td>String</td>
<td>The path of the document to add.</td>
</tr>
</tbody>
</table>

**Errors**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1701</td>
<td>The file name is empty. Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1705</td>
<td>Mapping could not be assigned to project. The file does not exist or is not a MapForce mapping. Maybe the file is already assigned to the target folder.</td>
</tr>
</tbody>
</table>

#### 17.2.1.18.2.3 CreateFolder

Creates a new folder as a child of this project item.

**Signature**

CreateFolder(in i_strFolderName: String) : ProjectItem

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_strFolderName</td>
<td>String</td>
<td>The name of the folder to create.</td>
</tr>
</tbody>
</table>
Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1701</td>
<td>Invalid folder name or invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1702</td>
<td>The project item does not support children.</td>
</tr>
</tbody>
</table>

17.2.1.18.2.4  CreateMappingForProject

Creates an initial mapping document for a Web service operation and saves it to `i_strFileName`. When using `Project.InsertWebService` you can set the `i_bGenerateMappings` flag to let MapForce automatically generate initial mappings for all ports.

Signature

CreateMappingForProject(in i_strFileName:String) : ProjectItem

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_strFileName</td>
<td>String</td>
<td>Specifies the path where the mapping should be saved.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1701</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1702</td>
<td>The project item does not support children.</td>
</tr>
<tr>
<td>1707</td>
<td>Cannot create new mapping. The project item does not support auto-creation of initial mappings or a mapping already exists.</td>
</tr>
<tr>
<td>1708</td>
<td>Operation not supported in current edition.</td>
</tr>
</tbody>
</table>

17.2.1.18.2.5  GenerateCode

Generates code for this project item and its children. The code language and output location is determined by `CodeGenSettings.UseDefault`, `CodeGenSettings.Language` and `CodeGenSettings.OutputFolder`. Children of this project item can have their own property settings related to code-generation.
Signature

```plaintext
GenerateCode() : Void
```

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1706</td>
<td>Error during code generation.</td>
</tr>
</tbody>
</table>

17.2.18.2.6  GenerateCodeEx

Generates code for this project item and its children. The code language and output location are determined by `CodeGenSettings_UseDefault`, `CodeGenSettings_Language` and `CodeGenSettings_OutputFolder`. Children of this project item can have their own property settings related to code-generation.

Signature

```plaintext
GenerateCodeEx() : ErrorMarkers
```

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1701</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1706</td>
<td>Error during code generation.</td>
</tr>
</tbody>
</table>

17.2.18.2.7  GenerateCodeIn

Generates code for the project item and its children in the specified language. The output location is determined by `CodeGenSettings_UseDefault` and `CodeGenSettings_OutputFolder`. Children of this project item can have their own property settings related to code-generation.

Signature

```plaintext
GenerateCodeIn(in i_nLanguage:ENUMProgrammingLanguage) : Void
```
Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_nLanguage</td>
<td>ENUMProgrammingLanguage</td>
<td>Specifies the programming language for code generation.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1701</td>
<td>Invalid language specified.</td>
</tr>
<tr>
<td>1706</td>
<td>Error during code generation.</td>
</tr>
</tbody>
</table>

17.2.1.18.2.8  GenerateCodeInEx

Generates code for the project item and its children in the specified language. The output location is determined by CodeGenSettings_UseDefault and CodeGenSettings_OutputFolder. Children of this project item can have their own property settings related to code-generation.

An object that can be used to iterate through all messages issued by the code generation process is returned. These messages are the same as those shown in the Messages window of MapForce.

Signature

GenerateCodeInEx(in i_nLanguage:ENUMProgrammingLanguage) : ErrorMarkers

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_nLanguage</td>
<td>ENUMProgrammingLanguage</td>
<td>Specifies the programming language for code generation.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1701</td>
<td>Invalid language specified or invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1706</td>
<td>Error during code generation.</td>
</tr>
</tbody>
</table>
17.2.18.2.9  Open

Opens the project item as a document or makes the corresponding document the active one, if it is already open. The project item must be a MapForce mapping or, for Enterprise edition only, Web service operation.

Signature

Open() : Document

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>1701</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1702</td>
<td>The project item does not refer to a MapForce mapping file.</td>
</tr>
<tr>
<td>1708</td>
<td>Operation not supported in current edition.</td>
</tr>
</tbody>
</table>

17.2.18.2.10  Remove

Remove this project item and all its children from the project tree.

Signature

Remove() : Void

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700</td>
<td>The object is no longer valid.</td>
</tr>
</tbody>
</table>
17.2.1.18.3  Events

17.2.1.18.3.1  OnModifiedFlagChanged

Occurs when the ProjectItem’s modification status changes.

**Signature**

```
OnModifiedFlagChanged(in Boolean): Void
```

17.2.1.18.3.2  OnProjectClosed

This event is triggered when the project is closed. The project object passed into the event handler should not be accessed. The corresponding open event is `Application.OnProjectOpened`.

**Signature**

```
OnProjectClosed(in Project): Void
```

17.2.2  Enumerations

17.2.2.1  ENUMApacheAxisVersion (obsolete)

This enumeration type is obsolete.

**Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eApacheAxisVersion_Axis</td>
<td>1</td>
</tr>
<tr>
<td>eApacheAxisVersion_Axis2</td>
<td>2</td>
</tr>
</tbody>
</table>

17.2.2.2  ENUMApplicationStatus

Enumeration values to indicate the status of the application.

**Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eApplicationRunning</td>
<td>0</td>
</tr>
</tbody>
</table>
17.2.2.3 **ENUM AppOutputLine_Severity**

Enumeration values to identify the severity of an `AppOutputLine`.

**Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>eSeverity_Undefined</code></td>
<td>-1</td>
</tr>
<tr>
<td><code>eSeverity_Info</code></td>
<td>0</td>
</tr>
<tr>
<td><code>eSeverity_Warning</code></td>
<td>1</td>
</tr>
<tr>
<td><code>eSeverity_Error</code></td>
<td>2</td>
</tr>
<tr>
<td><code>eSeverity_CriticalError</code></td>
<td>3</td>
</tr>
<tr>
<td><code>eSeverity_Success</code></td>
<td>4</td>
</tr>
<tr>
<td><code>eSeverity_Summary</code></td>
<td>5</td>
</tr>
<tr>
<td><code>eSeverity_Progress</code></td>
<td>6</td>
</tr>
<tr>
<td><code>eSeverity_DataEdit</code></td>
<td>7</td>
</tr>
<tr>
<td><code>eSeverity_ParserInfo</code></td>
<td>8</td>
</tr>
<tr>
<td><code>eSeverity_PossibleInconsistencyWarning</code></td>
<td>9</td>
</tr>
<tr>
<td><code>eSeverity_Message</code></td>
<td>10</td>
</tr>
<tr>
<td><code>eSeverity_Document</code></td>
<td>11</td>
</tr>
<tr>
<td><code>eSeverity_Rest</code></td>
<td>12</td>
</tr>
<tr>
<td><code>eSeverity_NoSelect</code></td>
<td>13</td>
</tr>
<tr>
<td><code>eSeverity_Select</code></td>
<td>14</td>
</tr>
<tr>
<td><code>eSeverity_Autoinsertion</code></td>
<td>15</td>
</tr>
<tr>
<td><code>eSeverity_GlobalResources_DefaultWarning</code></td>
<td>16</td>
</tr>
<tr>
<td><code>eSeverity_XPath_Styles_Changed</code></td>
<td>17</td>
</tr>
</tbody>
</table>
### 17.2.2.4 ENUMAppOutputLine_TextDecoration

Enumeration values for the different kinds of text decoration of an AppOutputLine.

**Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTextDecorationDefault</td>
<td>0</td>
</tr>
<tr>
<td>eTextDecorationBold</td>
<td>1</td>
</tr>
<tr>
<td>eTextDecorationDebugValues</td>
<td>2</td>
</tr>
<tr>
<td>eTextDecorationDBObjectName</td>
<td>3</td>
</tr>
<tr>
<td>eTextDecorationDBObjectLink</td>
<td>4</td>
</tr>
<tr>
<td>eTextDecorationDBObjectKind</td>
<td>5</td>
</tr>
<tr>
<td>eTextDecorationDBTimeoutValue</td>
<td>6</td>
</tr>
<tr>
<td>eTextDecorationFind_MatchingString</td>
<td>7</td>
</tr>
<tr>
<td>eTextDecorationValidation_Speclink</td>
<td>8</td>
</tr>
<tr>
<td>eTextDecorationValidation_ErrorPosition</td>
<td>9</td>
</tr>
<tr>
<td>eTextDecorationValidation_UnknownParam</td>
<td>10</td>
</tr>
</tbody>
</table>
17.2.2.5  **ENUMCodeGenErrorLevel**

Enumeration values to identify severity of code generation messages.

**Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eCodeGenErrorLevel_Information</td>
<td>0</td>
</tr>
<tr>
<td>eCodeGenErrorLevel_Warning</td>
<td>1</td>
</tr>
<tr>
<td>eCodeGenErrorLevel_Error</td>
<td>2</td>
</tr>
<tr>
<td>eCodeGenErrorLevel_Undefined</td>
<td>3</td>
</tr>
</tbody>
</table>

17.2.2.6  **ENUMComponentDatapointSide**

Enumeration values to indicate the side of a datapoint on its component. See also `Component.GetRootDatapoint`.

**Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDatapointSideInput</td>
<td>0</td>
</tr>
<tr>
<td>eDatapointSideOutput</td>
<td>1</td>
</tr>
</tbody>
</table>

17.2.2.7  **ENUMComponentSubType**

Enumeration values to indicate component sub types.

**Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eComponentSubType_None</td>
<td>0</td>
</tr>
<tr>
<td>eComponentSubType_Text_EDI</td>
<td>1</td>
</tr>
<tr>
<td>eComponentSubType_Text_Flex</td>
<td>2</td>
</tr>
<tr>
<td>eComponentSubType_Text_CSVFLF</td>
<td>3</td>
</tr>
</tbody>
</table>
17.2.2.8  **ENUMComponentType**

Enumeration values to indicate component types.

**Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eComponentType_Unknown</td>
<td>0</td>
</tr>
<tr>
<td>eComponentType_XML</td>
<td>1</td>
</tr>
<tr>
<td>eComponentType_DB</td>
<td>2</td>
</tr>
<tr>
<td>eComponentType_Text</td>
<td>3</td>
</tr>
<tr>
<td>eComponentType_Excel</td>
<td>4</td>
</tr>
<tr>
<td>eComponentType_WSDL</td>
<td>5</td>
</tr>
<tr>
<td>eComponentType_XBRL</td>
<td>6</td>
</tr>
<tr>
<td>eComponentType_Input</td>
<td>7</td>
</tr>
<tr>
<td>eComponentType_JSON</td>
<td>8</td>
</tr>
</tbody>
</table>

17.2.2.9  **ENUMComponentUsageKind**

Enumeration values to indicate component usage kind.

**Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eComponentUsageKind_Unknown</td>
<td>0</td>
</tr>
<tr>
<td>eComponentUsageKind_Instance</td>
<td>1</td>
</tr>
<tr>
<td>eComponentUsageKind_Input</td>
<td>2</td>
</tr>
<tr>
<td>eComponentUsageKind_Output</td>
<td>3</td>
</tr>
<tr>
<td>eComponentUsageKind_Variable</td>
<td>4</td>
</tr>
<tr>
<td>eComponentUsageKind_String</td>
<td>5</td>
</tr>
</tbody>
</table>
17.2.2.10  **ENUMConnectionType**

Enumeration values to indicate the type of a connection. See also `Connection.ConnectionType`.

**Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eConnectionTypeTargetDriven</td>
<td>0</td>
</tr>
<tr>
<td>eConnectionTypeSourceDriven</td>
<td>1</td>
</tr>
<tr>
<td>eConnectionTypeCopyAll</td>
<td>2</td>
</tr>
</tbody>
</table>

17.2.2.11  **ENUMDOMType**

Enumeration values to specify the DOM type used by generated C++ mapping code.

**NOTE:** The value `eDOMType_xerces` is obsolete. `eDOMType_xerces3` indicates Xerces 3.x usage. Obsolete in this context means that this value is not supported and should not be used.

**Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDOMType_xerces (obsolete)</td>
<td>1</td>
</tr>
<tr>
<td>eDOMType_xerces3</td>
<td>2</td>
</tr>
<tr>
<td>eDOMType_msxml6</td>
<td>3</td>
</tr>
</tbody>
</table>

17.2.2.12  **ENULibType**

Enumeration values to specify the library type used by the generated C++ mapping code.

**Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eLibType_static</td>
<td>0</td>
</tr>
<tr>
<td>eLibType_dll</td>
<td>1</td>
</tr>
</tbody>
</table>
17.2.2.13  **ENUMProgrammingLanguage**

Enumeration values to select a programming language.

**Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eUndefinedLanguage</td>
<td>-1</td>
</tr>
<tr>
<td>eJava</td>
<td>0</td>
</tr>
<tr>
<td>eCpp</td>
<td>1</td>
</tr>
<tr>
<td>eCSharp</td>
<td>2</td>
</tr>
<tr>
<td>eXSLT</td>
<td>3</td>
</tr>
<tr>
<td>eXSLT2</td>
<td>4</td>
</tr>
<tr>
<td>eXQuery</td>
<td>5</td>
</tr>
</tbody>
</table>

17.2.2.14  **ENUMProjectItemType**

Enumeration to identify the different kinds of project items that can be children of `Project` or folder-like `ProjectItems`. See also `ProjectItem.Kind`.

**Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eProjectItemType_MappingFolder</td>
<td>0</td>
</tr>
<tr>
<td>eProjectItemType_Mapping</td>
<td>1</td>
</tr>
<tr>
<td>eProjectItemType_WebServiceFolder</td>
<td>2</td>
</tr>
<tr>
<td>eProjectItemType_WebServiceRoot</td>
<td>3</td>
</tr>
<tr>
<td>eProjectItemType_WebServiceService</td>
<td>4</td>
</tr>
<tr>
<td>eProjectItemType_WebServicePort</td>
<td>5</td>
</tr>
<tr>
<td>eProjectItemType_WebServiceOperation</td>
<td>6</td>
</tr>
<tr>
<td>eProjectItemType_ExternalFolder</td>
<td>7</td>
</tr>
<tr>
<td>eProjectItemType_LibraryFolder</td>
<td>8</td>
</tr>
<tr>
<td>eProjectItemType_ResourceFolder</td>
<td>9</td>
</tr>
<tr>
<td>eProjectItemType_VirtualFolder</td>
<td>10</td>
</tr>
</tbody>
</table>
### ENUMProjectItemType

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eProjectItemType_Count</td>
<td>11</td>
</tr>
<tr>
<td>eProjectItemType_Invalid</td>
<td>-1</td>
</tr>
</tbody>
</table>

**17.2.2.15 ENUMProjectItemType**

Enumeration values to select a project type for generated C# and C++ mapping code.

#### Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eVisualStudio2008Project</td>
<td>5</td>
</tr>
<tr>
<td>eVisualStudio2010Project</td>
<td>6</td>
</tr>
<tr>
<td>eVisualStudio2013Project</td>
<td>7</td>
</tr>
<tr>
<td>eVisualStudio2015Project</td>
<td>8</td>
</tr>
<tr>
<td>eVisualStudio2017Project</td>
<td>9</td>
</tr>
<tr>
<td>eProjectTypeCount</td>
<td>10</td>
</tr>
</tbody>
</table>

### ENUMSearchDatapointFlags

Enumeration values used as bit-flags; to be used as combination of flags when searching for a datapoint. See also GetChild.

#### Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eSearchDatapointElement</td>
<td>1</td>
</tr>
<tr>
<td>eSearchDatapointAttribute</td>
<td>2</td>
</tr>
</tbody>
</table>
17.2.2.17 ENUMViewMode

Enumeration values to select a MapForce view.

### Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eMapForceView</td>
<td>0</td>
</tr>
<tr>
<td>eXSLView</td>
<td>1</td>
</tr>
<tr>
<td>eOutputView</td>
<td>2</td>
</tr>
</tbody>
</table>
Chapter 18
ActiveX Integration
18 **ActiveX Integration**

The MapForce user interface and the functionality described in this section can be integrated into custom applications that can consume ActiveX controls. ActiveX technology enables a wide variety of languages to be used for integration, such as C++, C#, VB.NET, HTML. (Note that ActiveX components integrated in HTML must be run with Microsoft Internet Explorer versions and platforms that support ActiveX). All components are full OLE Controls. Integration into Java is provided through wrapper classes.

To integrate the ActiveX controls into your custom code, the MapForce Integration Package must be installed (see [https://www.altova.com/components/download](https://www.altova.com/components/download)). Ensure that you install MapForce first, and then the MapForce Integration Package. Other prerequisites apply, depending on language and platform (see [Prerequisites](#)).

You can flexibly choose between two different levels of integration: application level and document level.

Integration at application level means embedding the complete interface of MapForce (including its menus, toolbars, panes, etc) as an ActiveX control into your custom application. For example, in the most simple scenario, your custom application could consist of only one form that embeds the MapForce graphical user interface. This approach is easier to implement than integration at document level but may not be suitable if you need flexibility to configure the MapForce graphical user interface according to your custom requirements.

Integration at document level means embedding MapForce into your own application piece-by-piece. This includes implementing not only the main MapForce control but also the main document editor window, and, optionally, any additional windows. This approach provides greater flexibility to configure the GUI, but requires advanced interaction with ActiveX controls in your language of choice.

The sections **Integration at the Application Level** and **Integration at Document Level** describe the key steps at these respective levels. The **ActiveX Integration Examples** section provides examples in C#, HTML, and Java. Looking through these examples will help you to make the right decisions quickly. The **Object Reference** section describes all COM objects that can be used for integration, together with their properties and methods.

For information about using MapForce as a Visual Studio plug-in, see [MapForce in Visual Studio](#).
18.1 Prerequisites

To integrate the MapForce ActiveX control into a custom application, the following must be installed on your computer:

- MapForce
- The MapForce Integration Package, available for download at [https://www.altova.com/components/download](https://www.altova.com/components/download)

To integrate the 64-bit ActiveX control, install the 64-bit versions of MapForce and MapForce Integration Package. For applications developed under Microsoft .NET platform with Visual Studio, both the 32-bit and 64-bit versions of MapForce and MapForce Integration Package must be installed, as explained below.

**Microsoft .NET (C#, VB.NET) with Visual Studio**

To integrate the MapForce ActiveX control into a 32-bit application developed under Microsoft .NET, the following must be installed on your computer:

- Microsoft .NET Framework 4.0 or later
- Visual Studio 2010/2012/2013/2015/2017
- MapForce 32-bit and MapForce Integration Package 32-bit
- The ActiveX controls must be added to the Visual Studio toolbox (see Adding the ActiveX Controls to the Toolbox).

If you want to integrate the 64-bit ActiveX control, the following prerequisites apply in addition to the ones above:

- MapForce 32-bit and MapForce Integration Package 32-bit must still be installed (this is required to provide the 32-bit ActiveX control to the Visual Studio designer, since Visual Studio runs on 32-bit)
- MapForce 64-bit and MapForce Integration Package 64-bit must be installed (provides the actual 64-bit ActiveX control to your custom application at runtime)
- In Visual Studio, create a 64-bit build configuration and build your application using this configuration. For an example, see Running the Sample C# Solution.

**Java**

To integrate the MapForce ActiveX control into Java application using the Eclipse development environment, the following must be installed on your computer:

- Java Runtime Environment (JRE) or Java Development Kit (JDK) 7 or later
- Eclipse
- MapForce and MapForce Integration Package

**Note:** To run the 64-bit version of the MapForce ActiveX control, use a 64-bit version of Eclipse, as well as the 64-bit version of MapForce and the MapForce Integration Package.

**MapForce integration and deployment on client computers**

If you create a .NET application and intend to distribute it to other clients, you will need to install
the following on the client computer(s):

- MapForce
- The MapForce Integration Package
- The custom integration code or application.
18.2 Adding the ActiveX Controls to the Toolbox

To use the MapForce ActiveX controls in an application developed with Visual Studio, the controls must first be added to the Visual Studio Toolbox, as follows:

1. On the Tools menu of Visual Studio, click Choose Toolbox Items.
2. On the COM Components tab, select the check boxes next to the MapForceControl, MapForceControl Document, and MapForceControl Placeholder.

In case the controls above are not available, follow the steps below:

1. On the COM Components tab, click Browse, and select the MapForceControl.ocx file from the MapForce installation folder. Remember that the MapForce Integration Package must be installed; otherwise, this file is not available, see Prerequisites.
2. If prompted to restart Visual Studio with elevated permissions, click Restart under different credentials.

If the steps above were successful, the MapForce ActiveX controls become available in the Visual Studio Toolbox.
Note: For an application-level integration, only the `MapForceControl` ActiveX control is used (see Integration at Application Level). The `MapForceControl Document` and `MapForceControl Placeholder` controls are used for document-level integration (see Integration at Document Level).
18.3 Integration at Application Level

Integration at application level allows you to embed the complete interface of MapForce into a window of your application. With this type of integration, you get the whole user interface of MapForce, including all menus, toolbars, the status bar, document windows, and helper windows. Customization of the application's user interface is restricted to what MapForce provides. This includes rearrangement and resizing of helper windows and customization of menus and toolbars.

The only ActiveX control you need to integrate is MapForceControl. Do not instantiate or access MapForceControlDocument or MapForceControlPlaceHolder ActiveX controls when integrating at application-level.

If you have any initialization to do or if you want to automate some behaviour of MapForce, use the properties, methods, and events described for MapForceControl. Consider using MapForceControl.Application for more complex access to MapForce functionality.

For an example that shows how the MapForce application can be embedded in an HTML page, see HTML Integration at Application Level.

In C# or VB.NET with Visual Studio, the steps to create a basic, one-form application which integrates the MapForce ActiveX controls at application level are as follows:

1. Check that all prerequisites are met (see Prerequisites).
2. Create a new Visual Studio Windows Forms project with a new empty form.
3. If you have not done that already, add the ActiveX controls to the toolbox (see Adding the ActiveX Controls to the Toolbox).
4. Drag the MapForceControl from the toolbox onto your new form.
5. Select the MapForceControl on the form, and, in the Properties window, set the IntegrationLevel property to ICAciveXIntegrationOnApplicationLevel.
6. Create a build platform configuration that matches the platform under which you want to build (x86, x64). Here is how you can create the build configuration:

   a. Right-click the solution in Visual Studio, and select **Configuration Manager**.
   b. Under **Active solution platform**, select **New...** and then select the x86 or x64 configuration (in this example, **x86**).

   ![New Solution Platform](image)

   You are now ready to build and run the solution in Visual Studio. Remember to build using the
configuration that matches your target platform (x86, x64).
18.4 Integration at Document Level

Compared to integration at application level, integration at document level is a more complex, yet more flexible way to embed MapForce functionality into your application by means of ActiveX controls. With this approach, your code can access selectively the following parts of the MapForce user interface:

- Document editing window
- Project window
- Libraries window
- Overview window
- Messages window

As mentioned in Integration at Application Level, for an ActiveX integration at application level, only one control is required, namely the MapForceControl. However, for an ActiveX integration at document level, functionality MapForce is provided by the following ActiveX controls:

1. MapForceControl
2. MapForceControl Document
3. MapForceControl Placeholder

These controls are supplied by the MapForceControl.ocx file available in the application installation folder of MapForce. When you develop the ActiveX integration with Visual Studio, you will need to add these controls to the Visual Studio toolbox (see Adding the ActiveX Controls to the Toolbox).

The basic steps to integrate the ActiveX controls at document level into your application are as follows:

1. First, instantiate MapForceControl in your application. Instantiating this control is mandatory; it enables support for the MapForceControl Document and MapForceControl Placeholder controls mentioned above. It is important to set the IntegrationLevel property to ICActiveXIntegrationOnDocumentLevel (or "1"). To hide the control from the user, set its Visible property to False.

   **Note:** When integrating at document level, do not use the Open method of the MapForceControl; this might lead to unexpected results. Use the corresponding open methods of MapForceControl Document and MapForceControl PlaceHolder instead.

2. Create at least one instance of MapForceControl Document in your application. This control supplies the document editing window of MapForce to your application and can be instantiated multiple times if necessary.

   Use the method Open to load any existing file. To access document-related functionality, use the Path and Save or methods and properties accessible via the property Document.

   **Note:** The control does not support a read-only mode. The value of the property ReadOnly is ignored.

3. Optionally, add to your application the MapForceControl Placeholder control for each
additional window (other than the document window) that must be available to your application.

Instances of MapForceControl PlaceHolder allow you to selectively embed additional windows of MapForce into your application. The window kind (for example, Project window) is defined by the property PlaceholderWindowID. Therefore, to set the window kind, set the property PlaceholderWindowID. For valid window identifiers, see MapForceControlPlaceholderWindow.

Note: Use only one MapForceControl PlaceHolder for each window identifier.

For placeholder controls that select the MapForce project window, additional methods are available. Use OpenProject to load a MapForce project. Use the property Project and the methods and properties from the MapForce automation interface to perform any other project related operations.

For example, in C# or VB.NET with Visual Studio, the steps to create a basic, one-form application which integrates the MapForce ActiveX controls at document level could be similar to those listed below. Note that your application may be more complex if necessary; however, the instructions below are important to understand the minimum requirements for an ActiveX integration at document level.

1. Create a new Visual Studio Windows Forms project with a new empty form.
2. If you have not done that already, add the ActiveX controls to the toolbox (see Adding the ActiveX Controls to the Toolbox).
3. Drag the MapForceControl from the toolbox onto your new form.
4. Set the IntegrationLevel property of the MapForceControl to ICAcitiveXIntegrationOnDocumentLevel, and the Visible property to False. You can do this either from code or from the Properties window.
5. Drag the MapForceControl Document from the toolbox onto the form. This control provides the main document window of MapForce to your application, so you may need to resize it to a reasonable size for a document.
6. Optionally, add one or more MapForceControl Placeholder controls to the form (one for each additional window type that your application needs, for example, the Project window). You will typically want to place such additional placeholder controls either below or to the right or left of the main document control, for example:
7. Set the **PlaceholderWindowID** property of each **MapForceControl Placeholder** control to a valid window identifier. For the list of valid values, see **MapForceControlPlaceholderWindow**.

8. Add commands to your application (at minimum, you will need to open, save and close documents), as shown below.

**Querying MapForce Commands**

When you integrate at document level, no MapForce menu or toolbar is available to your application. Instead, you can retrieve the required commands, view their status, and execute them programmatically, as follows:

- To retrieve all available commands, use the **CommandsList** property of the **MapForceControl**.
- To retrieve commands organized according to their menu structure, use the **MainMenu** property.
- To retrieve commands organized by the toolbar in which they appear, use the **Toolbars** property.
- To send commands to MapForce, use the **Exec** method.
- To query if a command is currently enabled or disabled, use the **QueryStatus** method.

This enables you to flexibly integrate MapForce commands into your application's menus and toolbars.

Your installation of MapForce also provides you with command label images used within MapForce. See the folder `<ApplicationFolder>\Examples\ActiveX\Images` of your MapForce installation for icons in GIF format. The file names correspond to the command names as they are listed in the **Command Reference** section.
General considerations

To automate the behaviour of MapForce, use the properties, methods, and events described for the MapForceControl, MapForceControl Document, and MapForceControl Placeholder.

For more complex access to MapForce functionality, consider using the following properties:

- MapForceControl.Application
- MapForceControlDocument.Document
- MapForceControlPlaceHolder.Project

These properties give you access to the MapForce automation interface (MapForceAPI)

**Note:** To open a document, always use MapForceControlDocument.Open or MapForceControlDocument.New on the appropriate document control. To open a project, always use MapForceControlPlaceHolder.OpenProject on a placeholder control embedding a MapForce project window.

For examples that show how to instantiate and access the necessary controls in different programming environments, see ActiveX Integration Examples.
18.5 ActiveX Integration Examples

This section contains examples of MapForce document-level integration using different container environments and programming languages. (The HTML section additionally contains examples of integration at application level.) Source code for all examples is available in the folder `<ApplicationFolder>\Examples\ActiveX` of your MapForce installation.

18.5.1 C#

A basic ActiveX integration example solution for C# and Visual Studio is available in the folder `<ApplicationFolder>\Examples\ActiveX\C#`. Before you compile the source code and run the sample, make sure that all prerequisites are met (see Running the Sample C# Solution).

18.5.1.1 Running the Sample C# Solution

The sample Visual Studio solution available in the folder `<ApplicationFolder>\Examples\ActiveX\C#` illustrates how to consume the MapForce ActiveX controls. Before attempting to build and run this solution, note the following steps:

**Step 1: Check the prerequisites**
Visual Studio 2010 or later is required to open the sample solution. For the complete list of prerequisites, see Prerequisites.

**Step 2: Copy the sample to a directory where you have write permissions**
To avoid running Visual Studio as an Administrator, copy the source code to a directory where you have write permissions, instead of running it from the default location.

**Step 3: Check and set all required control properties**
The sample application contains one instance of MapForceControlDocument and several instances of MapForceControlPlaceHolder controls. Double-check that the following properties of these controls are set as shown in the table below:

<table>
<thead>
<tr>
<th>Control name</th>
<th>Property</th>
<th>Property value</th>
</tr>
</thead>
<tbody>
<tr>
<td>axMapForceControl</td>
<td>IntegrationLevel</td>
<td>ICActiveXIntegrationOnDocumentLevel</td>
</tr>
<tr>
<td>axMapForceControlLibrary</td>
<td>PlaceholderWindowID</td>
<td>0</td>
</tr>
<tr>
<td>axMapForceControlOutput</td>
<td>PlaceholderWindowID</td>
<td>2</td>
</tr>
<tr>
<td>axMapForceControlPreview</td>
<td>PlaceholderWindowID</td>
<td>1</td>
</tr>
</tbody>
</table>
Here is how you can view or set the properties of an ActiveX control:

1. Open the MDIMain.cs form in the designer window.

   **Note:** On 64-bit Windows, it may be necessary to change the build configuration of the Visual Studio solution to "x86" **before** opening the designer window. If you need to build the sample as a 64-bit application, see [Prerequisites](#).

2. Open the Document Outline window of Visual Studio (On the **View** menu, click **Other Windows | Document Outline**).

3. Click an ActiveX control in the Document Outline window, and edit its required property in the Properties window, for example:
Step 4: Set the build platform

- Create a build platform configuration that matches the platform under which you want to build (x86, x64). Here is how you can create the build configuration:

  a. Right-click the solution in Visual Studio, and select Configuration Manager.
  b. Under Active solution platform, select New... and then select the x86 or x64 configuration (in this example, x86).

You are now ready to build and run the solution in Visual Studio. Remember to build using the configuration that matches your target platform (x86, x64); otherwise, runtime errors might occur.
On running the sample, the main MDI Frame window is displayed. Use File | Open to open a mapping file (for example, MarketingExpenses.mfd, which is in the MapForce examples folder). The file is loaded and displayed in a new document child window:

After you load the document, you can execute commands against the active document using the menu. Context menus are also available. You can also load additional documents. Save any modifications using the File | Save command.

### 18.5.1.2 Retrieving Command Information

The MapForceControl gives access to all commands of MapForce through its CommandsList, MainMenu, and Toolbars properties. The example project available in the folder `<ApplicationFolder>\Examples\ActiveX\C#` uses the MainMenu property to create the MapForce menu structure dynamically.

The code that gets the menu commands can be found in the MDIMain method in MDIMain.cs file:

```csharp
public MDIMain()
{
    // ...

    // Get the MainMenu property of the control and create the menu structure
```
from it.
    MFLib.MapForceCommand objCommand = this.axMapForceControl.MainMenu;
    InsertMenuStructure(mainMenu, objCommand);
}

In the code listing above, `mainMenu` is the existing static menu of the main MDI Frame window. If you open the `MDIMain.cs` form in the Visual Studio Designer, you will notice that this menu contains two menu items: `File` and `Window`.

The method `InsertMenuStructure` takes as parameters the `mainMenu` and the `objCommand` objects (the former is the existing static menu, while the latter contains the full menu structure retrieved from the MapForce ActiveX control). The retrieved MapForce menu structure is then merged into the existing static menu. Note that the menus `File`, `Project`, and `Window` are not added dynamically. This is intentional, because these menus deal with actively open documents, and they would require code which is beyond the scope of this example. The basic file management commands (create, open, save, bring into focus) are handled by the existing static menus `File` and `Window`. All other menus are inserted dynamically based on the information taken from the `MainMenu` property of the ActiveX control. The new menus are inserted after "File" but before "Window", i.e. starting at menu index 1.

The method `InsertMenuStructure` iterates through all top-level menus found in `MapForceCommand` object and adds a new menu item for each. Since each top-level menu has its own child menu items, a call to the method `InsertMenuCommand` takes place for each encountered child menu item. Furthermore, since each child menu item can have its own children menu items, and so on, the `InsertMenuCommand` method recurses into itself until no more child menu items exist.

The commands added dynamically are instances of the class `CustomMenuItem`, which is defined in `CustomMenuItem.cs`. This class is derived from `System.Windows.Forms.MenuItem` class and has an additional member to store the MapForce command ID.

```csharp
public class CustomMenuItem : System.Windows.Forms.MenuItem
{
    public int m_MapForceCmdID;
}
```

All dynamically added commands (except those that are containers for other commands) get the same event handler `AltovaMenuItem_Click` which does the processing of the command.
private void AltovaMenuItem_Click(object sender, EventArgs e)
{
    if(sender.GetType() == System.Type.GetType("MapForceApplication.CustomMenuItem"))
    {
        CustomMenuItem customItem = (CustomMenuItem)sender;
        ProcessCommand(customItem.m_MapForceCmdID);
    }
}

If the command is a container for other commands (that is, if it has child commands), it gets the event handler AltovaSubMenu_Popup. This handler queries the status of each child command and enables or disables it as required. This ensures that each command is enabled only when that is meaningful (for example, the File | Save menu item should be disabled if there is no active document open).

The method ProcessCommand delegates the execution either to the MapForceControl itself or to any active MapForce document loaded in a MapForceControlDocument control. This is necessary because the MapForceControl has no way to know which document is currently active in the hosting application.

private void ProcessCommand(int nID)
{
    MapForceDoc docMapForce = GetCurrentMapForceDoc();

    if(docMapForce != null)
        docMapForce.axMapForceControlDoc.Exec(nID);
    else
        axMapForceControl.Exec(nID);
}

18.5.1.3 Handling Events

Because all events in the MapForce library are based on connection points, you can use the C# delegate mechanism to provide the custom event handlers. You will always find a complete list of events on the property page of each control of the MapForce library. The image below shows the events of the main MapForceControl:
As you can see, the example project only overrides the `OnFileExternalChange` event. The creation of the C# delegate is done for you by the C# Framework. All you need to do is fill in the empty event handler.

For example, the handler implementation shown below turns off any file reloading and displays a message box to inform the user that a file loaded by the MapForceControl has been changed from outside:

```csharp
private void axMapForceControl_OnFileExternalChange(object sender, AxMapForceControlLib._DMapForceControlEvents_OnFileExternalChangeEvent e)
{
    MessageBox.Show("Attention: The file " + e.strPath + " has been changed from outside\nbut reloading is turned off in the sample application!");

    // This turns off any file reloading:
    e.varRet = false;
}
```
18.5.2 HTML

The code listings in this section show how to integrate the MapForceControl at application level and document level. Source code for all examples is available in the folder <ApplicationFolder>\Examples\ActiveX\HTML of your MapForce installation.

Note: ActiveX controls in an HTML page are supported only by Internet Explorer when it runs as a 32-bit application. When Internet Explorer 10 or 11 runs in 64-bit mode, it does not load ActiveX controls. The default browser security settings will normally block ActiveX, so you will need to explicitly allow blocked content to run on the page when prompted by Internet Explorer.

18.5.2.1 HTML Integration at Application Level

This example shows a simple integration of the MapForce control at application-level into a HTML page. The integration is described in the following sections:

- Instantiate a MapForceControl in HTML code.
- Implement buttons to load documents and automate code-generation tasks.
- Define actions for some application events.

The code for this example is available at the following location in your MapForce installation:
<ApplicationFolder>\Examples\ActiveX\HTML\MapForceActiveX_ApplicationLevel.htm

18.5.2.1.1 Instantiate the Control

The HTML Object tag is used to create an instance of the MapForceControl. The Classid is that of MapForceControl. Width and height specify the window size. No additional parameters are necessary, since application-level is the default.

```html
<OBJECT id="objMapForceControl"
    Classid="clsid:A38637E9-5759-4456-A167-F01160CC22C1"
    width="800"
    height="500"
    VIEWASTEXT>
</OBJECT>
```

18.5.2.1.2 Add Button to Open Default Document

As a simple example of how to automate some tasks, we add a button to the page:

```html
<input type="button" value="Open Marketing Expenses"
    onclick="BtnOpenMEFile()"
>
```

When clicked, a predefined document will be opened in the MapForceControl. The MakeAbsolutePath method creates an absolute path using the location of the script as a base
path.

```javascript
function BtnOpenMEFile()
{
    var strPath = MakeAbsolutePath("MarketingExpenses.mfd");
    var objDoc = objMapForceControl.Open(strPath );

    if (objDoc == null)
    {
        alert("Unable to locate MarketingExpenses.mfd at: " +
          objMapForceControl.BaseHref);
    }
}
```

### 18.5.2.1.3 Add Buttons for Code Generation

Although code-generation for the active document is available via menus, we want to have buttons that will generate code without asking the user for the location of the output. The method is similar to that used in the previous section.

First come the buttons:

```html
<input type="button" value="Generate XSLT" onclick="BtnGenerate( 0 )" >
<input type="button" value="Generate Java" onclick="BtnGenerate( 1 )" >
<input type="button" value="Generate C++" onclick="BtnGenerate( 2 )" >
<input type="button" value="Generate C#" onclick="BtnGenerate( 3 )" >
```

Then we provide the script that will generate the code into sub-folders of the currently defined default output folders.

```javascript
// ----------------------------------------------------------------------
// generate code for active document into language-specific subfolders of
// the current default output directory. No user interaction necessary.
function BtnGenerate(languageID)
{
    // get top-level object of automation interface
    var objApp = objMapForceControl.Application;

    // get the active document
    var objDocument = objApp.ActiveDocument;

    // retrieve object to set the generation output path
    var objOptions = objApp.Options;

    if ( objDocument == null )
    {
        alert( "no active document found" );
    }
    else
    {
        objOptions.XSLTDefaultOutputDirectory =
            objOptions.CodeDefaultOutputDirectory = GetDefaultOutputDirectory();

        if (languageID == 0)
        {
```
18.5.2.1.4 Connect to Custom Events

The example implements two event callbacks for MapForceControl custom events to show the principle:

```html
<!-- ----------------------------------------------------------- -->
<!--  custom event 'OnDocumentOpened' of MapForceControl object -->
<SCRIPT FOR="objMapForceControl" event="OnDocumentOpened( objDocument )"
LANGUAGE="javascript">
    // alert("Document '" + objDocument.Name + "' opened!");
</SCRIPT>

<!-- ----------------------------------------------------------- -->
<!--  custom event 'OnDocumentClosed' of MapForceControl object -->
<SCRIPT FOR="objMapForceControl" event="OnDocumentClosed( objDocument )"
LANGUAGE="javascript">
    // alert("Document '" + objDocument.Name + "' closed!");
</SCRIPT>
```

18.5.2.2 HTML Integration at Document Level

This example shows an integration of the MapForce control at document-level into a HTML page. The following topics are covered:
- Instantiate a MapForceControl ActiveX control object in HTML code
- Instantiate a MapForceControlDocument ActiveX control to allow editing a MapForce file
- Instantiate one MapForceControlPlaceHolder for a MapForceControl project window
- Instantiate one MapForceControlPlaceHolder to alternatively host one of the MapForce helper windows
- Create a simple custom toolbar for some heavy-used MapForce commands
- Add some more buttons that use the COM automation interface of MapForce
- Use event handlers to update command buttons

This example is available in its entirety in the file `MapForceActiveX_ApplicationLevel.htm` within the `<ApplicationFolder>\Examples\ActiveX\HTML\` folder of your MapForce installation.

18.5.2.2.1 Instantiate the MapForceControl

MapForceControl

The HTML `OBJECT` tag is used to create an instance of the MapForceControl. The Classid is that of MapForceControl. Width and height are set to 0 since we use this control as manager control without use for its user interface. The integration level is specified as a parameter within the `OBJECT` tag.

```html
<object id="objMapForceX" classid="clsid:A38637E9-5759-4456-A167-F01160CC22C1"
    width="0" height="0" VIEWASTEXT>
    <param name="IntegrationLevel" value="1">
</object>
```

18.5.2.2.2 Create Editor Window

The HTML `OBJECT` tag is used to embed an editing window. The additional custom parameter specifies that the control is to be initialized with a new empty document.

```html
<object id="objDoc1" classid="clsid:DFBB0871-DAFE-4502-BB66-08CEB7DF5255"
    width="600" height="500" VIEWASTEXT>
    <param name="NewDocument">
</object>
```

18.5.2.2.3 Create Project Window

The HTML `OBJECT` tag is used to create a MapForceControlPlaceHolder window. The parameter defines the placeholder to show the MapForce project window.

```html
<!-- ------------------------------- -->
<!-- create project window placeholder control. -->
<!-- initialize it with a project. -->
<object id="objProjectWindow" classid="clsid:FDEC3B04-05F2-427d-988C-F03A85DE53C2" width="200" height="200" VIEWASTEXT>
    <param name="PlaceholderWindowID" value="3">
</object>
```
ActiveX Integration

ActiveX Integration Examples 1687

</object>

18.5.2.2.4

Create Placeholder for Helper Windows

The MapForceControlPlaceHolder control is required to host an application helper window, see
also Integration at Document Level. In the code listing below, the HTML object tag is used to
instantiate a control that will host the Libraries window by default (PlaceholderWindowID is 0).
<!-- create helper window placeholder control.
-->
<!-- the editor with focus will automatically direct its -->
<!-- output to the appropriate helper window.
-->
<object id="objPlaceholderWindow" Classid="clsid:FDEC3B04-05F2-427d-988CF03A85DE53C2" width="200" height="200" VIEWASTEXT>
<param name="PlaceholderWindowID" value="0">
<param name="FileName" value="">
</object>

The example HTML page includes a few buttons which call the BtnHelperWindow method when
clicked. The BtnHelperWindow method reassigns the PlaceholderWindowID of the control, and
thus cause the ActiveX object to display a different helper window.
// specify which of the helper windows shall be shown in the placeholder
control.
function BtnHelperWindow(i_ePlaceholderWindowID)
{
objPlaceholderWindow.PlaceholderWindowID = i_ePlaceholderWindowID;
}

For the list of possible values of PlaceholderWindowID, see
MapForceControlPlaceholderWindow.

18.5.2.2.5

Create a Custom Toolbar

The example HTML page also includes a custom toolbar (intended as a replica of the MapForce
menu). The custom toolbar consists of buttons with images of MapForce commands, for example:
<button id="btnInsertXML" title="Insert XML Schema/File"
onclick="BtnDoCommand(32393)">
<img src="..\Images\ID_INSERT_XSD.gif" width="16" height="16" />
</button>
<button id="btnInsertDB" title="Insert Database"
onclick="BtnDoCommand(32389)">
<img src="..\Images\ID_INSERT_DATABASE.gif" width="16" height="16" />
</button>
<button id="btnInsertEDI" title="Insert EDI" onclick="BtnDoCommand(32390)">
<img src="..\Images\ID_INSERT_EDI.gif" width="16" height="16" />
</button>
<button id="btnInsertText" title="Insert Text file"

© 2019 Altova Gmb H

Altova MapForce 2019 Enterprise Edition


The names of button images correspond to the command ID numbers, see Command Reference. On clicking the button, the corresponding command ID is sent to the main control and executed:

```javascript
function BtnDoCommand(cmdID)
{
  objMapForceX.Exec (cmdID);
  msgtext.innerText = "Command " + cmdID + " performed."
}
```

### 18.5.2.2.6 Create More Buttons

In the example, we add some more buttons to show some automation code.

```html
<p>
<input type="button" value="New File" onclick="BtnNewFile(objDoc1)"/>
<input type="button" value="Save File" onclick="BtnSaveFile(objDoc1)"/>
<input type="text" title="Path" id="strPath" width="150" />
<input type="button" value="Open MarketingExpenses" onclick="BtnOpenMEFile(objDoc1)"/>
<input type="button" value="Open MapForce Sample Project" onclick="BtnOpenProjectFile(objDoc1)"/>
</p>
```

The corresponding JavaScript looks like this:

```javascript
// perform any command specified by cmdID.
// command routing includes application, active document and view.
function BtnDoCommand(cmdID)
{
  objMapForceX.Exec (cmdID);
  msgtext.innerText = "Command " + cmdID + " performed."
}
```

// perform any command specified by cmdID.
// command routing includes application, active document and view.
function BtnDoCommand(cmdID)
{
  objMapForceX.Exec (cmdID);
  msgtext.innerText = "Command " + cmdID + " performed."
}
```

```javascript
// open a new empty document in the specified document control window.
function BtnNewFile(objDocCtrl)
{
  objDocCtrl.OpenDocument(""");
  objDocCtrl.setActive();
}
```

// Saves the current file in the specified document control window.
function BtnSaveFile(objDocCtrl)
{
  if(objDocCtrl.Path.length > 0)
  objDocCtrl.SaveDocument();
  else
  {
    if(strPath.value.length > 0)
ActiveX Integration

ActiveX Integration Examples

18.5.2.2.7 Create Event Handler to Update Button Status

Availability of a command may vary with every mouse click or keystroke. The custom event
OnUpdateCmdUI of MapForceControl gives us an opportunity to update the enabled/disabled
state of buttons associated with MapForce commands. The method
MapForceControl.QueryStatus is used to query whether a command is enabled or not.

```javascript
<!-- custom event 'OnUpdateCmdUI" of MapForceControl object -->
function objMapForceX::OnUpdateCmdUI()
{
    if (document.readyState == "complete") // 'complete'
    {
        // update status of buttons
        // set activity status of simulated toolbar
        GenerateXSLT.disabled = ! (objDoc1.QueryStatus(32360) & 0x02); // not enabled
        GenerateJava.disabled = ! (objDoc1.QueryStatus(32358) & 0x02); // not enabled
    }
}
```
18.5.3 Java

MapForce ActiveX components can be accessed from Java code. Java integration is provided by the libraries listed below. These libraries are available in the folder `<ApplicationFolder>\Examples\JavaAPI` of your MapForce installation, after you have installed both MapForce and the MapForce Integration Package (see also Prerequisites).

- **AltovaAutomation.dll**: a JNI wrapper for Altova automation servers (in case of the 32-bit installation of MapForce)
- **AltovaAutomation_x64.dll**: a JNI wrapper for Altova automation servers (in case of the 64-bit installation of MapForce)
- **AltovaAutomation.jar**: Java classes to access Altova automation servers
- **MapForceActiveX.jar**: Java classes that wrap the MapForce ActiveX interface
- **MapForceActiveX_JavaDoc.zip**: a Javadoc file containing help documentation for the Java interface

**Note**: In order to use the Java ActiveX integration, the .dll and .jar files must be included in the Java class search path.

**Example Java project**

An example Java project is supplied with your product installation. You can test the Java project and modify and use it as you like. For more details, see Example Java Project.
Rules for mapping the ActiveX Control names to Java

For the documentation of ActiveX controls, see Object Reference. Note that the object naming conventions are slightly different in Java compared to other languages. Namely, the rules for mapping between the ActiveX controls and the Java wrapper are as follows:

- **Classes and class names**
  For every component of the MapForce ActiveX interface a Java class exists with the name of the component.

- **Method names**
  Method names on the Java interface are the same as used on the COM interfaces but start with a small letter to conform to Java naming conventions. To access COM properties, Java methods that prefix the property name with get and set can be used. If a property does not support write-access, no setter method is available. Example: For the IntegrationLevel property of the MapForceControl, the Java methods getIntegrationLevel and setIntegrationLevel are available.

- **Enumerations**
  For every enumeration defined in the ActiveX interface, a Java enumeration is defined with the same name and values.

- **Events and event handlers**
  For every interface in the automation interface that supports events, a Java interface with the same name plus 'Event' is available. To simplify the overloading of single events, a Java class with default implementations for all events is provided. The name of this Java class is the name of the event interface plus 'DefaultHandler'. For example: MapForceControl: Java class to access the application MapForceControlEvents: Events interface for the MapForceControl MapForceControlEventsDefaultHandler: Default handler for MapForceControlEvents

Exceptions to mapping rules

There are some exceptions to the rules listed above. These are listed below:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Changes in Java class</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapForceControlDocument, method New</td>
<td>Renamed to newDocument</td>
</tr>
<tr>
<td>MapForceControlDocument, method OpenDocument</td>
<td>Removed. Use the Open method</td>
</tr>
<tr>
<td>MapForceControlDocument, method SaveDocument</td>
<td>Removed. Use the Save method</td>
</tr>
</tbody>
</table>

This section

This section shows how some basic MapForce ActiveX functionality can be accessed from Java code. It is organized into the following sub-sections:

- **Example Java Project**
18.5.3.1  Example Java Project

The MapForce installation package contains an example Java project, located in the ActiveX Examples folder of the application folder: \Examples\ActiveX\Java\.

The Java example shows how to integrate the MapForceControl in a common desktop application created with Java. You can test it directly from the command line using the batch file BuildAndRun.bat, or you can compile and run the example project from within Eclipse. See below for instructions on how to use these procedures.

File list

The Java examples folder contains all the files required to run the example project. These files are listed below:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.classpath</td>
<td>Eclipse project helper file</td>
</tr>
<tr>
<td>.project</td>
<td>Eclipse project file</td>
</tr>
<tr>
<td>AltovaAutomation.dll</td>
<td>Java-COM bridge: DLL part (for the 32-bit installation)</td>
</tr>
<tr>
<td>AltovaAutomation_x64.dll</td>
<td>Java-COM bridge: DLL part (for the 64-bit installation)</td>
</tr>
<tr>
<td>AltovaAutomation.jar</td>
<td>Java-COM bridge: Java library part</td>
</tr>
<tr>
<td>BuildAndRun.bat</td>
<td>Batch file to compile and run example code from the command line prompt. Expects folder where Java Virtual Machine resides as parameter.</td>
</tr>
<tr>
<td>MapForceActiveX.jar</td>
<td>Java classes of the MapForce ActiveX control</td>
</tr>
<tr>
<td>MapForceActiveX_JavaDoc.zip</td>
<td>Javadoc file containing help documentation for the Java API</td>
</tr>
<tr>
<td>MapForceContainer.java</td>
<td>Java example source code</td>
</tr>
<tr>
<td>MapForceContainerEventHandler.java</td>
<td>Java example source code</td>
</tr>
<tr>
<td>MapForceTable.java</td>
<td>Java example source code</td>
</tr>
</tbody>
</table>

What the example does

The example places one MapForce document editor window, the MapForce project window, the MapForce library window and the MapForce validation window in an AWT frame window. It reads out the main menu defined for MapForce and creates an AWT menu with the same structure. You
can use this menu or the project window to open and work with files in the document editor.

You can modify the example in any way you like.

The following specific features are described in code listings:

- **Creating the ActiveX Controls**: Starts MapForce, which is registered as an automation server, or activates MapForce if it is already running.
- **Loading Data in the Controls**: Locates one of the example documents installed with MapForce and opens it.
- **Basic Event Handling**: Changes the view of all open documents to Text View. The code also shows how to iterate through open documents.
- **Menus**: Validates the active document and shows the result in a message box. The code shows how to use output parameters.
- **UI Update Event Handling**: Shows how to handle MapForce events.
- **Creating a MapForce Mapping Table**: Shows how to create a MapForce mapping table and prepare it for modal activation.

**Updating the path to the Examples folder**

Before running the provided sample, you may need to edit the `MapForceContainer.java` file. Namely, check that the following path refers to the actual folder where the MapForce example files are stored on your operating system:

```java
final String strExamplesFolder = System.getenv( "USERPROFILE" ) + "\Documents \Altova\MapForce2019\MapForceExamples\";
```

**Running the example from the command line**

To run the example from the command line:

1. Check that all prerequisites are met (see Prerequisites).
2. Open a command prompt window, change the current directory to the sample Java project folder, and type:

   ```bash
   buildAndRun.bat "<Path-to-the-Java-bin-folder>"
   ```

3. Press Enter.

   The Java source in `MapForceContainer.java` will be compiled and then executed.

**Compiling and running the example in Eclipse**

To import the sample Java project into Eclipse:

1. Check that all prerequisites are met (see Prerequisites).
2. On the File menu, click Import.
3. Select Existing Projects into Workspace, and browse for the Eclipse project file located at `<ApplicationFolder>\Examples\ActiveX\Java\`. Since you may not have write-access in this folder, it is recommended to select the Copy projects into
workspace check box on the Import dialog box.

To run the example application, right-click the project in Package Explorer and select the command Run as | Java Application.

Help for Java API classes is available through comments in code as well as the Javadoc view of Eclipse. To enable the Javadoc view in Eclipse, select the menu command Window | Show View | JavaDoc.

18.5.3.2 Creating the ActiveX Controls

The code listing below show how ActiveX controls can be created. The constructors will create the Java wrapper objects. Adding these Canvas-derived objects to a panel or to a frame will trigger the creation of the wrapped ActiveX object.

```java
/**
 * MapForce manager control - always needed
 */
public static MapForceControl mapForceControl = null;

/**
 * MapForceDocument editing control
 */
public static MapForceControlDocument mapForceDocument = null;

/**
 * Tool windows - MapForce place-holder controls
 */
private static MapForceControlPlaceHolder mapForceProjectToolWindow = null;
private static MapForceControlPlaceHolder mapForceValidationToolWindow = null;
private static MapForceControlPlaceHolder mapForceLibraryToolWindow = null;

// Create the MapForce ActiveX control; the parameter determines that we want
// to place document controls and place-holder controls individually.
// It gives us full control over the menu, as well.
mapForceControl = new MapForceControl(
    ICActiveXIntegrationLevel.ICActiveXIntegrationOnDocumentLevel.getValue(),
    false);

mapForceDocument = new MapForceControlDocument();
frame.add( mapForceDocument, BorderLayout.CENTER );

// Create a project window and open the sample project in it
mapForceProjectToolWindow = new MapForceControlPlaceHolder(
    MapForceControlPlaceholderWindow.MapForceXProjectWindow.getValue(),
    strExamplesFolder + "MapForceExamples.mfp" );
mapForceProjectToolWindow.setPreferredSize( new Dimension( 200, 200 ) );
```
18.5.3.3 **Loading Data in the Controls**

The code listing below show how data can be loaded in the ActiveX controls.

```
// Locate samples installed with the product.
final String strExamplesFolder = System.getenv( "USERPROFILE" ) + "\Documents\Altova\MapForce2019\MapForceExamples\";
mapForceProjectToolWindow = new MapForceControlPlaceHolder( MapForceControlPlaceHolderWindow.MapForceXProjectWindow.getValue(), strExamplesFolder + "MapForceExamples.mfp" ) ;
```

18.5.3.4 **Basic Event Handling**

The code listing below shows how basic events can be handled. When calling the `MapForceControl.open` method, or when trying to open a file via the menu or Project tree, the `onOpenedOrFocused` event is sent to the attached event handler. The basic handling for this event is opening the file by calling the `MapForceDocumentControl.open` method.

```
// Open the Marketing file when button is pressed
btnMarkExp.addActionListener( new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            // Instruct the Document control to open the file - avoid calling
            // the open method of MapForceControl (see help)
            mapForceDocument.open( strExamplesFolder + "MarketingExpenses.mfd" );
            mapForceDocument.requestFocusInWindow();
        } catch (AutomationException e1) {
            e1.printStackTrace();
        }
    }
});

public void onOpenedOrFocused( String i_strFileName, boolean i_bOpenWithThisControl, boolean i_bFileAlreadyOpened ) throws AutomationException {
    if ( !i_bFileAlreadyOpened ) {
        // This is basically an SDI interface, so open the file in the already
        // existing document control
        try {
            MapForceContainer.mapForceDocument.open( i_strFileName );
            MapForceContainer.mapForceDocument.requestFocusInWindow();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```
### 18.5.3.5 Menus

The code listing below shows how menu items can be created. Each `MapForceCommand` object gets a corresponding `MenuItem` object, with the `ActionCommand` set to the ID of the command. The actions generated by all menu items are handled by the same function, which can perform specific handlings (like reinterpreting the closing mechanism) or can delegate the execution to the `MapForceControl` object by calling its `exec` method. The `menuMap` object that is filled during menu creation is used later (see section UI Update Event Handling).

```java
// Load the file menu when the button is pressed
btnMenu.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            // Create the menubar that will be attached to the frame
            MenuBar mb = new MenuBar();
            // Load the main menu's first item - the File menu
            MapForceCommand xmlSpyMenu = mapForceControl.getMainMenu().getSubCommands().getItem(0);
            // Create Java menu items from the Commands objects
            Menu fileMenu = new Menu();
            handlerObject.fillMenu(fileMenu, xmlSpyMenu.getSubCommands());
            fileMenu.setLabel(xmlSpyMenu.getLabel().replace("&", ";");
            mb.add(fileMenu);
            frame.setMenuBar(mb);
            frame.validate();
        } catch (AutomationException e1) {
            e1.printStackTrace();
        }
        // Disable the button when the action has been performed
        ((AbstractButton) e.getSource()).setEnabled(false);
    }
});

/**
 * Populates a menu with the commands and submenus contained in an MapForceCommands object
 */
public void fillMenu(Menu newMenu, MapForceCommands mapForceMenu) throws AutomationException {
    // For each command/submenu in the mapForceMenu
    for (int i = 0; i < mapForceMenu.getCount(); ++i) {
        MapForceCommand mapForceCommand = mapForceMenu.getItem(i);
        if (mapForceCommand.getIsSeparator())
            newMenu.addSeparator();
        else
            MapForceCommands subCommands = mapForceCommand.getSubCommands();
            // Is it a command (leaf), or a submenu?
            if (!subCommands.isNull() || subCommands.getCount() == 0)
                // Command -> add it to the menu, set its ActionCommand to its ID
                // and store it in the menuMap
                MenuItem mi = new MenuItem(mapForceCommand.getLabel().replace("&", ";");
```
```java
mi.setActionCommand("" + mapForceCommand.getID());
mi.addActionListener(this);
newMenu.add(mi);
menuMap.put(mapForceCommand.getID(), mi);
}
else
{
// Submenu -> create submenu and repeat recursively
Menu newSubMenu = new Menu();
fillMenu(newSubMenu, subCommands);
newSubMenu.setLabel(mapForceCommand.getLabel().replace("&", "");
newMenu.add(newSubMenu);
}
}
/**
 * Action handler for the menu items
 * Called when the user selects a menu item; the item's action command corresponds to the command table for MapForce
 */
public void actionPerformed(ActionEvent e)
{
try
{
int iCmd = Integer.parseInt(e.getActionCommand());
// Handle explicitly the Close commands
switch (iCmd)
{
    case 57602: // Close
    case 34050: // Close All
        MapForceContainer.initMapForceDocument();
        break;
    default:
        MapForceContainer.mapForceControl.exec(iCmd);
        break;
}
catch (Exception ex)
{
    ex.printStackTrace();
}
}
```

# 18.5.3.6 UI Update Event Handling

The code listing below shows how a UI-Update event handler can be created.

```java
/**
 * Call-back from the MapForceControl.
 * Called to enable/disable commands
 */
@Override
public void onUpdateCmdUI() throws AutomationException
{
```
// A command should be enabled if the result of queryStatus contains the 
Supported (1) and Enabled (2) flags
for ( java.util.Map.Entry<Integer, MenuItem> pair : menuMap.entrySet() )
    pair.getValue().setEnabled( MapForceContainer.mapForceControl.queryStatus( 
        pair.getKey() ) > 2 );
/**
 * Call-back from the MapForceControl.
 * Usually called while enabling/disabling commands due to UI updates
 */
@Override
public boolean onIsActiveEditor( String i_strFilePath ) throws AutomationException
{
    try
    {
        return MapForceContainer.mapForceDocument.getDocument().getFullName().equalsIgnoreCase( 
            i_strFilePath );
    } catch ( Exception e ) {
        return false;
    }
}

18.5.3.7 Listing the Properties of a MapForce Mapping

The listing below shows how a Mapping object in MapForce can be loaded as a table and 
prepared for modal activation.

//access MapForce Java-COM bridge
import com.altova.automation.MapForce.*;
//access AWT and Swing components
import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;

/**
 * A simple example of a table control loading the structure from a Mapping 
object.
 * The class receives an Mapping object, loads its components in a JTable, 
and prepares 
for modal activation.
 * Feel free to modify and extend this sample.
 * @author Altova GmbH
 */
class MapForceTable extends JDialog
{
    /**
     * The table control
private JTable myTable;

/**
 * Constructor that prepares the modal dialog containing the filled table control
 * @param mapping The data to be displayed in the table
 * @param parent Parent frame
 */
public MapForceTable( Mapping mapping, Frame parent )
{
    // Construct the modal dialog
    super( parent, "MapForce component table", true );
    // Build up the tree
    fillTable( mapping );
    // Arrange controls in the dialog
    getContentPane( new JScrollPane( myTable ) );
}

/**
 * Loads the components of a Mapping object in the table
 * @param mapping Source data
 */
private void fillTable( Mapping mapping)
{
    try
    {
        // count how many Instance components do we have
        int size = 0;
        for (Component comp : mapping.getComponents())
            if ( comp.getUsageKind() == ENUMComponentUsageKind.eComponentUsageKind_Instance )
                ++size;

        // Prepare data
        final String[] columnNames = { "Component", "Has inputs", "Has outputs", "Input file", "Output file", "Schema" };
        final Object[][] data = new Object[size][ 7 ] ;
        int index = 0;
        for (Component comp : mapping.getComponents())
            if ( comp.getUsageKind() == ENUMComponentUsageKind.eComponentUsageKind_Instance )
                { 
                    int i = 0;
                    data[ index ][ i++ ] = comp.getName() ;
                    data[ index ][ i++ ] = new Boolean( comp.getHasIncomingConnections() );
                    data[ index ][ i++ ] = new Boolean( comp.getHasOutgoingConnections() );
                    data[ index ][ i++ ] = comp.getInputInstanceFile();
                    data[ index ][ i++ ] = comp.getOutputInstanceFile();
                    data[ index++ ][ i ] = comp.getSchema();
                }
        // Set up table
        myTable = new JTable( new AbstractTableModel() {
            public String getColumnName(int col) { return columnNames[col]; } 
            public int getRowCount() { return data.length; } 
            public int getColumnCount() { return columnNames.length; } 
        });
    } catch (Exception e) {}
    public Object getValueAt(int row, int col) { return data[row][col]; }
    public boolean isCellEditable(int row, int col) { return false; }
    public Class getColumnClass(int c) { return getValueAt(0, c).getClass(); }
}

    // Set width
    for ( index = 0 ; index < columnNames.length ; ++index )
        myTable.getColumnModel().getColumn( index ).setMinWidth( 80 );
    myTable.getColumnModel().getColumn( 5 ).setMinWidth( 400 );

18.5.4 VB.NET

Source code which illustrates integration of MapForceControl into a VB.NET application can be found in the folder <ApplicationFolder>\Examples\ActiveX\VB.NET of your MapForce installation. The solution consists of three windows, as follows:

1. **MainWindow.vb** - the main document window, which also includes a basic application menu.

2. **LibraryWindow.vb** - the Library window. The contents of this window is populated by a Placeholder control which has the **PlaceholderWindowID** property set to 0 (this value instructs the control to display specifically the Library window).

3. **OutputWindow.vb** - the Messages (Output) window. The contents of this window is
populated by a Placeholder control which has the PlaceholderWindowID property set to 2 (this value instructs the control to display specifically the Output window).

Before attempting to build and run this solution, note the following steps:

**Step 1: Check the prerequisites**
For the list of prerequisites, see [Prerequisites](#).

**Step 2: Copy the sample to a directory where you have write permissions**
To avoid running Visual Studio as an Administrator, copy the source code to a directory where you have write permissions, instead of running it from the default location.

**Step 3: Set the build platform**
- Create a build platform configuration that matches the platform under which you want to build (x86, x64). Here is how you can create the build configuration:
  a. Right-click the solution in Visual Studio, and select **Configuration Manager**.
  b. Under **Active solution platform**, select **New...** and then select the x86 or x64 configuration (in this example, **x86**).
You are now ready to build and run the solution in Visual Studio. Remember to build using the configuration that matches your target platform (x86, x64); otherwise, runtime errors might occur.
18.6 Command Reference

This section lists the names and identifiers of all menu commands that are available within MapForce. Every sub-section lists the commands from the corresponding top-level menu of MapForce. The command tables are organized as follows:

- The "Menu Item" column shows the command's menu text as it appears in MapForce, to make it easier for you to identify the functionality behind the command.
- The "Command Name" column specifies the string that can be used to get an icon with the same name from ActiveXImages folder of the MapForce installation directory.
- The "ID" column shows the numeric identifier of the column that must be supplied as argument to methods which execute or query this command.

To execute a command, use the MapForceControl.Exec or the MapForceControlDocument.Exec methods. To query the status of a command, use the MapForceControl.QueryStatus or MapForceControlDocument.QueryStatus methods.

Depending on the edition of MapForce you have installed, some of these commands might not be supported.

18.6.1 "File" Menu

The "File" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>New...</td>
<td>ID_FILE_NEW</td>
<td>57600</td>
</tr>
<tr>
<td>Open...</td>
<td>ID_FILE_OPEN</td>
<td>57601</td>
</tr>
<tr>
<td>Save</td>
<td>ID_FILE_SAVE</td>
<td>57603</td>
</tr>
<tr>
<td>Save As...</td>
<td>ID_FILE_SAVE_AS</td>
<td>57604</td>
</tr>
<tr>
<td>Save All</td>
<td>ID_FILE_SAVEALL</td>
<td>32377</td>
</tr>
<tr>
<td>Reload</td>
<td>IDC_FILE_RELOAD</td>
<td>32467</td>
</tr>
<tr>
<td>Close</td>
<td>ID_WINDOW_CLOSE</td>
<td>32453</td>
</tr>
<tr>
<td>Close All</td>
<td>ID_WINDOW_CLOSEALL</td>
<td>32454</td>
</tr>
<tr>
<td>Print...</td>
<td>ID_FILE_PRINT</td>
<td>57607</td>
</tr>
<tr>
<td>Print Preview</td>
<td>ID_FILE_PRINT_PREVIEW</td>
<td>57609</td>
</tr>
<tr>
<td>Print Setup...</td>
<td>ID_FILE_PRINT_SETUP</td>
<td>57606</td>
</tr>
<tr>
<td>Validate Mapping</td>
<td>ID_MAPPING_VALIDATE</td>
<td>32347</td>
</tr>
<tr>
<td>Mapping Settings</td>
<td>ID_MAPPING_SETTINGS</td>
<td>32396</td>
</tr>
<tr>
<td>Generate Code in Selected Language</td>
<td>ID_FILE_GENERATE_SELECTED_COD</td>
<td>32362</td>
</tr>
</tbody>
</table>
### 18.6.2 "Edit" Menu

The "Edit" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undo</td>
<td>ID_EDIT_UNDO</td>
<td>57643</td>
</tr>
<tr>
<td>Redo</td>
<td>ID_EDIT_REDO</td>
<td>57644</td>
</tr>
<tr>
<td>Find...</td>
<td>ID_EDIT_FIND</td>
<td>57636</td>
</tr>
<tr>
<td>Find Next</td>
<td>ID_EDIT_FINDNEXT</td>
<td>32349</td>
</tr>
<tr>
<td>Find Previous</td>
<td>ID_EDIT_FINDPREV</td>
<td>32350</td>
</tr>
<tr>
<td>Cut</td>
<td>ID_EDIT_CUT</td>
<td>57635</td>
</tr>
<tr>
<td>Copy</td>
<td>ID_EDIT_COPY</td>
<td>57634</td>
</tr>
<tr>
<td>Paste</td>
<td>ID_EDIT_PASTE</td>
<td>57637</td>
</tr>
<tr>
<td>Delete</td>
<td>ID_EDIT_CLEAR</td>
<td>57632</td>
</tr>
<tr>
<td>Select All</td>
<td>ID_EDIT_SELECT_ALL</td>
<td>57642</td>
</tr>
</tbody>
</table>
18.6.3 "Insert" Menu

The "Insert" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Schema/File...</td>
<td>ID_INSERT_XSD</td>
<td>32393</td>
</tr>
<tr>
<td>Database...</td>
<td>ID_INSERT_DATABASE</td>
<td>32389</td>
</tr>
<tr>
<td>EDI...</td>
<td>ID_INSERT_EDI</td>
<td>32390</td>
</tr>
<tr>
<td>Text File...</td>
<td>ID_INSERT_TXT</td>
<td>32392</td>
</tr>
<tr>
<td>Web Service Function...</td>
<td>ID_INSERT_WEBSERVICE_FUNCTION</td>
<td>32319</td>
</tr>
<tr>
<td>Excel 2007+ File...</td>
<td>ID_INSERT_EXCEL</td>
<td>32376</td>
</tr>
<tr>
<td>XBRL Document...</td>
<td>ID_INSERT_XBRL</td>
<td>32469</td>
</tr>
<tr>
<td>JSON Schema/File...</td>
<td>ID_INSERT_JSON</td>
<td>32531</td>
</tr>
<tr>
<td>Insert Input...</td>
<td>ID_FUNCTION_INSERT_INPUT</td>
<td>32383</td>
</tr>
<tr>
<td>Insert Output...</td>
<td>ID_FUNCTION_INSERT_OUTPUT</td>
<td>32402</td>
</tr>
<tr>
<td>Constant...</td>
<td>ID_INSERT_CONSTANT</td>
<td>32388</td>
</tr>
<tr>
<td>Variable...</td>
<td>ID_INSERT_VARIABLE</td>
<td>32500</td>
</tr>
<tr>
<td>Join</td>
<td>ID_INSERT_JOIN</td>
<td>32581</td>
</tr>
<tr>
<td>Sort: Nodes/Rows</td>
<td>ID_INSERT_SORT</td>
<td>32444</td>
</tr>
<tr>
<td>Filter: Nodes/Rows</td>
<td>ID_INSERT_FILTER</td>
<td>32391</td>
</tr>
<tr>
<td>SQL-WHERE/ORDER</td>
<td>ID_INSERT_SQLWHERE_CONDITION</td>
<td>32351</td>
</tr>
<tr>
<td>Value-Map</td>
<td>ID_INSERT_VALUEMAP</td>
<td>32354</td>
</tr>
<tr>
<td>IF-Else Condition</td>
<td>ID_INSRT_CONDITION</td>
<td>32394</td>
</tr>
<tr>
<td>Exception</td>
<td>ID_INSERT_EXCEPTION</td>
<td>32311</td>
</tr>
</tbody>
</table>

18.6.4 "Project" Menu

The "Project" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reload Project</td>
<td>ID_PROJECT_RELOAD</td>
<td>32476</td>
</tr>
</tbody>
</table>
### Menu item
<table>
<thead>
<tr>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close Project</td>
<td>ID_FILE_CLOSEPROJECT 32355</td>
</tr>
<tr>
<td>Save Project</td>
<td>ID_FILE_SAVEPROJECT 32378</td>
</tr>
<tr>
<td>Add Files to Project...</td>
<td>ID_PROJECT_ADDFILESTOPROJECT 32420</td>
</tr>
<tr>
<td>Add Active File to Project</td>
<td>ID_PROJECT_ADDACTIVEFILETOPROJECT 32419</td>
</tr>
<tr>
<td>Create Folder...</td>
<td>ID_PROJECT_CREATE_FOLDER 32310</td>
</tr>
<tr>
<td>Open Mapping</td>
<td>ID_PROJECT_OPEN_MAPPING 32307</td>
</tr>
<tr>
<td>Create Mapping for Operation...</td>
<td>ID_PROJECT_CREATE_MAPPING_FOR_OPERATION 32399</td>
</tr>
<tr>
<td>Add Mapping File for Operation...</td>
<td>ID_PROJECT_ADD_MAPPING 32309</td>
</tr>
<tr>
<td>Insert Web Service...</td>
<td>ID_PROJECT_INSERT_WEBSERVICE 32306</td>
</tr>
<tr>
<td>Open File in XMLSpy</td>
<td>ID_PROJECT_OPEN_IN_XMLSPY 32305</td>
</tr>
<tr>
<td>Generate Code for Entire Project</td>
<td>ID_PROJECT_GENERATE_ALL 32303</td>
</tr>
<tr>
<td>XSLT 1.0</td>
<td>ID_PROJECT_GENERATEXSLTCODE_ENTIRE 32408</td>
</tr>
<tr>
<td>XSLT 2.0</td>
<td>ID_PROJECT_GENERATEXSLT2CODE_ENTIRE 32409</td>
</tr>
<tr>
<td>XQuery</td>
<td>ID_PROJECT_GENERATEXQUERYCODE_ENTIRE 32410</td>
</tr>
<tr>
<td>Java</td>
<td>ID_PROJECT_GENERATEJAVACODE_ENTIRE 32411</td>
</tr>
<tr>
<td>C# (Sharp)</td>
<td>ID_PROJECT_GENERATECSCODE_ENTIRE 32412</td>
</tr>
<tr>
<td>C++</td>
<td>ID_PROJECT_GENERATECPPCODE_ENTIRE 32413</td>
</tr>
<tr>
<td>Properties</td>
<td>ID_PROJECT_PROPERTIES 32404</td>
</tr>
<tr>
<td>Recent Project</td>
<td>ID_FILE_MRU_PROJECT1 32364</td>
</tr>
</tbody>
</table>

## 18.6.5 "Component" Menu

The "Component" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Root Element...</td>
<td>ID_COMPONENT_CHANGEROOTELEM 32334</td>
<td></td>
</tr>
<tr>
<td>Menu item</td>
<td>Command name</td>
<td>ID</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>ENT</td>
<td>ID_COMPONENT_EDIT_SCHEMA</td>
<td>32337</td>
</tr>
<tr>
<td>Edit Schema Definition in XMLSpy</td>
<td>ID_COMPONENT_EDIT_MFT</td>
<td>32301</td>
</tr>
<tr>
<td>Edit FlexText Configuration</td>
<td>ID_COMPONENT_EDIT_MFT</td>
<td>32301</td>
</tr>
<tr>
<td>Add/Remove/Edit Database Objects...</td>
<td>ID_COMPONENT_SELECTTABLES</td>
<td>32346</td>
</tr>
<tr>
<td>Create Mapping to EDI X12 997</td>
<td>ID_COMPONENT_CREATE_MAPPING_TO_997</td>
<td>32483</td>
</tr>
<tr>
<td>Create Mapping to EDI X12 999</td>
<td>ID_COMPONENT_CREATE_MAPPING_TO_999</td>
<td>32484</td>
</tr>
<tr>
<td>Refresh</td>
<td>IDC_COMMAND_REFRESH_COMPONENT</td>
<td>32373</td>
</tr>
<tr>
<td>Add Duplicate Input Before</td>
<td>ID_COMPONENT_CREATE_DUPLICATE_ICON_BEFOR</td>
<td>32503</td>
</tr>
<tr>
<td>Add Duplicate Input After</td>
<td>ID_COMPONENT_CREATE_DUPLICATE_ICON</td>
<td>32335</td>
</tr>
<tr>
<td>Remove Duplicate</td>
<td>ID_COMPONENT_REMOVE_DUPLICATE_ICON</td>
<td>32339</td>
</tr>
<tr>
<td>Add Comment Before</td>
<td>ID_COMPONENT_ADD_COMMENT_BEFOR</td>
<td>32518</td>
</tr>
<tr>
<td>Add Comment After</td>
<td>ID_COMPONENT_ADD_COMMENT_AFTER</td>
<td>32519</td>
</tr>
<tr>
<td>Add Processing Instruction Before...</td>
<td>ID_COMPONENT_ADD_PI_BEFOR</td>
<td>32520</td>
</tr>
<tr>
<td>Add Processing Instruction After...</td>
<td>ID_COMPONENT_ADD_PI_AFTOR</td>
<td>32521</td>
</tr>
<tr>
<td>Edit Processing Instruction Name...</td>
<td>ID_COMPONENT_EDIT_PI</td>
<td>32524</td>
</tr>
<tr>
<td>Delete Comment/Processing Instruction</td>
<td>ID_COMPONENT_REMOVE_COMMENT_PI</td>
<td>32522</td>
</tr>
<tr>
<td>Write Content as CDATA Section</td>
<td>ID_COMPONENT_TOGGLE_CDATAB</td>
<td>32525</td>
</tr>
<tr>
<td>Database Table Actions</td>
<td>ID_POPUP_DATABASETABLEACTIONS</td>
<td>32400</td>
</tr>
<tr>
<td>Query Database...</td>
<td>ID_QUERY_DATABASE</td>
<td>32341</td>
</tr>
<tr>
<td>Align Tree Left</td>
<td>ID_COMPONENT_LEFTALIGNTREE</td>
<td>32338</td>
</tr>
<tr>
<td>Align Tree Right</td>
<td>ID_COMPONENT_RIGHTALIGNTREE</td>
<td>32340</td>
</tr>
<tr>
<td>Properties</td>
<td>ID_COMPONENT_PROPERTIES</td>
<td>32336</td>
</tr>
</tbody>
</table>
18.6.6 "Connection" Menu

The "Connection" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Connect Matching Children</td>
<td>ID_CONNECTION_AUTOCONNECTCHILDREN</td>
<td>32342</td>
</tr>
<tr>
<td>Settings for Connect Matching Children</td>
<td>ID_CONNECTION_SETTINGS</td>
<td>32344</td>
</tr>
<tr>
<td>Connect Matching Children...</td>
<td>ID_CONNECTION_MAPCHILDELEMENTS</td>
<td>32343</td>
</tr>
<tr>
<td>Target Driven (Standard)</td>
<td>ID_POPUP_NORMALCONNECTION</td>
<td>32401</td>
</tr>
<tr>
<td>Copy-All (Copy Child Items)</td>
<td>ID_POPUP_NORMALWITHCHILDREN_CONNECTION</td>
<td>32460</td>
</tr>
<tr>
<td>Source Driven (Mixed Content)</td>
<td>ID_POPUP_ORDERBYSOURCECONNECTION</td>
<td>32403</td>
</tr>
<tr>
<td>Properties</td>
<td>ID_POPUP_CONNECTION_SETTINGS</td>
<td>32398</td>
</tr>
</tbody>
</table>

18.6.7 "Function" Menu

The "Function" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create User-Defined Function...</td>
<td>ID_FUNCTION_CREATE_EMPTY</td>
<td>32380</td>
</tr>
<tr>
<td>Create User-Defined Function from Selection...</td>
<td>ID_FUNCTION_CREATE_FROM_SELECTION</td>
<td>32381</td>
</tr>
<tr>
<td>Function Settings</td>
<td>ID_FUNCTION_SETTINGS</td>
<td>32387</td>
</tr>
<tr>
<td>Remove Function</td>
<td>ID_FUNCTION_REMOVE</td>
<td>32385</td>
</tr>
<tr>
<td>Insert Input...</td>
<td>ID_FUNCTION_INSERT_INPUT</td>
<td>32383</td>
</tr>
<tr>
<td>Insert Output...</td>
<td>ID_FUNCTION_INSERT_OUTPUT</td>
<td>32402</td>
</tr>
</tbody>
</table>

18.6.8 "Output" Menu

The "Output" menu has the following commands:
### 18.6.9 "Debug" Menu

The "Debug" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Debugging</td>
<td>ID_DEBUG_START</td>
<td>32540</td>
</tr>
<tr>
<td>Stop Debugging</td>
<td>ID_DEBUG_STOP</td>
<td>32541</td>
</tr>
<tr>
<td>Step Into</td>
<td>ID_DEBUG_STEP_INTO</td>
<td>32545</td>
</tr>
<tr>
<td>Step Over</td>
<td>ID_DEBUG_STEP_OVER</td>
<td>32551</td>
</tr>
<tr>
<td>Step Out</td>
<td>ID_DEBUG_STEP_OUT</td>
<td>32552</td>
</tr>
<tr>
<td>Minimal Step</td>
<td>ID_DEBUG_STEP_NEXT_TRACE</td>
<td>32554</td>
</tr>
</tbody>
</table>
18.6.10 "View" Menu

The "View" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Annotations</td>
<td>ID_SHOW_ANNOTATION</td>
<td>32435</td>
</tr>
<tr>
<td>Show Types</td>
<td>ID_SHOW_TYPES</td>
<td>32437</td>
</tr>
<tr>
<td>Show Library in Function Header</td>
<td>ID_VIEW_SHOWLIBRARYINFUNCIONHEADER</td>
<td>32448</td>
</tr>
<tr>
<td>Show Tips</td>
<td>ID_SHOW_TIPS</td>
<td>32436</td>
</tr>
<tr>
<td>XBRL Display Options</td>
<td>ID_VIEW_XBRL_DISPLAY_OPTIONS</td>
<td>32473</td>
</tr>
<tr>
<td>Show Selected Component Connectors</td>
<td>ID_VIEW_AUTOHIGHLIGHTCOMPONENTCONNECTIONS</td>
<td>32443</td>
</tr>
<tr>
<td>Show Connectors from Source to Target</td>
<td>ID_VIEW_RECURSIVEAUTOHIGHLIGHT</td>
<td>32447</td>
</tr>
<tr>
<td>Zoom...</td>
<td>ID_VIEW_ZOOM</td>
<td>32451</td>
</tr>
<tr>
<td>Back</td>
<td>ID_CMD_BACK</td>
<td>32479</td>
</tr>
<tr>
<td>Forward</td>
<td>ID_CMD_FORWARD</td>
<td>32478</td>
</tr>
<tr>
<td>Status Bar</td>
<td>ID_VIEW_STATUS_BAR</td>
<td>59393</td>
</tr>
<tr>
<td>Library Window</td>
<td>ID_VIEW_LIBRARY_WINDOW</td>
<td>32445</td>
</tr>
<tr>
<td>Messages</td>
<td>ID_VIEW_VALIDATION_OUTPUT</td>
<td>32450</td>
</tr>
<tr>
<td>Overview</td>
<td>ID_VIEW_OVERVIEW_WINDOW</td>
<td>32446</td>
</tr>
<tr>
<td>Project Window</td>
<td>ID_VIEW_PROJECT_WINDOW</td>
<td>32302</td>
</tr>
<tr>
<td>Values</td>
<td>ID_DEBUG_VIEW_VALUES_WINDOW</td>
<td>32544</td>
</tr>
<tr>
<td>Context</td>
<td>ID_DEBUG_VIEW_CONTEXT_WINDOW</td>
<td>32546</td>
</tr>
<tr>
<td>Breakpoints</td>
<td>ID_DEBUG_VIEW_DEBUGPOINTS_WINDOW</td>
<td>32547</td>
</tr>
</tbody>
</table>

18.6.11 "Tools" Menu

The "Tools" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Resources</td>
<td>IDC_GLOBALRESOURCES</td>
<td>37401</td>
</tr>
<tr>
<td>Menu item</td>
<td>Command name</td>
<td>ID</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>&lt;plugin not loaded&gt;</td>
<td>IDC_GLOBALRESOURCES_SUBMENU ENTRY1</td>
<td>37408</td>
</tr>
<tr>
<td>Create Reversed Mapping</td>
<td>ID_CREATE_REVERSED_MAPPING</td>
<td>32489</td>
</tr>
<tr>
<td>Customize...</td>
<td>IDC_APP_TOOLS-customize</td>
<td>32959</td>
</tr>
<tr>
<td>Options...</td>
<td>ID_TOOLS_OPTIONS</td>
<td>32441</td>
</tr>
</tbody>
</table>

### 18.6.12 "Window" Menu

The "Window" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascade</td>
<td>ID_WINDOW.Cascade</td>
<td>57650</td>
</tr>
<tr>
<td>Tile Horizontal</td>
<td>ID_WINDOW TILE_HORIZ</td>
<td>57651</td>
</tr>
<tr>
<td>Tile Vertical</td>
<td>ID_WINDOW TILE VERT</td>
<td>57652</td>
</tr>
</tbody>
</table>

### 18.6.13 "Help" Menu

The "Help" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents...</td>
<td>IDC_HELP CONTENTS</td>
<td>32966</td>
</tr>
<tr>
<td>Index...</td>
<td>IDC_HELP_INDEX</td>
<td>32967</td>
</tr>
<tr>
<td>Search...</td>
<td>IDC_HELP SEARCH</td>
<td>32969</td>
</tr>
<tr>
<td>Software Activation...</td>
<td>IDC_ACTIVATION</td>
<td>32970</td>
</tr>
<tr>
<td>Order Form...</td>
<td>IDC_OPEN_ORDER_PAGE</td>
<td>32971</td>
</tr>
<tr>
<td>Registration...</td>
<td>IDC_REGISTRATION</td>
<td>32972</td>
</tr>
<tr>
<td>Check for Updates...</td>
<td>IDC_CHECK FOR UPDATES</td>
<td>32973</td>
</tr>
<tr>
<td>MapForce Product Comparison...</td>
<td>IDC_PRODUCT_COMPARISON</td>
<td>32955</td>
</tr>
<tr>
<td>Support Center...</td>
<td>IDC_OPEN_SUPPORT_PAGE</td>
<td>32961</td>
</tr>
<tr>
<td>FAQ on the Web...</td>
<td>IDC_OPEN_FAQ_PAGE</td>
<td>32962</td>
</tr>
<tr>
<td>Download Components and Free Tools</td>
<td>IDC_OPEN COMPONENTS PAGE</td>
<td>32963</td>
</tr>
<tr>
<td>MapForce on the Internet...</td>
<td>IDC_OPEN_HOME_PAGE</td>
<td>32964</td>
</tr>
<tr>
<td>Menu item</td>
<td>Command name</td>
<td>ID</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>MapForce Training...</td>
<td>IDC_OPEN_TRAINING_PAGE</td>
<td>32965</td>
</tr>
<tr>
<td>About MapForce...</td>
<td>ID_APPABOUT</td>
<td>57664</td>
</tr>
</tbody>
</table>
18.7 Object Reference

Objects:
MapForceCommand
MapForceCommands
MapForceControl
MapForceControlDocument
MapForceControlPlaceHolder

To give access to standard MapForce functionality, objects of the MapForce automation interface can be accessed as well. See MapForceControl.Application, MapForceControlDocument.Document and MapForceControlPlaceHolder.Project for more information.

18.7.1 MapForceCommand

Properties:
ID
Label
Name
IsSeparator
ToolTip
StatusText
Accelerator
SubCommands

Description:
A command object can be one of the following: an executable command, a command container (for example, a menu, submenu, or toolbar), or a menu separator. To determine what kind of information is stored in the current Command object, query its ID, IsSeparator, and SubCommands properties, as follows.

<table>
<thead>
<tr>
<th>The Command object is...</th>
<th>When...</th>
</tr>
</thead>
<tbody>
<tr>
<td>An executable command</td>
<td>• ID is greater than zero</td>
</tr>
<tr>
<td></td>
<td>• IsSeparator is false</td>
</tr>
<tr>
<td></td>
<td>• SubCommands is empty</td>
</tr>
<tr>
<td>A command container</td>
<td>• ID is zero</td>
</tr>
<tr>
<td></td>
<td>• IsSeparator is true</td>
</tr>
<tr>
<td></td>
<td>• SubCommands contains a collection of Command objects.</td>
</tr>
<tr>
<td>Separator</td>
<td>• ID is zero</td>
</tr>
<tr>
<td></td>
<td>• IsSeparator is true</td>
</tr>
</tbody>
</table>
18.7.1.1 Accelerator

Property: Accelerator as string

Description:
Returns the accelerator key defined for the command. If the command has no accelerator key assigned, this property returns the empty string. The string representation of the accelerator key has the following format:

[ALT+][CTRL+][SHIFT+]key

Where key is converted using the Windows Platform SDK function **GetKeyNameText**.

18.7.1.2 ID

Property: ID as long

Description:
This property gets the unique identifier of the command. A command's ID is required to execute the command (using **Exec**) or query its status (using **QueryStatus**). If the command is a container for other commands (for example, a top-level menu), or a separator, the ID is 0.

18.7.1.3 IsSeparator

Property: IsSeparator as boolean

Description:
The property returns true if the command object is a menu separator; false otherwise. See also **Command**.

18.7.1.4 Label

Property: Label as string

Description:
This property gets the text of the command as it is displayed in the graphical user interface of MapForce. If the command is a separator, "Label" is an empty string. This property may also return an empty string for some toolbar commands that do not have any GUI text associated with them.

18.7.1.5 Name

Property: Name as string

Description:
This property gets the unique name of the command. This value can be used to get the icon file of the command, where it is available. The available icon files can be found in the folder...
18.7.1.6 StatusText

Property: Label as string

Description:
The status text is the text shown in the status bar of MapForce when the command is selected. It applies only to command objects that are not separators or containers of other commands; otherwise, the property is an empty string.

18.7.1.7 SubCommands

Property: SubCommands as Commands

Description:
The SubCommands property gets the collection of Command objects that are sub-commands of the current command. The property is applicable only to commands that are containers for other commands (menus, submenus, or toolbars). Such container commands have the ID set to 0, and the IsSeparator property set to false.

18.7.1.8 ToolTip

Property: ToolTip as string

Description:
This property gets the text that is shown as a tool-tip for each command. If the command does not have a tooltip text, the property returns an empty string.

18.7.2 MapForceCommands

Properties:
Count
Item

Description:
Collection of Command objects to get access to command labels and IDs of the MapForceControl. Those commands can be executed with the Exec method and their status can be queried with QueryStatus.

18.7.2.1 Count

Property: Count as long

Description:
Number of Command objects on this level of the collection.

18.7.2.2 Item

Property: Item (n as long) as Command

Description:
Gets the command with the index n in this collection. Index is 1-based.

18.7.3 MapForceControl

Properties:
IntegrationLevel
Appearance
Application
BorderStyle
CommandsList
CommandsStructure (deprecated)
EnableUserPrompts
MainMenu
Toolbars

Methods:
Open
Exec
QueryStatus

Events:
OnUpdateCmdUI
OnOpenedOrFocused
OnCloseEditingWindow
OnFileChangedAlert
OnContextChanged
OnDocumentOpened
OnValidationWindowUpdated

This object is a complete ActiveX control and should only be visible if the MapForce library is used in the Application Level mode.

CLSID: A38637E9-5759-4456-A167-F01160CC22C1
ProgID: Altova.MapForceControl

18.7.3.1 Properties

The following properties are defined:
IntegrationLevel
EnableUserPrompts
Appearance
BorderStyle

Command related properties:
CommandsList
MainMenu
Toolbars
CommandsStructure (deprecated)

Access to MapForceAPI:
Application

18.7.3.1.1 Appearance

**Property:** Appearance as short

**Dispatch Id:** -520

**Description:**
A value not equal to 0 displays a client edge around the control. Default value is 0.

18.7.3.1.2 Application

**Property:** Application as Application

**Dispatch Id:** 1

**Description:**
The **Application** property gives access to the **Application** object of the complete MapForce automation server API. The property is read-only.

18.7.3.1.3 BorderStyle

**Property:** BorderStyle as short

**Dispatch Id:** -504

**Description:**
A value of 1 displays the control with a thin border. Default value is 0.

18.7.3.1.4 CommandsList

**Property:** CommandList as Commands (read-only)
1718

ActiveX Integration

Object Reference

Dispatch Id: 1004
Description:
This property returns a flat list of all commands defined available with MapForceControl. To get
commands organized according to their menu structure, use MainMenu. To get toolbar
commands, use Toolbars.

public void GetAllMapForceCommands()
{
// Get all commands from the MapForce ActiveX control assigned to the
current form
MapForceControlLib.MapForceCommands commands =
this.axMapForceControl1.CommandList;
// Iterate through all commands
for (int i = 0; i < commands.Count; i++)
{
// Get each command by index and output it to the console
MapForceControlLib.MapForceCommand cmd =
axMapForceControl1.CommandList[i];
Console.WriteLine("{0} {1} {2}", cmd.ID, cmd.Name,
cmd.Label.Replace("&", ""));
}
}
C# example

18.7.3.1.5

EnableUserPrompts

Property: EnableUserPrompts as boolean
Dispatch Id: 1006
Description:
Setting this property to false, disables user prompts in the control. The default value is true.

18.7.3.1.6

IntegrationLevel

Property: IntegrationLevel as ICActiveXIntegrationLevel
Dispatch Id: 1000
Description:
The IntegrationLevel property determines the operation mode of the control. See also
Integration at Application Level and Integration at Document Level for more information.
Note: It is important to set this property immediately after the creation of the MapForceControl
object.

Altova MapForce 2019 Enterprise Edition

© 2019 Altova Gmb H


18.7.3.1.7  MainMenu

Property: MainMenu as Command (read-only)

Dispatch Id: 1003

Description:
This property provides information about the structure and commands available in the MapForceControl main menu, as a Command object. The Command object contains all available submenus of MapForce (for example "File", "Edit", "View" etc.). To access the submenu objects, use the SubCommands property of the MainMenu property. Each submenu is also a Command object. For each submenu, you can then further iterate through their SubCommands property in order to get their corresponding child commands and separators (this technique may be used, for example, to create the application menu programmatically). Note that some menu commands act as containers ("parents") for other menu commands, in which case they also have a SubCommands property. To get the structure of all menu commands programmatically, you will likely need to create a recursive function.

```csharp
public void GetMapForceMenus()
{
    // Get the main menu from the MapForce ActiveX control assigned to the current form
    MapForceControlLib.MapForceCommand mainMenu = this.axMapForceControl1.MainMenu;

    // Loop through entries of the main menu (e.g. File, Edit, etc.)
    for (int i = 0; i < mainMenu.SubCommands.Count; i++)
    {
        MapForceControlLib.MapForceCommand menu = mainMenu.SubCommands[i];
        Console.WriteLine("{0} menu has {1} children items (including separators)", menu.Label.Replace("&", ""), menu.SubCommands.Count);
    }
}
```

C# example

18.7.3.1.8  Toolbars

Property: Toolbars as Commands (read-only)

Dispatch Id: 1005

Description:
This property provides information about the structure of MapForceControl toolbars, as a Command object. The Command object contains all available toolbars of MapForce. To access the toolbars, use the SubCommands property of the Toolbars property. Each toolbar is also a Command object. For each toolbar, you can then further iterate through their SubCommands property in order to get their commands (this technique may be used, for example, to create the application's toolbars programmatically).
public void GetMapForceToolbars()
{
    // Get the application toolbars from the MapForce ActiveX control assigned
to the current form
    MapForceControlLib.MapForceCommands toolbars =
        this.axMapForceControl1.Toolbars;

    // Iterate through all toolbars
    for (int i = 0; i < toolbars.Count; i++)
    {
        MapForceControlLib.MapForceCommand toolbar = toolbars[i];
        Console.WriteLine();
        Console.WriteLine("The toolbar "{0}" has the following commands:",
            toolbar.Label);
        // Iterate through all commands of this toolbar
        for (int j = 0; j < toolbar.SubCommands.Count; j++)
        {
            MapForceControlLib.MapForceCommand cmd = toolbar.SubCommands[j];
            // Output only command objects that are not separators
            if (!cmd.IsSeparator)
            {
                Console.WriteLine("{0}, {1}, {2}", cmd.ID, cmd.Name,
                    cmd.Label.Replace("&", "}");
            }
        }
    }
}

C# example

18.7.3.2 Methods

The following methods are defined:

Open
Exec
QueryStatus

18.7.3.2.1 Exec

Method: Exec (nCmdID as long) as boolean

Dispatch Id: 6

Description:
This method calls the MapForce command with the ID nCmdID. If the command can be executed,
the method returns true. To get a list of all available commands, use CommandsList. To retrieve
the status of any command, use QueryStatus.
Open

**Method:** Open (strFilePath as string) as boolean

**Dispatch Id:** 5

**Description:**
The result of the method depends on the extension passed in the argument strFilePath. If the file extension is .sps, a new document is opened. If the file extension is .svp, the corresponding project is opened. If a different file extension is passed into the method, the control tries to load the file as a new component into the active document.

Do not use this method to load documents or projects when using the control in document-level integration mode. Instead, use MapForceControlDocument.Open and MapForceControlPlaceHolder.OpenProject.

QueryStatus

**Method:** QueryStatus (nCmdID as long) as long

**Dispatch Id:** 7

**Description:**
QueryStatus returns the enabled/disabled and checked/unchecked status of the command specified by nCmdID. The status is returned as a bit mask.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Supported</td>
<td>Set if the command is supported.</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Enabled</td>
<td>Set if the command is enabled (can be executed).</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Checked</td>
<td>Set if the command is checked.</td>
</tr>
</tbody>
</table>

This means that if QueryStatus returns 0 the command ID is not recognized as a valid MapForce command. If QueryStatus returns a value of 1 or 5, the command is disabled.

Events

The MapForceControl ActiveX control provides the following connection point events:

- OnUpdateCmdUI
- OnOpenedOrFocused
- OnCloseEditingWindow
- OnFileChangedAlert
- OnContextChanged
- OnDocumentOpened
- OnValidationWindowUpdated
18.7.3.3.1 OnCloseEditingWindow

**Event:** OnCloseEditingWindow (i_strFilePath as String) as boolean

**Dispatch Id:** 1002

**Description:**
This event is triggered when MapForce needs to close an already open document. As an answer to this event, clients should close the editor window associated with `i_strFilePath`. Returning `true` from this event indicates that the client has closed the document. Clients can return `false` if no specific handling is required and MapForceControl should try to close the editor and destroy the associated document control.

18.7.3.3.2 OnContextChanged

**Event:** OnContextChanged (i_strContextName as String, i_bActive as bool) as bool

**Dispatch Id:** 1004

**Description:**
This event is not used in MapForce

18.7.3.3.3 OnDocumentOpened

**Event:** OnDocumentOpened (objDocument as Document)

**Dispatch Id:** 1

**Description:**
This event is triggered whenever a document is opened. The argument `objDocument` is a Document object from the MapForce automation interface and can be used to query for more details about the document, or perform additional operations. When integrating on document-level, it is often better to use the event `MapForceControlDocument.OnDocumentOpened` instead.

18.7.3.3.4 OnFileChangedAlert

**Event:** OnFileChangedAlert (i_strFilePath as String) as bool

**Dispatch Id:** 1001

**Description:**
This event is triggered when a file loaded with MapForceControl is changed on the hard disk by another application. Clients should return `true`, if they handled the event, or `false`, if MapForce should handle it in its customary way, i.e. prompting the user for reload.
18.7.3.3.5  OnLicenseProblem

**Event:** OnLicenseProblem(i_strLicenseProblemText as String)

**Dispatch Id:** 1005

**Description:**
This event is triggered when MapForceControl detects that no valid license is available for this control. In case of restricted user licenses this can happen some time after the control has been initialized. Integrators should use this event to disable access to this control's functionality. After returning from this event, the control will block access to its functionality (e.g. show empty windows in its controls and return errors on requests).

18.7.3.3.6  OnOpenedOrFocused

**Event:** OnOpenedOrFocused(i_strFilePath as String, i_bOpenWithThisControl as bool)

**Dispatch Id:** 1000

**Description:**
When integrating at application level, this event informs clients that a document has been opened, or made active by MapForce.
When integrating at document level, this event instructs the client to open the file i_strFilePath in a document window. If the file is already open, the corresponding document window should be made the active window.

If i_bOpenWithThisControl is true, the document must be opened with MapForceControl, since internal access is required. Otherwise, the file can be opened with different editors.

18.7.3.3.7  OnToolWindowUpdated

**Event:** OnToolWindowUpdated(pToolWnd as long)

**Dispatch Id:** 1006

**Description:**
This event is triggered when the tool window is updated.

18.7.3.3.8  OnUpdateCmdUI

**Event:** OnUpdateCmdUI()

**Dispatch Id:** 1003
Description:
Called frequently to give integrators a good opportunity to check status of MapForce commands using `MapForceControl.QueryStatus`. Do not perform long operations in this callback.

18.7.3.9  OnValidationWindowUpdated

Event: OnValidationWindowUpdated()

Dispatch Id: 3

Description:
This event is triggered whenever the validation output window is updated with new information.

18.7.4  MapForceControlDocument

Properties:
Appearance
BorderStyle
Document
IsModified
Path
ReadOnly

Methods:
Exec
New
Open
QueryStatus
Reload
Save
SaveAs

Events:
OnDocumentOpened
OnDocumentClosed
OnModifiedFlagChanged
OnContextChanged
OnFileChangedAlert
OnActivate

If the MapForceControl is integrated in the Document Level mode each document is displayed in an own object of type `MapForceControlDocument`. The `MapForceControlDocument` contains only one document at the time but can be reused to display different files one after another.

This object is a complete ActiveX control.

CLSID: DFBB0871-DAFE-4502-BB66-08CEB7DF5255
ProgID: Altova.MapForceControlDocument
18.7.4.1 Properties

The following properties are defined:

- **ReadOnly**
- **IsModified**
- **Path**
- **Appearance**
- **BorderStyle**

Access to MapForceAPI:

- **Document**

18.7.4.1.1 Appearance

**Property:** Appearance as short

**Dispatch Id:** -520

**Description:**
A value not equal to 0 displays a client edge around the document control. Default value is 0.

18.7.4.1.2 BorderStyle

**Property:** BorderStyle as short

**Dispatch Id:** -504

**Description:**
A value of 1 displays the control with a thin border. Default value is 0.

18.7.4.1.3 Document

**Property:** Document as Document

**Dispatch Id:** 1

**Description:**
The Document property gives access to the Document object of the MapForce automation server API. This interface provides additional functionality which can be used with the document loaded in the control. The property is read-only.
18.7.4.1.4  IsModified

*Property:* IsModified as boolean (read-only)

*Dispatch Id:* 1006

*Description:*  
IsModified is true if the document content has changed since the last open, reload or save operation. It is false, otherwise.

18.7.4.1.5  Path

*Property:* Path as string

*Dispatch Id:* 1005

*Description:*  
Sets or gets the full path name of the document loaded into the control.

18.7.4.1.6  ReadOnly

*Property:* ReadOnly as boolean

*Dispatch Id:* 1007

*Description:*  
Using this property you can turn on and off the read-only mode of the document. If ReadOnly is true it is not possible to do any modifications.

18.7.4.2  Methods

The following methods are defined:

Document handling:
- New
- Open
- Reload
- Save
- SaveAs

Command Handling:
- Exec
- QueryStatus
18.7.4.2.1  Exec

Method: Exec (nCmdID as long) as boolean

Dispatch Id: 8

Description:
Exec calls the MapForce command with the ID nCmdID. If the command can be executed, the method returns true. This method should be called only if there is currently an active document available in the application.

To get commands organized according to their menu structure, use the MainMenu property of MapForceControl. To get toolbar commands, use the Toolbars property of the MapForceControl.

18.7.4.2.2  New

Method: New () as boolean

Dispatch Id: 1000

Description:
This method initializes a new document inside the control.

18.7.4.2.3  Open

Method: Open (strFileName as string) as boolean

Dispatch Id: 1001

Description:
Open loads the file strFileName as the new document into the control.

18.7.4.2.4  QueryStatus

Method: QueryStatus (nCmdID as long) as long

Dispatch Id: 9

Description:
QueryStatus returns the enabled/disabled and checked/unchecked status of the command specified by nCmdID. The status is returned as a bit mask.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Supported</td>
<td>Set if the command is supported.</td>
</tr>
</tbody>
</table>
1728   ActiveX Integration                                      Object Reference

1 2  Enabled  Set if the command is enabled (can be executed).
2 4  Checked Set if the command is checked.

This means that if \texttt{QueryStatus} returns 0 the command ID is not recognized as a valid
MapForce command. If \texttt{QueryStatus} returns a value of 1 or 5 the command is disabled. The
client should call the \texttt{QueryStatus} method of the document control if there is currently an active
document available in the application.

18.7.4.2.5   \textit{Reload}

\textbf{Method:} \texttt{Reload () as boolean}

\textbf{Dispatch Id:} 1002

\textbf{Description:}
\texttt{Reload} updates the document content from the file system.

18.7.4.2.6   \textit{Save}

\textbf{Method:} \texttt{Save () as boolean}

\textbf{Dispatch Id:} 1003

\textbf{Description:}
\texttt{Save} saves the current document at the location \texttt{Path}.

18.7.4.2.7   \textit{SaveAs}

\textbf{Method:} \texttt{SaveAs (strFileName as string) as boolean}

\textbf{Dispatch Id:} 1004

\textbf{Description:}
\texttt{SaveAs} sets \texttt{Path} to \texttt{strFileName} and then saves the document to this location.

18.7.4.3   \textit{Events}

The MapForce\textit{ControlDocument} ActiveX control provides following connection point events:
\texttt{OnDocumentOpened}
\texttt{OnDocumentClosed}
\texttt{OnModifiedFlagChanged}
\texttt{OnContextChanged}
\texttt{OnFileChangedAlert}
\texttt{OnActivate}
18.7.4.3.1  OnActivate

**Event:** OnActivate ()

**Dispatch Id:** 1005

**Description:** This event is triggered when the document control is activated, has the focus, and is ready for user input.

18.7.4.3.2  OnContextChanged

**Event:** OnContextChanged (i_strContextName as String, i_bActive as bool) as bool

**Dispatch Id:** 1004

**Description:** None

18.7.4.3.3  OnDocumentClosed

**Event:** OnDocumentClosed (objDocument as Document)

**Dispatch Id:** 1001

**Description:** This event is triggered whenever the document loaded into this control is closed. The argument objDocument is a Document object from the MapForce automation interface and should be used with care.

18.7.4.3.4  OnDocumentOpened

**Event:** OnDocumentOpened (objDocument as Document)

**Dispatch Id:** 1000

**Description:** This event is triggered whenever a document is opened in this control. The argument objDocument is a Document object from the MapForce automation interface, and can be used to query for more details about the document, or perform additional operations.
18.7.4.3.5  OnDocumentSaveAs

**Event:** OnContextDocumentSaveAs (i_strFileName as String)

**Dispatch Id:** 1007

**Description:**
This event is triggered when this document gets internally saved under a new name.

18.7.4.3.6  OnFileChangedAlert

**Event:** OnFileChangedAlert () as bool

**Dispatch Id:** 1003

**Description:**
This event is triggered when the file loaded into this document control is changed on the hard disk by another application. Clients should return true, if they handled the event, or false, if MapForce should handle it in its customary way, i.e. prompting the user for reload.

18.7.4.3.7  OnModifiedFlagChanged

**Event:** OnModifiedFlagChanged (i_bIsModified as boolean)

**Dispatch Id:** 1002

**Description:**
This event gets triggered whenever the document changes between modified and unmodified state. The parameter i_bIsModified is true if the document contents differs from the original content, and false, otherwise.

18.7.4.3.8  OnSetEditorTitle

**Event:** OnSetEditorTitle ()

**Dispatch Id:** 1006

**Description:**
This event is being raised when the contained document is being internally renamed.

18.7.5  MapForceControlPlaceHolder

**Properties available for all kinds of placeholder windows:**

DoctrineWindowID
Properties for project placeholder window:

Methods for project placeholder window:

The MapForceControlPlaceHolder control is used to show the additional MapForce windows like Overview, Library or Project window. It is used like any other ActiveX control and can be placed anywhere in the client application.

CLSID: FDEC3B04-05F2-427d-988C-F03A85DE53C2
ProgID: Altova.MapForceControlPlaceHolder

18.7.5.1 Properties

The following properties are defined:

PlaceholderWindowID

Access to MapForceAPI:

Project

18.7.5.1.1 Label

Property: Label as String (read-only)

Dispatch Id: 1001

Description:
This property gives access to the title of the placeholder. The property is read-only.

18.7.5.1.2 PlaceholderWindowID

Property: PlaceholderWindowID as MapForceControlPlaceholderWindow

Dispatch Id: 1

Description:
This property specifies which MapForce window should be displayed in the client area of the control. The PlaceholderWindowID can be set at any time to any valid value of the MapForceControlPlaceholderWindow enumeration. The control changes its state immediately and shows the new MapForce window.
18.7.5.1.3  Project

Property: Project as Project (read-only)

Dispatch Id: 2

Description:
The Project property gives access to the Project object of the MapForce automation server API. This interface provides additional functionality which can be used with the project loaded into the control. The property will return a valid project interface only if the placeholder window has PlaceholderWindowID with a value of MapForceXProjectWindow (=3). The property is read-only.

18.7.5.2  Methods

The following method is defined:

OpenProject  
CloseProject

18.7.5.2.1  OpenProject

Method: OpenProject (strFileName as string) as boolean

Dispatch Id: 3

Description:
OpenProject loads the file strFileName as the new project into the control. The method will fail if the placeholder window has a PlaceholderWindowID different to XMLSpyXProjectWindow (=3).

18.7.5.2.2  CloseProject

Method: CloseProject ()

Dispatch Id: 4

Description:
CloseProject closes the project loaded the control. The method will fail if the placeholder window has a PlaceholderWindowID different to MapForceXProjectWindow (=3).
18.7.5.3 Events

The MapForceControlPlaceholder ActiveX control provides following connection point events:

OnModifiedFlagChanged

18.7.5.3.1 OnModifiedFlagChanged

Event: OnModifiedFlagChanged(i_bIsModified as boolean)

Dispatch Id: 1

Description:
This event gets triggered only for placeholder controls with a PlaceholderWindowID of MapForceXProjectWindow (=3). The event is fired whenever the project content changes between modified and unmodified state. The parameter i_bIsModified is true if the project contents differs from the original content, and false, otherwise.

18.7.5.3.2 OnSetLabel

Event: OnSetLabel(i_strNewLabel as string)

Dispatch Id: 1000

Description:
Raised when the title of the placeholder window is changed.

18.7.6 Enumerations

The following enumerations are defined:

ICAAspectXIntegrationLevel
MapForceControlPlaceholderWindow

18.7.6.1 ICAAspectXIntegrationLevel

Possible values for the IntegrationLevel property of the MapForceControl:

ICAAspectXIntegrationOnApplicationLevel = 0
ICAAspectXIntegrationOnDocumentLevel     = 1
18.7.6.2 **MapForceControlPlaceholderWindow**

This enumeration contains the list of the supported additional MapForce windows.

<table>
<thead>
<tr>
<th>Window Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapForceXNoWindow</td>
<td>-1</td>
</tr>
<tr>
<td>MapForceXLibraryWindow</td>
<td>0</td>
</tr>
<tr>
<td>MapForceXOverviewWindow</td>
<td>1</td>
</tr>
<tr>
<td>MapForceXValidationWindow</td>
<td>2</td>
</tr>
<tr>
<td>MapForceXProjectWindow</td>
<td>3</td>
</tr>
<tr>
<td>MapForceXDebuggerValuesWindow</td>
<td>4</td>
</tr>
<tr>
<td>MapForceXDebuggerContextWindow</td>
<td>5</td>
</tr>
<tr>
<td>MapForceXDebuggerPointsWindow</td>
<td>6</td>
</tr>
</tbody>
</table>
Chapter 19

Appendices
19 Appendices

These appendices contain technical information about MapForce and important licensing information. Each appendix contains sub-sections as given below:

**Technical Data**
- OS and memory requirements
- Altova XML Parser
- Altova XSLT and XQuery Engines
- Unicode support
- Internet usage
- License metering

**License Information**
- Electronic software distribution
- Copyrights
- End User License Agreement
19.1 Engine information

This section contains information about implementation-specific features of the Altova XML Validator, Altova XSLT 1.0 Engine, Altova XSLT 2.0 Engine, and Altova XQuery Engine.

19.1.1 XSLT and XQuery Engine Information

The XSLT and XQuery engines of MapForce follow the W3C specifications closely and are therefore stricter than previous Altova engines—such as those in previous versions of XMLSpy. As a result, minor errors that were ignored by previous engines are now flagged as errors by MapForce.

For example:

- It is a type error (err:XPTY0018) if the result of a path operator contains both nodes and non-nodes.
- It is a type error (err:XPTY0019) if E1 in a path expression E1/E2 does not evaluate to a sequence of nodes.

If you encounter this kind of error, modify either the XSLT/XQuery document or the instance document as appropriate.

This section describes implementation-specific features of the engines, organized by specification:

- XSLT 1.0
- XSLT 2.0
- XQuery 1.0

19.1.1.1 XSLT 1.0

The XSLT 1.0 Engine of MapForce conforms to the World Wide Web Consortium’s (W3C’s) XSLT 1.0 Recommendation of 16 November 1999 and XPath 1.0 Recommendation of 16 November 1999. Note the following information about the implementation.

Notes about the implementation

When the method attribute of xsl:output is set to HTML, or if HTML output is selected by default, then special characters in the XML or XSLT file are inserted in the HTML document as HTML character references in the output. For instance, the character \U00A0 (the hexadecimal character reference for a non-breaking space) is inserted in the HTML code either as a character reference (\#160; or \#xA0;) or as an entity reference, &nbsp;.
19.1.1.2 XSLT 2.0

This section:
- Engine conformance
- Backward compatibility
- Namespaces
- Schema awareness
- Implementation-specific behavior

Conformance
The XSLT 2.0 engine of MapForce conforms to the World Wide Web Consortium’s (W3C’s) XSLT 2.0 Recommendation of 23 January 2007 and XPath 2.0 Recommendation of 14 December 2010.

Backwards Compatibility
The XSLT 2.0 engine is backwards compatible. The only time the backwards compatibility of the XSLT 2.0 engine comes into effect is when using the XSLT 2.0 engine to process an XSLT 1.0 stylesheet. Note that there could be differences in the outputs produced by the XSLT 1.0 Engine and the backwards-compatible XSLT 2.0 engine.

Namespaces
Your XSLT 2.0 stylesheet should declare the following namespaces in order for you to be able to use the type constructors and functions available in XSLT 2.0. The prefixes given below are conventionally used; you could use alternative prefixes if you wish.

<table>
<thead>
<tr>
<th>Namespace Name</th>
<th>Prefix</th>
<th>Namespace URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Schema types</td>
<td>xs:</td>
<td><a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a></td>
</tr>
<tr>
<td>XPath 2.0 functions</td>
<td>fn:</td>
<td><a href="http://www.w3.org/2005/xpath-functions">http://www.w3.org/2005/xpath-functions</a></td>
</tr>
</tbody>
</table>

Typically, these namespaces will be declared on the `<xsl:stylesheet>` or `<xsl:transform>` element, as shown in the following listing:

```xml
true
<xsl:stylesheet version="2.0"
xmlns:xsl=http://www.w3.org/1999/XSL/Transform
xmlns:xs=http://www.w3.org/2001/XMLSchema
xmlns:fn=http://www.w3.org/2005/xpath-functions
...
</xsl:stylesheet>
```

The following points should be noted:
- The XSLT 2.0 engine uses the XPath 2.0 and XQuery 1.0 Functions namespace (listed in the table above) as its default functions namespace. So you can use XPath 2.0 and XSLT 2.0 functions in your stylesheet without any prefix. If you declare the XPath 2.0 Functions namespace in your stylesheet with a prefix, then you can additionally use the prefix assigned in the declaration.
- When using type constructors and types from the XML Schema namespace, the prefix
used in the namespace declaration must be used when calling the type constructor (for example, xs:date).

- Some XPath 2.0 functions have the same name as XML Schema datatypes. For example, for the XPath functions fn:string and fn:boolean there exist XML Schema datatypes with the same local names: xs:string and xs:boolean. So if you were to use the XPath expression string('Hello'), the expression evaluates as fn:string('Hello')—not as xs:string('Hello').

### Schema-awareness

The XSLT 2.0 engine is schema-aware. So you can use user-defined schema types and the xsl:validate instruction.

### Implementation-specific behavior

Given below is a description of how the XSLT 2.0 engine handles implementation-specific aspects of the behavior of certain XSLT 2.0 functions.

**xsl:result-document**

Additionally supported encodings are (the Altova-specific): x-base16tobinary and x-base64tobinary.

**function-available**

The function tests for the availability of in-scope functions (XSLT, XPath, and extension functions).

**unparsed-text**

The href attribute accepts (i) relative paths for files in the base-uri folder, and (ii) absolute paths with or without the file:// protocol. Additionally supported encodings are (the Altova-specific): x-binarytobase16 and x-binarytobase64.

**unparsed-text-available**

The href attribute accepts (i) relative paths for files in the base-uri folder, and (ii) absolute paths with or without the file:// protocol. Additionally supported encodings are (the Altova-specific): x-binarytobase16 and x-binarytobase64.

**Note:** The following encoding values, which were implemented in earlier versions of RaptorXML's predecessor product, AltovaXML, are now deprecated: base16tobinary, base64tobinary, binarytobase16 and binarytobase64.

### 19.1.1.3 XQuery 1.0

**This section:**

- Engine conformance
- Schema awareness
- Encoding
- Namespaces
- XML source and validation
- Static and dynamic type checking
- Library modules
- External functions
Collations
Precision of numeric data
XQuery instructions support

Conformance
The XQuery 1.0 Engine of MapForce conforms to the World Wide Web Consortium's (W3C's) XQuery 1.0 Recommendation of 14 December 2010. The XQuery standard gives implementations discretion about how to implement many features. Given below is a list explaining how the XQuery 1.0 Engine implements these features.

Schema awareness
The XQuery 1.0 Engine is schema-aware.

Encoding
The UTF-8 and UTF-16 character encodings are supported.

Namespaces
The following namespace URIs and their associated bindings are pre-defined.

<table>
<thead>
<tr>
<th>Namespace Name</th>
<th>Prefix</th>
<th>Namespace URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Schema types</td>
<td>xs:</td>
<td><a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a></td>
</tr>
<tr>
<td>Schema instance</td>
<td>xsi:</td>
<td><a href="http://www.w3.org/2001/XMLSchema-instance">http://www.w3.org/2001/XMLSchema-instance</a></td>
</tr>
<tr>
<td>Built-in functions</td>
<td>fn:</td>
<td><a href="http://www.w3.org/2005/xpath-functions">http://www.w3.org/2005/xpath-functions</a></td>
</tr>
<tr>
<td>Local functions</td>
<td>local:</td>
<td><a href="http://www.w3.org/2005/xquery-local-functions">http://www.w3.org/2005/xquery-local-functions</a></td>
</tr>
</tbody>
</table>

The following points should be noted:

- The XQuery 1.0 Engine recognizes the prefixes listed above as being bound to the corresponding namespaces.
- Since the built-in functions namespace listed above is the default functions namespace in XQuery, the fn: prefix does not need to be used when built-in functions are invoked (for example, string("Hello") will call the fn:string function). However, the prefix fn: can be used to call a built-in function without having to declare the namespace in the query prolog (for example: fn:string("Hello")).
- You can change the default functions namespace by declaring the default function namespace expression in the query prolog.
- When using types from the XML Schema namespace, the prefix xs: may be used without having to explicitly declare the namespaces and bind these prefixes to them in the query prolog. (Example: xs:date and xs:yearMonthDuration.) If you wish to use some other prefix for the XML Schema namespace, this must be explicitly declared in the query prolog. (Example: declare namespace alt = "http://www.w3.org/2001/XMLSchema"; alt:date("2004-10-04").)
- Note that the untypedAtomic, dayTimeDuration, and yearMonthDuration datatypes have been moved, with the CRs of 23 January 2007, from the XPath Datatypes namespace to the XML Schema namespace, so: xs:yearMonthDuration.
If namespaces for functions, type constructors, node tests, etc are wrongly assigned, an error is reported. Note, however, that some functions have the same name as schema datatypes, e.g. fn:string and fn:boolean. (Both xs:string and xs:boolean are defined.) The namespace prefix determines whether the function or type constructor is used.

**XML source document and validation**

XML documents used in executing an XQuery document with the XQuery 1.0 Engine must be well-formed. However, they do not need to be valid according to an XML Schema. If the file is not valid, the invalid file is loaded without schema information. If the XML file is associated with an external schema and is valid according to it, then post-schema validation information is generated for the XML data and will be used for query evaluation.

**Static and dynamic type checking**

The static analysis phase checks aspects of the query such as syntax, whether external references (e.g. for modules) exist, whether invoked functions and variables are defined, and so on. If an error is detected in the static analysis phase, it is reported and the execution is stopped.

Dynamic type checking is carried out at run-time, when the query is actually executed. If a type is incompatible with the requirement of an operation, an error is reported. For example, the expression xs:string("1") + 1 returns an error because the addition operation cannot be carried out on an operand of type xs:string.

**Library Modules**

Library modules store functions and variables so they can be reused. The XQuery 1.0 Engine supports modules that are stored in a single external XQuery file. Such a module file must contain a module declaration in its prolog, which associates a target namespace. Here is an example module:

```xquery
module namespace libns="urn:module-library";
declare variable $libns:company := "Altova";
declare function libns:webaddress() { "http://www.altova.com" };
```

All functions and variables declared in the module belong to the namespace associated with the module. The module is used by importing it into an XQuery file with the import module statement in the query prolog. The import module statement only imports functions and variables declared directly in the library module file. As follows:

```xquery
import module namespace modlib = "urn:module-library" at "modulefilename.xq";

if ($modlib:company = "Altova")
then modlib:webaddress()
else error("No match found.")
```

**External functions**

External functions are not supported, i.e. in those expressions using the external keyword, as in:

```xquery
declare function hoo($param as xs:integer) as xs:string external;
```
Collations
The default collation is the Unicode-codepoint collation, which compares strings on the basis of their Unicode codepoint. Other supported collations are the ICU collations listed [here](#). To use a specific collation, supply its URI as given in the list of supported collations. Any string comparisons, including for the `fn:max` and `fn:min` functions, will be made according to the specified collation. If the collation option is not specified, the default Unicode-codepoint collation is used.

Precision of numeric types
- The `xs:integer` datatype is arbitrary-precision, i.e. it can represent any number of digits.
- The `xs:decimal` datatype has a limit of 20 digits after the decimal point.
- The `xs:float` and `xs:double` datatypes have limited-precision of 15 digits.

XQuery Instructions Support
The `Pragma` instruction is not supported. If encountered, it is ignored and the fallback expression is evaluated.

19.1.2 XSLT and XPath/XQuery Functions
This section lists Altova extension functions and other extension functions that can be used in XPath and/or XQuery expressions. Altova extension functions can be used with Altova's XSLT and XQuery engines, and provide functionality additional to that available in the function libraries defined in the W3C standards.

General points
The following general points should be noted:
- Functions from the core function libraries defined in the W3C specifications can be called without a prefix. That's because the XSLT and XQuery engines read non-prefixed functions as belonging to a default functions namespace which is that specified in the XPath/XQuery functions specifications [http://www.w3.org/2005/xpath-functions](http://www.w3.org/2005/xpath-functions). If this namespace is explicitly declared in an XSLT or XQuery document, the prefix used in the namespace declaration can also optionally be used on function names.
- In general, if a function expects a sequence of one item as an argument, and a sequence of more than one item is submitted, then an error is returned.
- All string comparisons are done using the Unicode codepoint collation.
- Results that are QNames are serialized in the form `[prefix:]localname`.

**Precision of `xs:decimal`**
The precision refers to the number of digits in the number, and a minimum of 18 digits is required by the specification. For division operations that produce a result of type `xs:decimal`, the precision is 19 digits after the decimal point with no rounding.
**Implicit timezone**

When two `date`, `time`, or `dateTime` values need to be compared, the timezone of the values being compared need to be known. When the timezone is not explicitly given in such a value, the implicit timezone is used. The implicit timezone is taken from the system clock, and its value can be checked with the `implicit-timezone()` function.

**Collations**

The default collation is the Unicode codepoint collation, which compares strings on the basis of their Unicode codepoint. The engine uses the Unicode Collation Algorithm. Other supported collations are the [ICU collations](https://icu-project.org/collation/) listed below; to use one of these, supply its URI as given in the table below. Any string comparisons, including for the `max` and `min` functions, will be made according to the specified collation. If the collation option is not specified, the default Unicode-codepoint collation is used.

<table>
<thead>
<tr>
<th>Language</th>
<th>URIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>da: Danish</td>
<td>da_DK</td>
</tr>
<tr>
<td>de: German</td>
<td>de_AT, de_BE, de_CH, de_DE, de_LI, de_LU</td>
</tr>
<tr>
<td>es: Spanish</td>
<td>es_419, es_AR, es_BO, es_CL, es_CO, es_CR, es_DO, es_EC, es_ES, es_GQ, es_GT, es_HN, es_MX, es_NI, es_PA, es_PE, es_PR, es_PY, es_SV, es_US, es_UY, es_VE</td>
</tr>
<tr>
<td>it: Italian</td>
<td>it_CH, it_IT</td>
</tr>
<tr>
<td>ja: Japanese</td>
<td>ja_JP</td>
</tr>
<tr>
<td>nb: Norwegian</td>
<td>nb_NO</td>
</tr>
<tr>
<td>nn: Nynorsk</td>
<td>nn_NO</td>
</tr>
<tr>
<td>nl: Dutch</td>
<td>nl_AW, nl_BE, nl_NL</td>
</tr>
<tr>
<td>ru: Russian</td>
<td>ru_MD, ru_RU, ru_UA</td>
</tr>
<tr>
<td>sv: Swedish</td>
<td>sv_FI, sv_SE</td>
</tr>
</tbody>
</table>

**Namespace axis**

The namespace axis is deprecated in XPath 2.0. Use of the namespace axis is, however, supported. To access namespace information with XPath 2.0 mechanisms, use the `in-scope-prefixes()`, `namespace-uri()` and `namespace-uri-for-prefix()` functions.
19.1.2.1 Altova Extension Functions

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the Altova extension functions namespace, http://www.altova.com/xslt-extensions, and are indicated in this section with the prefix altova:, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

Functions defined in the W3C's XPath/XQuery Functions specifications can be used in: (i) XPath expressions in an XSLT context, and (ii) in XQuery expressions in an XQuery document. In this documentation we indicate the functions that can be used in the former context (XPath in XSLT) with an XP symbol and call them XPath functions; those functions that can be used in the latter (XQuery) context are indicated with an xq symbol; they work as XQuery functions. The W3C's XSLT specifications—not XPath/XQuery Functions specifications—also define functions that can be used in XPath expressions in XSLT documents. These functions are marked with an XSLT symbol and are called XSLT functions. The XPath/XQuery and XSLT versions in which a function can be used are indicated in the description of the function (see symbols below). Functions from the XPath/XQuery and XSLT function libraries are listed without a prefix. Extension functions from other libraries, such as Altova extension functions, are listed with a prefix.

<table>
<thead>
<tr>
<th>XPath functions (used in XPath expressions in XSLT):</th>
<th>XP1 XP2 XP3.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLT functions (used in XPath expressions in XSLT):</td>
<td>XSLT1 XSLT2 XSLT3</td>
</tr>
<tr>
<td>XQuery functions (used in XQuery expressions in XQuery):</td>
<td>XQ1 XQ3.1</td>
</tr>
</tbody>
</table>

**XSLT functions**

XSLT functions can only be used in XPath expressions in an XSLT context (similarly to XSLT 2.0's current-group() or key() functions). These functions are not intended for, and will not work in, a non-XSLT context (for instance, in an XQuery context). Note that XSLT functions for XBRL can be used only with editions of Altova products that have XBRL support.

**XPath/XQuery functions**

XPath/XQuery functions can be used both in XPath expressions in XSLT contexts as well as in XQuery expressions:

- Date/Time
- Geolocation
- Image-related
- Numeric
- Sequence
- String
- Miscellaneous
**Chart functions (Enterprise and Server Editions only)**

Altova extension functions for charts are supported only in the Enterprise and Server Editions of Altova products and enable charts to be generated from XML data.

### 19.1.2.1.1 XSLT Functions

**XSLT extension functions** can be used in XPath expressions in an XSLT context. They will not work in a non-XSLT context (for instance, in an XQuery context).

**Note about naming of functions and language applicability**

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace**, http://www.altova.com/xslt-extensions, and are indicated in this section with the prefix `altova:`, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

| *XPath functions (used in XPath expressions in XSLT):* | XP1 XP2 XP3.1 |
| *XSLT functions (used in XPath expressions in XSLT):* | XSLT1 XSLT2 XSLT3 |
| *XQuery functions (used in XQuery expressions in XQuery):* | XQ1 XQ3.1 |

**Standard functions**

- **distinct-nodes** [altova:]

  ```
  altova:distinct-nodes(node()*) as node()*
  ```

  Takes a set of one or more nodes as its input and returns the same set minus nodes with duplicate values. The comparison is done using the XPath/XQuery function `fn:deep-equal`.

  **Examples**

  - `altova:distinct-nodes(country)` returns all child `country` nodes less those having duplicate values.

- **evaluate** [altova:]

  ```
  altova:evaluate(XPathExpression as xs:string[, ValueOf$p1, ... ValueOf$pN])
  ```

  Takes an XPath expression, passed as a string, as its mandatory argument. It returns the output of the evaluated expression. For example: `altova:evaluate('//Name[1]')` returns the contents of the first `Name` element in the document. Note that the expression `//Name[1]` is passed as a string by enclosing it in single quotes.

  The `altova:evaluate` function can optionally take additional arguments. These arguments are the values of in-scope variables that have the names `p1`, `p2`, `p3`... `pN`. Note the following points about usage: (i) The variables must be defined with names of the form `pX`, where `x` is...
an integer; (ii) the `altova:evaluate` function's arguments (see signature above), from the second argument onwards, provide the values of the variables, with the sequence of the arguments corresponding to the numerically ordered sequence of variables: p1 to pN. The second argument will be the value of the variable p1, the third argument that of the variable p2, and so on; (iii) The variable values must be of type `item*`.

Example

```xml
<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
<xsl:value-of select="altova:evaluate($xpath, 10, 20, 'hi')" />
outputs "hi 20 10"
```

In the listing above, notice the following:

- The second argument of the `altova:evaluate` expression is the value assigned to the variable $p1, the third argument that assigned to the variable $p2, and so on.
- Notice that the fourth argument of the function is a string value, indicated by its being enclosed in quotes.
- The `select` attribute of the `xs:variable` element supplies the XPath expression. Since this expression must be of type `xs:string`, it is enclosed in single quotes.

Examples to further illustrate the use of variables

- `<xsl:variable name="xpath" select="'p1'" />
  <xsl:value-of select="altova:evaluate($xpath, //Name[1])" />
Outputs value of the first Name element.

- `<xsl:variable name="xpath" select="'p1'" />
  <xsl:value-of select="altova:evaluate($xpath, '//Name[1]' )" />
Outputs 
```

The `altova:evaluate()` extension function is useful in situations where an XPath expression in the XSLT stylesheet contains one or more parts that must be evaluated dynamically. For example, consider a situation in which a user enters his request for the sorting criterion and this criterion is stored in the attribute `UserReq/@sortkey`. In the stylesheet, you could then have the expression: `<xsl:sort select="altova:evaluate(../UserReq/@sortkey)" order="ascending")/>. The `altova:evaluate()` function reads the `sortkey` attribute of the `UserReq` child element of the parent of the context node. Say the value of the `sortkey` attribute is `Price`, then `Price` is returned by the `altova:evaluate()` function and becomes the value of the `select` attribute: `<xsl:sort select="Price" order="ascending")/>. If this sort instruction occurs within the context of an element called `Order`, then the `Order` elements will be sorted according to the values of their `Price` children. Alternatively, if the value of `@sortkey` were, say, `Date`, then the `Order` elements would be sorted according to the values of their `Date` children. So the sort criterion for `Order` is selected from the `sortkey` attribute at runtime. This could not have been achieved with an expression like: `<xsl:sort select="/UserReq/@sortkey" order="ascending")/>. In the case shown above, the sort criterion would be the `sortkey` attribute itself, not `Price` or `Date` (or any other current content of `sortkey`).

**Note:** The static context includes namespaces, types, and functions—but not variables—
from the calling environment. The base URI and default namespace are inherited.

More examples

- Static variables: `<xsl:value-of select="$i3, $i2, $i1" />`
  Outputs the values of three variables.

- Dynamic XPath expression with dynamic variables:
  `<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />`
  `<xsl:value-of select="altova:evaluate($xpath, 10, 20, 30)" />`
  Outputs "30 20 10"

- Dynamic XPath expression with no dynamic variable:
  `<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />`
  `<xsl:value-of select="altova:evaluate($xpath)" />`
  Outputs error: No variable defined for $p3.

encode-for-rtf [altova:]

```
altova:encode-for-rtf(input as xs:string, preserveallwhitespace as xs:boolean, preservenewlines as xs:boolean) as xs:string
```

Converts the input string into code for RTF. Whitespace and new lines will be preserved according to the boolean value specified for their respective arguments.

[ Top ]

XBRL functions

Altova XBRL functions can be used only with editions of Altova products that have XBRL support.

- xbrl-footnotes [altova:]
  
  ```
  altova:xbrl-footnotes(node()) as node() *
  ```
  Takes a node as its input argument and returns the set of XBRL footnote nodes referenced by the input node.

- xbrl-labels [altova:]
  
  ```
  altova:xbrl-labels(xs:QName, xs:string) as node() *
  ```
  Takes two input arguments: a node name and the taxonomy file location containing the node. The function returns the XBRL label nodes associated with the input node.

[ Top ]
19.1.2.1.2 **XPath/XQuery Functions: Date and Time**

Altova's date/time extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data held as XML Schema's various date and time datatypes. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace**, http://www.altova.com/xslt-extensions, and are indicated in this section with the prefix **altova:​**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

<table>
<thead>
<tr>
<th>XPath functions (used in XPath expressions in XSLT):</th>
<th>XP1 XP2 XP3.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLT functions (used in XPath expressions in XSLT):</td>
<td>XSLT1 XSLT2 XSLT3</td>
</tr>
<tr>
<td>XQuery functions (used in XQuery expressions in XQuery):</td>
<td>XQ1 XQ3.1</td>
</tr>
</tbody>
</table>

**Grouped by functionality**

- Add a duration to xs:dateTime and return xs:dateTime
- Add a duration to xs:date and return xs:date
- Add a duration to xs:time and return xs:time
- Format and retrieve durations
- Remove timezone from functions that generate current date/time
- Return days, hours, minutes, and seconds from durations
- Return weekday as integer from date
- Return week number as integer from date
- Build date, time, or duration type from lexical components of each type
- Construct date, dateTime, or time type from string input
- Age-related functions

**Grouped alphabetically**

altova:add-days-to-date
altova:add-days-to-dateTime
altova:add-hours-to-dateTime
altova:add-hours-to-time
altova:add-minutes-to-dateTime
altova:add-minutes-to-time
altova:add-months-to-date
altova:add-months-to-dateTime
altova:add-seconds-to-dateTime
altova:add-seconds-to-time
altova:add-years-to-date
altova:add-years-to-dateTime
altova:age
altova:age-details
Add a duration to xs:dateTime

These functions add a duration to xs:dateTime and return xs:dateTime. The xs:dateTime type has a format of CCYY-MM-DDThh:mm:ss.sss. This is a concatenation of the xs:date and xs:time formats separated by the letter T. A timezone suffix +01:00 (for example) is optional.

### add-years-to-dateTime [altova:]

```
altova:add-years-to-dateTime(DateTime as xs:dateTime, Years as xs:integer) as xs:dateTime
```

Adds a duration in years to an xs:dateTime (see examples below). The second argument is the number of years to be added to the xs:dateTime supplied as the first argument. The result is of type xs:dateTime.

**Examples**

- `altova:add-years-to-dateTime(xs:dateTime("2014-01-15T14:00:00"), 10)` returns `2024-01-15T14:00:00`
- `altova:add-years-to-dateTime(xs:dateTime("2014-01-15T14:00:00"), -4)` returns `2010-01-15T14:00:00`

### add-months-to-dateTime [altova:]

```
altova:add-months-to-dateTime(DateTime as xs:dateTime, Months as xs:integer) as xs:dateTime
```

Adds a duration in months to an xs:dateTime (see examples below). The second argument is the number of months to be added to the xs:dateTime supplied as the first argument. The result is of type xs:dateTime.
Examples

- `altova:add-months-to-dateTime(xs:dateTime("2014-01-15T14:00:00"), 10)` returns `2014-11-15T14:00:00`
- `altova:add-months-to-dateTime(xs:dateTime("2014-01-15T14:00:00"), -2)` returns `2013-11-15T14:00:00`

add-days-to-dateTime [altova:]

\[ altova:add-days-to-dateTime(DateTime as xs:dateTime, Days as xs:integer) as xs:dateTime \]

Adds a duration in days to an `xs:dateTime` (see examples below). The second argument is the number of days to be added to the `xs:dateTime` supplied as the first argument. The result is of type `xs:dateTime`.

Examples

- `altova:add-days-to-dateTime(xs:dateTime("2014-01-15T14:00:00"), 10)` returns `2014-01-25T14:00:00`
- `altova:add-days-to-dateTime(xs:dateTime("2014-01-15T14:00:00"), -8)` returns `2014-01-07T14:00:00`

add-hours-to-dateTime [altova:]

\[ altova:add-hours-to-dateTime(DateTime as xs:dateTime, Hours as xs:integer) as xs:dateTime \]

Adds a duration in hours to an `xs:dateTime` (see examples below). The second argument is the number of hours to be added to the `xs:dateTime` supplied as the first argument. The result is of type `xs:dateTime`.

Examples

- `altova:add-hours-to-dateTime(xs:dateTime("2014-01-15T13:00:00"), 10)` returns `2014-01-15T23:00:00`
- `altova:add-hours-to-dateTime(xs:dateTime("2014-01-15T13:00:00"), -8)` returns `2014-01-15T05:00:00`

add-minutes-to-dateTime [altova:]

\[ altova:add-minutes-to-dateTime(DateTime as xs:dateTime, Minutes as xs:integer) as xs:dateTime \]

Adds a duration in minutes to an `xs:dateTime` (see examples below). The second argument is the number of minutes to be added to the `xs:dateTime` supplied as the first argument. The result is of type `xs:dateTime`.

Examples

- `altova:add-minutes-to-dateTime(xs:dateTime("2014-01-15T14:10:00"), -5)` returns `2014-01-15T14:05:00`

add-seconds-to-dateTime [altova:]

\[ altova:add-seconds-to-dateTime(DateTime as xs:dateTime, Seconds as xs:integer) as xs:dateTime \]

Adds a duration in seconds to an `xs:dateTime` (see examples below). The second argument is the number of seconds to be added to the `xs:dateTime` supplied as the first argument. The result is of type `xs:dateTime`.

Examples

altova:add-seconds-to-dateTime(DateTime as xs:dateTime, Seconds as xs:integer) as xs:dateTime

Adds a duration in seconds to an xs:dateTime (see examples below). The second argument is the number of seconds to be added to the xs:dateTime supplied as the first argument. The result is of type xs:dateTime.

Examples

- altova:add-seconds-to-dateTime(xs:dateTime("2014-01-15T14:00:10"), 20) returns 2014-01-15T14:00:30
- altova:add-seconds-to-dateTime(xs:dateTime("2014-01-15T14:00:10"), -5) returns 2014-01-15T14:00:05

Add a duration to xs:date

These functions add a duration to xs:dateTime and return xs:date. The xs:dateTime type has a format of CCYY-MM-DD.

^ add-years-to-date [altova:]

altova:add-years-to-date(Date as xs:date, Years as xs:integer) as xs:date

Adds a duration in years to a date. The second argument is the number of years to be added to the xs:date supplied as the first argument. The result is of type xs:date.

Examples

- altova:add-years-to-date(xs:date("2014-01-15"), 10) returns 2024-01-15
- altova:add-years-to-date(xs:date("2014-01-15"), -4) returns 2010-01-15

^ add-months-to-date [altova:]

altova:add-months-to-date(Date as xs:date, Months as xs:integer) as xs:date

Adds a duration in months to a date. The second argument is the number of months to be added to the xs:date supplied as the first argument. The result is of type xs:date.

Examples


^ add-days-to-date [altova:]

altova:add-days-to-date(Date as xs:date, Days as xs:integer) as xs:date

Adds a duration in days to a date. The second argument is the number of days to be added to the xs:date supplied as the first argument. The result is of type xs:date.

Examples

Format and retrieve durations  

These functions add a duration to \texttt{xs:date} and return \texttt{xs:date}. The \texttt{xs:date} type has a format of CCYY-MM-DD.

\begin{itemize}
  \item \texttt{altova:format-duration} (\texttt{Duration} as \texttt{xs:duration}, \texttt{Picture} as \texttt{xs:string}) as \texttt{xs:string}
  \begin{itemize}
    \item \texttt{altova:format-duration} (xs:duration("P2DT53M11.7S"), "Days:[D01] Hours:[H01] Minutes:[m01] Seconds:[s01] Fractions:[f0]") returns "Days:02 Hours:02 Minutes:53 Seconds:01 Fractions:7"
    \item \texttt{altova:format-duration} (xs:duration("P3M11.7S"), "Months:[M01] Days:[D01] Hours:[H01] Minutes:[m01]") returns "Months:03 Days:01 Hours:02 Minutes:53"
  \end{itemize}
\end{itemize}

\begin{itemize}
  \item \texttt{altova:parse-duration} (\texttt{InputString} as \texttt{xs:string}, \texttt{Picture} as \texttt{xs:string}) as \texttt{xs:duration}
  \begin{itemize}
    \item \texttt{altova:parse-duration} ("Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7", "Days:[D01] Hours:[H01] Minutes:[m01] Seconds:[s01] Fractions:[f0]") returns "P2DT53M11.7S"
    \item \texttt{altova:parse-duration} ("Months:03 Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7", "Months:[M01] Days:[D01] Hours:[H01] Minutes:[m01]"") returns "P3M2DT2H53M"
  \end{itemize}
\end{itemize}

Add a duration to \texttt{xs:time}  

These functions add a duration to \texttt{xs:time} and return \texttt{xs:time}. The \texttt{xs:time} type has a lexical form of \texttt{hh:mm:ss.sss}. An optional time zone may be suffixed. The letter \texttt{Z} indicates Coordinated Universal Time (UTC). All other time zones are represented by their difference from UTC in the format \texttt{+hh:mm}, or \texttt{-hh:mm}. If no time zone value is present, it is considered unknown; it is not assumed to be UTC.

\begin{itemize}
  \item \texttt{altova:add-hours-to-time} (\texttt{InputTime} as \texttt{xs:time}, \texttt{Duration} as \texttt{xs:duration}) as \texttt{xs:time}
  \begin{itemize}
    \item \texttt{altova:add-hours-to-time} (xs:time("2011-12-31T00:30:00.000-08:00"), xs:duration("P2DT2H53M11.7S")) returns "2011-12-31T02:53:00.000" Z
  \end{itemize}
\end{itemize}
altova:add-hours-to-time(Time as xs:time, Hours as xs:integer) as xs:time
XP3.1 XQ3.1
Adds a duration in hours to a time. The second argument is the number of hours to be added to the xs:time supplied as the first argument. The result is of type xs:time.

Examples
- altova:add-hours-to-time(xs:time("11:00:00"), 10) returns 21:00:00
- altova:add-hours-to-time(xs:time("11:00:00"), -7) returns 04:00:00

add-minutes-to-time [altova:]

altova:add-minutes-to-time(Time as xs:time, Minutes as xs:integer) as xs:time
XP3.1 XQ3.1
Adds a duration in minutes to a time. The second argument is the number of minutes to be added to the xs:time supplied as the first argument. The result is of type xs:time.

Examples
- altova:add-minutes-to-time(xs:time("14:10:00"), 45) returns 14:55:00
- altova:add-minutes-to-time(xs:time("14:10:00"), -5) returns 14:05:00

add-seconds-to-time [altova:]

altova:add-seconds-to-time(Time as xs:time, Seconds as xs:integer) as xs:time
XP3.1 XQ3.1
Adds a duration in seconds to a time. The second argument is the number of seconds to be added to the xs:time supplied as the first argument. The result is of type xs:time. The Seconds component can be in the range of 0 to 59.999.

Examples
- altova:add-seconds-to-time(xs:time("14:00:00"), 20) returns 14:00:20
- altova:add-seconds-to-time(xs:time("14:00:00"), 20.895) returns 14:00:20.895

Remove the timezone part from date/time datatypes

These functions remove the timezone from the current xs:dateTime, xs:date, or xs:time values, respectively. Note that the difference between xs:dateTime and xs:dateTimeStamp is that in the case of the latter the timezone part is required (while it is optional in the case of the former). So the format of an xs:dateTimeStamp value is: CCYY-MM-DDThh:mm:ss.sss±hh:mm or CCYY-MM-DDThh:mm:ss.sssZ. If the date and time is read from the system clock as xs:dateTimeStamp, the current-dateTime-no-TZ() function can be used to remove the timezone if so required.

current-date-no-TZ [altova:]

altova:current-date-no-TZ() as xs:date XP3.1 XQ3.1
This function takes no argument. It removes the timezone part of current-date() (which is the current date according to the system clock) and returns an xs:date value.

Examples
If the current date is `2014-01-15+01:00`:

- `altova:current-date-no-TZ()` returns `2014-01-15`

```
altova:current-dateTime-no-TZ() as xs:dateTime  XP3.1 XQ3.1
```

This function takes no argument. It removes the timezone part of `current-dateTime()` (which is the current date-and-time according to the system clock) and returns an `xs:dateTime` value.

**Examples**

If the current dateTime is `2014-01-15T14:00:00+01:00`:

- `altova:current-dateTime-no-TZ()` returns `2014-01-15T14:00:00`

```
altova:current-time-no-TZ() as xs:time  XP3.1 XQ3.1
```

This function takes no argument. It removes the timezone part of `current-time()` (which is the current time according to the system clock) and returns an `xs:time` value.

**Examples**

If the current time is `14:00:00+01:00`:

- `altova:current-time-no-TZ()` returns `14:00:00`

```
altova:date-no-TZ(InputDate as xs:date) as xs:date  XP3.1 XQ3.1
```

This function takes an `xs:date` argument, removes the timezone part from it, and returns an `xs:date` value. Note that the date is not modified.

**Examples**

- `altova:date-no-TZ(xs:date("2014-01-15+01:00"))` returns `2014-01-15`

```
altova:dateTime-no-TZ(InputDateTime as xs:dateTime) as xs:dateTime  XP3.1 XQ3.1
```

This function takes an `xs:dateTime` argument, removes the timezone part from it, and returns an `xs:dateTime` value. Note that neither the date nor the time is modified.

**Examples**

- `altova:dateTime-no-TZ(xs:dateTime("2014-01-15T14:00:00+01:00"))` returns `2014-01-15T14:00:00`

```
altova:time-no-TZ(InputTime as xs:time) as xs:time  XP3.1 XQ3.1
```

**Examples**

- `altova:time-no-TZ(xs:time("2014-01-15T14:00:00+01:00"))` returns `2014-01-15T14:00:00`
This function takes an `xs:time` argument, removes the timezone part from it, and returns an `xs:time` value. Note that the time is not modified.

**Examples**

- `altova:time-no-TZ(xs:time("14:00:00+01:00"))` returns `14:00:00`

---

Return the number of days, hours, minutes, seconds from durations

These functions return the number of days in a month, and the number of hours, minutes, and seconds, respectively, from durations.

### days-in-month [altova:]

```xml
altova:days-in-month(Year as xs:integer, Month as xs:integer) as xs:integer
```

Returns the number of days in the specified month. The month is specified by means of the `Year` and `Month` arguments.

**Examples**

- `altova:days-in-month(2018, 10)` returns `31`
- `altova:days-in-month(2018, 2)` returns `28`
- `altova:days-in-month(2020, 2)` returns `29`

### hours-from-dayTimeDuration-accumulated

```xml
altova:hours-from-dayTimeDuration-accumulated(DayAndTime as xs:duration) as xs:integer
```

Returns the total number of hours in the duration submitted by the `DayAndTime` argument (which is of type `xs:duration`). The hours in the `Day` and `Time` components are added together to give a result that is an integer. A new hour is counted for a full 60 minutes. Negative durations result in a negative hour value.

**Examples**

- `altova:hours-from-dayTimeDuration-accumulated(xs:duration("P5D"))` returns `120`, which is the total number of hours in 5 days.
- `altova:hours-from-dayTimeDuration-accumulated(xs:duration("P5DT2H"))` returns `122`, which is the total number of hours in 5 days plus 2 hours.
- `altova:hours-from-dayTimeDuration-accumulated(xs:duration("P5DT2H60M"))` returns `123`, which is the total number of hours in 5 days plus 2 hours and 60 mins.
- `altova:hours-from-dayTimeDuration-accumulated(xs:duration("P5DT2H119M"))` returns `123`, which is the total number of hours in 5 days plus 2 hours and 119 mins.
- `altova:hours-from-dayTimeDuration-accumulated(xs:duration("P5DT2H120M"))` returns `124`, which is the total number of hours in 5 days plus 2 hours and 120 mins.
- `altova:hours-from-dayTimeDuration-accumulated(xs:duration("-P5DT2H"))` returns `-122`
minutes-from-dayTimeDuration-accumulated

```
altova:minutes-from-dayTimeDuration-accumulated(DayAndTime as xs:duration) as xs:integer XP3.1 XQ3.1
```

Returns the total number of minutes in the duration submitted by the DayAndTime argument (which is of type xs:duration). The minutes in the Day and Time components are added together to give a result that is an integer. Negative durations result in a negative minute value.

**Examples**

- `altova:minutes-from-dayTimeDuration-accumulated(xs:duration("PT60M")) returns 60`
- `altova:minutes-from-dayTimeDuration-accumulated(xs:duration("PT1H")) returns 60, which is the total number of minutes in 1 hour.`
- `altova:minutes-from-dayTimeDuration-accumulated(xs:duration("PT1H40M")) returns 100`
- `altova:minutes-from-dayTimeDuration-accumulated(xs:duration("P1D")) returns 1440, which is the total number of minutes in 1 day.`
- `altova:minutes-from-dayTimeDuration-accumulated(xs:duration("-P1DT60M")) returns -1500`

seconds-from-dayTimeDuration-accumulated

```
altova:seconds-from-dayTimeDuration-accumulated(DayAndTime as xs:duration) as xs:integer XP3.1 XQ3.1
```

Returns the total number of seconds in the duration submitted by the DayAndTime argument (which is of type xs:duration). The seconds in the Day and Time components are added together to give a result that is an integer. Negative durations result in a negative seconds value.

**Examples**

- `altova:seconds-from-dayTimeDuration-accumulated(xs:duration("PT1M")) returns 60, which is the total number of seconds in 1 minute.`
- `altova:seconds-from-dayTimeDuration-accumulated(xs:duration("PT1H")) returns 3600, which is the total number of seconds in 1 hour.`
- `altova:seconds-from-dayTimeDuration-accumulated(xs:duration("PT1H2M")) returns 3720`
- `altova:seconds-from-dayTimeDuration-accumulated(xs:duration("P1D")) returns 86400, which is the total number of seconds in 1 day.`
- `altova:seconds-from-dayTimeDuration-accumulated(xs:duration("-P1DT1M")) returns -86460`

Return the weekday from xs:dateTime or xs:date

These functions return the weekday (as an integer) from xs:dateTime or xs:date. The days of the week are numbered (using the American format) from 1 to 7, with Sunday=1. In the European format, the week starts with Monday (=1). The American format, where Sunday=1, can be set by using the integer 0 where an integer is accepted to indicate the format.

```
weekday-from-dateTime [altova:]
```
altova:weekday-from-dateTime(DateTime as xs:dateTime) as xs:integer

XP3.1
XQ3.1

Takes a date-with-time as its single argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with Sunday=1. If the European format is required (where Monday=1), use the other signature of this function (see next signature below).

Examples

- altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00")) returns 2, which would indicate a Monday.

altova:weekday-from-dateTime(DateTime as xs:dateTime, Format as xs:integer)

as xs:integer
XP3.1
XQ3.1

Takes a date-with-time as its first argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with Monday=1. If the second (integer) argument is 0, then the weekdays are numbered 1 to 7 starting with Sunday=1. If the second argument is an integer other than 0, then Monday=1. If there is no second argument, the function is read as having the other signature of this function (see previous signature).

Examples

- altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00"), 1) returns 1, which would indicate a Monday
- altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00"), 4) returns 1, which would indicate a Monday
- altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00"), 0) returns 2, which would indicate a Monday.

weekday-from-date [altova:]

altova:weekday-from-date(Date as xs:date) as xs:integer

XP3.1
XQ3.1

Takes a date as its single argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with Sunday=1. If the European format is required (where Monday=1), use the other signature of this function (see next signature below).

Examples

- altova:weekday-from-date(xs:date("2014-02-03+01:00")) returns 2, which would indicate a Monday.

altova:weekday-from-date(Date as xs:date, Format as xs:integer) as xs:integer

XP3.1
XQ3.1

Takes a date as its first argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with Monday=1. If the second (Format) argument is 0, then the weekdays are numbered 1 to 7 starting with Sunday=1. If the second argument is an integer other than 0, then Monday=1. If there is no second argument, the function is read as having the other signature of this function (see previous signature).

Examples

- altova:weekday-from-date(xs:date("2014-02-03"), 1) returns 1, which would indicate a Monday
- altova:weekday-from-date(xs:date("2014-02-03"), 4) returns 1, which would indicate a Monday
- altova:weekday-from-date(xs:date("2014-02-03"), 0) returns 2, which would
indicate a Monday.

Return the week number from xs:dateTime or xs:date

These functions return the week number (as an integer) from xs:dateTime or xs:date. Week-numbering is available in the US, ISO/European, and Islamic calendar formats. Week-numbering is different in these calendar formats because the week is considered to start on different days (on Sunday in the US format, Monday in the ISO/European format, and Saturday in the Islamic format).

\[
\text{weeknumber-from-date [altova:]} \\
altova:weeknumber-from-date(Date \text{ as xs:date}, Calendar \text{ as xs:integer}) \text{ as xs:integer} \\
\]

Returns the week number of the submitted date argument as an integer. The second argument (Calendar) specifies the calendar system to follow. Supported Calendar values are:

- 0 = US calendar (week starts Sunday)
- 1 = ISO standard, European calendar (week starts Monday)
- 2 = Islamic calendar (week starts Saturday)

Default is 0.

**Examples**

- \text{altova:weeknumber-from-date(xs:date("2014-03-23"), 0)} returns 13
- \text{altova:weeknumber-from-date(xs:date("2014-03-23"), 1)} returns 12
- \text{altova:weeknumber-from-date(xs:date("2014-03-23"), 2)} returns 13
- \text{altova:weeknumber-from-date(xs:date("2014-03-23"))} returns 13

The day of the date in the examples above (2014-03-23) is Sunday. So the US and Islamic calendars are one week ahead of the European calendar on this day.

\[
\text{weeknumber-from-dateTime [altova:]} \\
altova:weeknumber-from-dateTime(DateTime \text{ as xs:dateTime}, Calendar \text{ as xs:integer}) \text{ as xs:integer} \\
\]

Returns the week number of the submitted dateTime argument as an integer. The second argument (Calendar) specifies the calendar system to follow. Supported Calendar values are:

- 0 = US calendar (week starts Sunday)
- 1 = ISO standard, European calendar (week starts Monday)
- 2 = Islamic calendar (week starts Saturday)

Default is 0.
Examples

- `altova:weeknumber-from-dateTime(xs:dateTime("2014-03-23T00:00:00"), 0)` returns 13
- `altova:weeknumber-from-dateTime(xs:dateTime("2014-03-23T00:00:00"), 1)` returns 12
- `altova:weeknumber-from-dateTime(xs:dateTime("2014-03-23T00:00:00"), 2)` returns 13
- `altova:weeknumber-from-dateTime(xs:dateTime("2014-03-23T00:00:00"))` returns 13

The day of the dateTime in the examples above (2014-03-23T00:00:00) is Sunday. So the US and Islamic calendars are one week ahead of the European calendar on this day.

Build date, time, and duration datatypes from their lexical components

The functions take the lexical components of the `xs:date`, `xs:time`, or `xs:duration` datatype as input arguments and combine them to build the respective datatype.

- `build-date` [altova:]

  ```xml
  altova:build-date(Year as xs:integer, Month as xs:integer, Date as xs:integer) as xs:date
  ``

  The first, second, and third arguments are, respectively, the year, month, and date. They are combined to build a value of `xs:date` type. The values of the integers must be within the correct range of that particular date part. For example, the second argument (for the month part) should not be greater than 12.

  **Examples**

  - `altova:build-date(2014, 2, 03)` returns `2014-02-03`

- `build-time` [altova:]

  ```xml
  altova:build-time(Hours as xs:integer, Minutes as xs:integer, Seconds as xs:integer) as xs:time
  ``

  The first, second, and third arguments are, respectively, the hour (0 to 23), minutes (0 to 59), and seconds (0 to 59) values. They are combined to build a value of `xs:time` type. The values of the integers must be within the correct range of that particular time part. For example, the second (Minutes) argument should not be greater than 59. To add a timezone part to the value, use the other signature of this function (see next signature).

  **Examples**

  - `altova:build-time(23, 4, 57)` returns `23:04:57`

  ```xml
  altova:build-time(Hours as xs:integer, Minutes as xs:integer, Seconds as xs:integer, TimeZone as xs:string) as xs:time
  ```

  The first, second, and third arguments are, respectively, the hour (0 to 23), minutes (0 to 59), and seconds (0 to 59) values. The fourth argument is a string that provides the timezone part.
of the value. The four arguments are combined to build a value of xs:time type. The values of the integers must be within the correct range of that particular time part. For example, the second (Minutes) argument should not be greater than 59.

**Examples**

- `altova:build-time(23, 4, 57, '+1')` returns `23:04:57+01:00`

**build-duration [altova:]**

```xml
altova:build-duration(Years as xs:integer, Months as xs:integer) as xs:yearMonthDuration  XP3.1  XQ3.1
```

Takes two arguments to build a value of type xs:yearMonthDuration. The first arguments provides the Years part of the duration value, while the second argument provides the Months part. If the second (Months) argument is greater than or equal to 12, then the integer is divided by 12; the quotient is added to the first argument to provide the Years part of the duration value while the remainder (of the division) provides the Months part. To build a duration of type xs:dayTimeDuration., see the next signature.

**Examples**

- `altova:build-duration(2, 10)` returns `P2Y10M`
- `altova:build-duration(14, 27)` returns `P16Y3M`
- `altova:build-duration(2, 24)` returns `P4Y`

```xml
altova:build-duration(Days as xs:integer, Hours as xs:integer, Minutes as xs:integer, Seconds as xs:integer) as xs:dayTimeDuration  XP3.1  XQ3.1
```

Takes four arguments and combines them to build a value of type xs:dayTimeDuration. The first argument provides the Days part of the duration value, the second, third, and fourth arguments provide, respectively, the Hours, Minutes, and Seconds parts of the duration value. Each of the three Time arguments is converted to an equivalent value in terms of the next higher unit and the result is used for calculation of the total duration value. For example, 72 seconds is converted to `1M+12S` (1 minute and 12 seconds), and this value is used for calculation of the total duration value. To build a duration of type xs:yearMonthDuration., see the previous signature.

**Examples**

- `altova:build-duration(2, 10, 3, 56)` returns `P2DT10H3M56S`
- `altova:build-duration(1, 0, 100, 0)` returns `P1DT1H40M`
- `altova:build-duration(1, 0, 0, 3600)` returns `P1DT1H`

[ Top ]

**Construct date, dateTime, and time datatypes from string input**

These functions take strings as arguments and construct xs:date, xs:dateTime, or xs:time datatypes. The string is analyzed for components of the datatype based on a submitted pattern argument.

**parse-date [altova:]**

```xml
altova:parse-date(Date as xs:string, DatePattern as xs:string) as xs:date  XP2  XQ1  XP3.1  XQ3.1
```

Returns the input string `Date` as an xs:date value. The second argument `DatePattern`
specifies the pattern (sequence of components) of the input string. **DatePattern** is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

- **D** Date
- **M** Month
- **Y** Year

The pattern in **DatePattern** must match the pattern in **Date**. Since the output is of type **xs:date**, the output will always have the lexical format `YYYY-MM-DD`.

**Examples**

```xml
• altova:parse-date(xs:string("09-12-2014"), "[D]-[M]-[Y]") returns 2014-12-09
• altova:parse-date(xs:string("09-12-2014"), "[M]-[D]-[Y]") returns 2014-09-12
• altova:parse-date("06/03/2014", "/[M]/[D]/[Y]") returns 2014-06-03
• altova:parse-date("06 03 2014", "/[M] [D] [Y]") returns 2014-06-03
• altova:parse-date("6 3 2014", "/[M] [D] [Y]") returns 2014-06-03
```

### parse-datetime [altova:]

```xml
altova:parse-datetime(DateTime as xs:string, DateTimePattern as xs:string) as xs:dateTime
```

Returns the input string **DateTime** as an **xs:dateTime** value. The second argument **DateTimePattern** specifies the pattern (sequence of components) of the input string. **DateTimePattern** is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

- **D** Date
- **M** Month
- **Y** Year
- **H** Hour
- **m** minutes
- **s** seconds

The pattern in **DateTimePattern** must match the pattern in **DateTime**. Since the output is of type **xs:dateTime**, the output will always have the lexical format `YYYY-MM-DDTHH:mm:ss`.

**Examples**

```xml
• altova:parse-datetime(xs:string("09-12-2014 13:56:24"), "/[M]-[D]-[Y]
 [H]:[m]:[s]") returns 2014-09-12T13:56:24
• altova:parse-datetime("time=13:56:24; date=09-12-2014", "time=[H]:[m]:
 [s]; date=[D]-[M]-[Y]") returns 2014-09-12T13:56:24
```

### parse-time [altova:]

```xml
altova:parse-time(Time as xs:string, TimePattern as xs:string) as xs:time
```

Returns the input string Time as an xs:time value. The second argument TimePattern specifies the pattern (sequence of components) of the input string. TimePattern is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

H
m
s

The pattern in TimePattern must match the pattern in Time. Since the output is of type xs:time, the output will always have the lexical format HH:mm:ss.

**Examples**

- `altova:parse-time("13-56-24", "[H]-[m]")` returns `13:56:00`
- `altova:parse-time("time=13h56m24s", "time=[H]h[m]m[s]s")` returns `13:56:24`
- `altova:parse-time("time=24s56m13h", "time=[s]s[m]m[H]h")` returns `13:56:24`

**Age-related functions**

These functions return the age as calculated (i) between one input argument date and the current date, or (ii) between two input argument dates. The `altova:age` function returns the age in terms of years, the `altova:age-details` function returns the age as a sequence of three integers giving the years, months, and days of the age.

```xml
altova:age(StartDate as xs:date) as xs:integer
```

Returns an integer that is the age in years of some object, counting from a start-date submitted as the argument and ending with the current date (taken from the system clock). If the input argument is a date anything greater than or equal to one year in the future, the return value will be negative.

**Examples**

- `altova:age(xs:date("2013-01-15"))` returns `1`
- `altova:age(xs:date("2013-01-16"))` returns `0`
- `altova:age(xs:date("2015-01-15"))` returns `-1`
- `altova:age(xs:date("2015-01-14"))` returns `0`

```xml
altova:age(StartDate as xs:date, EndDate as xs:date) as xs:integer
```

Returns an integer that is the age in years of some object, counting from a start-date that is submitted as the first argument up to an end-date that is the second argument. The return value will be negative if the first argument is one year or more later than the second argument.
Examples

If the current date is 2014-01-15:

- `altova:age(xs:date("2000-01-15"), xs:date("2010-01-15"))` returns 10
- `altova:age(xs:date("2000-01-15"), current-date())` returns 14 if the current date is 2014-01-15
- `altova:age(xs:date("2014-01-15"), xs:date("2010-01-15"))` returns -4

## age-details [altova:]

```
altova:age-details(InputDate as xs:date) as xs:integer)* XP3.1 XQ3.1
```

Returns three integers that are, respectively, the years, months, and days between the date that is submitted as the argument and the current date (taken from the system clock). The sum of the returned years+months+days together gives the total time difference between the two dates (the input date and the current date). The input date may have a value earlier or later than the current date, but whether the input date is earlier or later is not indicated by the sign of the return values; the return values are always positive.

### Examples

If the current date is 2014-01-15:

- `altova:age-details(xs:date("2014-01-16"))` returns (0 0 1)
- `altova:age-details(xs:date("2014-01-14"))` returns (0 0 1)
- `altova:age-details(xs:date("2013-01-16"))` returns (1 0 1)
- `altova:age-details(current-date())` returns (0 0 0)

```
altova:age-details(Date-1 as xs:date, Date-2 as xs:date) as xs:integer)* XP3.1 XQ3.1
```

Returns three integers that are, respectively, the years, months, and days between the two argument dates. The sum of the returned years+months+days together gives the total time difference between the two input dates; it does not matter whether the earlier or later of the two dates is submitted as the first argument. The return values do not indicate whether the input date occurs earlier or later than the current date. Return values are always positive.

### Examples

- `altova:age-details(xs:date("2014-01-16"), xs:date("2014-01-15"))` returns (0 0 1)
- `altova:age-details(xs:date("2014-01-15"), xs:date("2014-01-16"))` returns (0 0 1)

[Top]

### 19.1.2.1.3 XPath/XQuery Functions: Geolocation

The following geolocation XPath/XQuery extension functions are supported in the current version of MapForce and can be used in (i) XPath expressions in an XSLT context, or (ii) XQuery expressions in an XQuery document.
Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the Altova extension functions namespace, http://www.altova.com/xslt-extensions, and are indicated in this section with the prefix altova:, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

| XPath functions (used in XPath expressions in XSLT): | XP1  XP2  XP3.1 |
| XSLT functions (used in XPath expressions in XSLT): | XSLT1 XSLT2 XSLT3 |
| XQuery functions (used in XQuery expressions in XQuery): | XQ1  XQ3.1 |

#### format-geolocation [altova:]

```xml
<altova:format-geolocation(Latitude as xs:decimal, Longitude as xs:decimal, GeolocationOutputStringFormat as xs:integer) as xs:string>
```

Takes the latitude and longitude as the first two arguments, and outputs the geolocation as a string. The third argument, GeolocationOutputStringFormat, is the format of the geolocation output string; it uses integer values from 1 to 4 to identify the output string format (see 'Geolocation output string formats' below). Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

**Note:** The image-exif-data function and the Exif metadata’s attributes can be used to supply the input strings.

- **Examples**
  - `altova:format-geolocation(33.33, -22.22, 4)` returns the xs:string "33.33° 22.22"N
  - `altova:format-geolocation(33.33, -22.22, 2)` returns the xs:string "33.33N 22.22W"
  - `altova:format-geolocation(-33.33, 22.22, 2)` returns the xs:string "33.33S 22.22E"
  - `altova:format-geolocation(33.33, -22.22, 1)` returns the xs:string "33° 19'48.00"S 22°13'12.00"E"

- **Geolocation output string formats:**
  The supplied latitude and longitude is formatted in one of the output formats given below. The desired format is identified by its integer ID (1 to 4). Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

<table>
<thead>
<tr>
<th>Format ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Degrees, minutes, decimal seconds, with suffixed orientation (N/S, E/W)</td>
</tr>
<tr>
<td></td>
<td>D°M'S''S''N/S D°M'S''S''E/W</td>
</tr>
</tbody>
</table>

**Example:** 33°55'11.11"N  22°44'66.66"W
Decimal degrees, with suffixed orientation (N/S, E/W)

Example: 33.33N 22.22W

Degrees, minutes, decimal seconds, with prefixed sign (+/-); plus sign for (N/E) is optional

Example: 33°55'11.11"  -22°44'66.66"

Decimal degrees, with prefixed sign (+/-); plus sign for (N/E) is optional

Example: 33.33 -22.22

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute Geolocation from standard Exif metadata tags. Geolocation is a concatenation of four Exif tags: GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (see table below).

<table>
<thead>
<tr>
<th>GPSLatitude</th>
<th>GPSLatitudeRef</th>
<th>GPSLongitude</th>
<th>GPSLongitudeRef</th>
<th>Geolocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>33 51</td>
<td>S</td>
<td>151 13</td>
<td>E</td>
<td>33°51'21.91&quot;S 151°13'11.73&quot;E</td>
</tr>
</tbody>
</table>

parse-geolocation [altova:]

```
altova:parse-geolocation(GeolocationInputString as xs:string) as xs:decimal+
XP3.1 XQ3.1
```

 Parses the supplied GeolocationInputString argument and returns the geolocation's latitude and longitude (in that order) as a sequence two xs:decimal items. The formats in which the geolocation input string can be supplied are listed below.

Note: The image-exif-data function and the Exif metadata's @Geolocation attribute can be used to supply the geolocation input string (see example below).

Examples

- `altova:parse-geolocation("33.33 -22.22")` returns the sequence of two xs:decimals (33.33, 22.22)
- `altova:parse-geolocation("48°51'29.66"N 24°17'40.26"E")` returns the
sequence of two `xs:decimals` (48.8582222222222, 24.2945)

- `altova:parse-geolocation(['48°51′29.6″N 24°17′40.2″W'])` returns the sequence of two `xs:decimals` (48.8582222222222, 24.2945)

- `altova:parse-geolocation( image-exif-data(/MyImages/ Image20141130.01) /@Geolocation )` returns a sequence of two `xs:decimals`:

**Geolocation input string formats:**
The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from \(+90\) to \(-90\) (\(N\) to \(S\)). Longitude values range from \(+180\) to \(-180\) (\(E\) to \(W\)).

**Note:** If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow ('') while unit indicators that are escaped are highlighted in blue ('"').

- **Degrees, minutes, decimal seconds, with suffixed orientation \((N/S, W/E)\)**
  
  \(D°M′.SS″N/S\) \(D°M′.SS″W/E\)
  
  **Example:** \(33°55′11.11″N\) \(22°44′55.25″W\)

- **Degrees, minutes, decimal seconds, with prefixed sign \((+/−)\); the plus sign for \((N/W)\) is optional**
  
  \(+/−D°M′.SS″\) \(+/−D°M′.SS″\)
  
  **Example:** \(33°55′11.11″\) \(-22°44′55.25″\)

- **Degrees, decimal minutes, with suffixed orientation \((N/S, W/E)\)**
  
  \(D°.MM′N/S\) \(D°.MM′W/E\)
  
  **Example:** \(33°55.55′N\) \(22°44.44′W\)

- **Degrees, decimal minutes, with prefixed sign \((+/−)\); the plus sign for \((N/W)\) is optional**
  
  \(+/−D°.MM′\) \(+/−D°.MM′\)
  
  **Example:** \(+33°55.55′\) \(-22°44.44′\)

- **Decimal degrees, with suffixed orientation \((N/S, W/E)\)**
  
  \(D.DDN/S\) \(D.DDW/E\)
  
  **Example:** \(33.33N\) \(22.22W\)

- **Decimal degrees, with prefixed sign \((+/−)\); the plus sign for \((N/W)\) is optional**
  
  \(+/−D.DD\) \(+/−D.DD\)
  
  **Example:** \(33.33\) \(-22.22\)

**Examples of format-combinations:**

- \(33.33N\) \(-22°44′55.25″\)
- \(33.33\) \(22°44′55.25″W\)
- \(33.33\) \(22.45\)
Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute *Geolocation* from standard Exif metadata tags. *Geolocation* is a concatenation of four Exif tags: GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (see table below).

<table>
<thead>
<tr>
<th>GPSLatitude</th>
<th>GPSLatitude Ref</th>
<th>GPSLongitude</th>
<th>GPSLongitude Ref</th>
<th>Geolocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>33 51</td>
<td>S</td>
<td>151 13</td>
<td>E</td>
<td>33°51'21.91&quot;S 151°</td>
</tr>
<tr>
<td>21.91</td>
<td></td>
<td>11.73</td>
<td></td>
<td>13'11.73&quot;E</td>
</tr>
</tbody>
</table>

geolocation-distance-km [altova:]

```
altova:geolocation-distance-km(GeolocationInputString-1 as xs:string,
GeolocationInputString-2 as xs:string) as xs:decimal
```

Calculates the distance between two geolocations in kilometers. The formats in which the geolocation input string can be supplied are listed below. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The *image-exif-data* function and the Exif metadata's @Geolocation attribute can be used to supply geolocation input strings.

Examples

- `altova:geolocation-distance-km("33.33  -22.22", "48°51'29.6"N  24°17'40.2"W")` returns the `xs:decimal 4183.08132372392`

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow ("), while unit indicators that are escaped are highlighted in blue ("').

- **Degrees, minutes, decimal seconds, with suffixed orientation (N/S, W/E)**

  D°M'S.S'SS"N/S  D°M'S.S'SS"W/E

  *Example:* 33°55'11.11"N  22°44'55.25"W

- **Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (N/W) is optional**

  +/-D°M'S.S'SS"  +/-D°M'S.S'SS"
Example: 33°55'11.11"  -22°44'55.25"

- Degrees, decimal minutes, with suffixed orientation (N/S, W/E)
  D°M.MM'N/S  D°M.MM'W/E
  Example: 33°55.55'N  22°44.44'W

- Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (N/W) is optional
  +/-D°M.MM'  +/-D°M.MM'
  Example: +33°55.55'  -22°44.44'

- Decimal degrees, with suffixed orientation (N/S, W/E)
  D.DD N/S  D.DD W/E
  Example: 33.33N  22.22W

- Decimal degrees, with prefixed sign (+/-); the plus sign for (N/W) is optional
  +/-D.DD  +/-D.DD
  Example: 33.33  -22.22

Examples of format-combinations:
33.33N  -22°44'55.25"
33.33  22°44'55.25"W
33.33  22.22

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute Geolocation from
standard Exif metadata tags. Geolocation is a concatenation of four Exif tags:
GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added
(see table below).

<table>
<thead>
<tr>
<th>GPSLatitude</th>
<th>GPSLatitudeRef</th>
<th>GPSLongitude</th>
<th>GPSLongitudeRef</th>
<th>Geolocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>33.51</td>
<td>S</td>
<td>151.13</td>
<td>E</td>
<td>33°51'21.91&quot;S 151°13'11.73&quot;E</td>
</tr>
<tr>
<td>21.91</td>
<td></td>
<td>11.73</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

geolocation-distance-mi [altova:]

altova:geolocation-distance-mi(GeolocationInputString-1 as xs:string, GeolocationInputString-2 as xs:string) as xs:decimal

Calculates the distance between two geolocations in miles. The formats in which a
geolocation input string can be supplied are listed below. Latitude values range from +90 to
-90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The image-exif-data function and the Exif metadata's @Geolocation attribute can
be used to supply geolocation input strings.

Examples
• `altova:geolocation-distance-mi(33.33 -22.22, 48°51'29.6"N 24°17'40.2"W)` returns the `xs:decimal` 2599.40652340653

Geolocation input string formats:
The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

**Note:** If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow ("), while unit indicators that are escaped are highlighted in blue (").

- **Degrees, minutes, decimal seconds, with suffixed orientation (N/S, W/E)**
  
  \[D°M'S.SS"N/S \quad D°M'S.SS"W/E\]
  
  **Example:** 33°55'11.11"N 22°44'55.25"W

- **Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (N/W) is optional**
  
  \[+/D°M'S.SS" \quad +/-D°M'S.SS"\]
  
  **Example:** 33°55'11.11" -22°44'55.25"W

- **Degrees, decimal minutes, with suffixed orientation (N/S, W/E)**
  
  \[D°M.MM'N/S \quad D°M.MM'W/E\]
  
  **Example:** 33°55.55'N 22°44.44'W

- **Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (N/W) is optional**
  
  \[+/D°M.MM' \quad +/-D°M.MM'\]
  
  **Example:** +33°55.55' -22°44.44'

- **Decimal degrees, with suffixed orientation (N/S, W/E)**
  
  \[D.DDN/S \quad D.DDW/E\]
  
  **Example:** 33.33N 22.22W

- **Decimal degrees, with prefixed sign (+/-); the plus sign for (N/W) is optional**
  
  \[+/D.DD \quad +/-D.DD\]
  
  **Example:** 33.33 -22.22

**Examples of format-combinations:**

33.33N -22°44'55.25"
33.33 22°44'55.25"W
33.33 22.45

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute `Geolocation` from
standard Exif metadata tags. Geolocation is a concatenation of four Exif tags:
GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added
(see table below).

<table>
<thead>
<tr>
<th>GPSLatitude</th>
<th>GPSLatitudeRef</th>
<th>GPSLongitude</th>
<th>GPSLongitudeRef</th>
<th>Geolocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>33 51</td>
<td>S</td>
<td>151 13</td>
<td>E</td>
<td>33°51'21.91&quot;S 151°13'11.73&quot;E</td>
</tr>
</tbody>
</table>

**geolocation-within-polygon [altova:]**

`altova:geolocation-within-polygon(Geolocation as xs:string, ((PolygonPoint as xs:string)+)) as xs:boolean` XP3.1 XQ3.1

Determines whether Geolocation (the first argument) is within the polygonal area described
by the PolygonPoint arguments. If the PolygonPoint arguments do not form a closed figure
(formed when the first point and the last point are the same), then the first point is implicitly
added as the last point in order to close the figure. All the arguments (Geolocation and
PolygonPoint+) are given by geolocation input strings (formats listed below). If the
Geolocation argument is within the polygonal area, then the function returns true();
otherwise it returns false(). Latitude values range from +90 to -90 (n to s). Longitude values
range from +180 to -180 (E to W).

**Note:** The `image-exif-data` function and the Exif metadata’s @Geolocation attribute can
be used to supply geolocation input strings.

**Examples**

- `altova:geolocation-within-polygon("33 -22", ("58 -32", "-78 -55", "48°51'29.6"N 24°17'40.2"W"))` returns true()

**Geolocation input string formats:**

The geolocation input string must contain latitude and longitude (in that order) separated
by whitespace. Each can be in any of the following formats. Combinations are allowed.
So latitude can be in one format and longitude can be in another. Latitude values range
from +90 to -90 (n to s). Longitude values range from +180 to -180 (E to W).

**Note:** If single quotes or double quotes are used to delimit the input string argument,
this will create a mismatch with the single quotes or double quotes that are used,
respectively, to indicate minute-values and second-values. In such cases, the quotes
that are used for indicating minute-values and second-values must be escaped by
doubling them. In the examples in this section, quotes used to delimit the input string
are highlighted in yellow (’’) while unit indicators that are escaped are highlighted in blue
(“”).
- Degrees, minutes, decimal seconds, with suffixed orientation (N/S, W/E)
  \[\text{D}^\circ\text{M}'\text{S}.SS" \text{N/S} \text{ D}^\circ\text{M}'\text{S}.SS" \text{W/E}\]
  \textit{Example:} 33°55'11.11"N 22°44'55.25"W

- Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (N/W) is optional
  \[\pm\text{D}^\circ\text{M}'\text{S}.SS"\]
  \textit{Example:} 33°55'11.11" 22°44'55.25"W

- Degrees, decimal minutes, with suffixed orientation (N/S, W/E)
  \[\text{D}.D\text{M}.M'M" \text{N/S} \text{ D}.D\text{M}.M'M" \text{W/E}\]
  \textit{Example:} 33°55.55'N 22°44.44'W

- Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (N/W) is optional
  \[\pm\text{D}.D\text{M}.M'M"\]
  \textit{Example:} +33°55.55' 22°44.44'

- Decimal degrees, with suffixed orientation (N/S, W/E)
  \[\text{D}.D\text{D} \text{N/S} \text{ D}.D\text{D} \text{W/E}\]
  \textit{Example:} 33.33N 22.22W

- Decimal degrees, with prefixed sign (+/-); the plus sign for (N/W) is optional
  \[\pm\text{D}.D\text{D}\]
  \textit{Example:} 33.33 22.22

\textbf{Examples of format-combinations:}
33.33N -22°44'55.25"W
33.33 22°44'55.25"W
33.33 22.45

\textbf{Altova Exif Attribute: Geolocation}

The Altova XPath/XQuery Engine generates the custom attribute \texttt{Geolocation} from standard Exif metadata tags. \texttt{Geolocation} is a concatenation of four Exif tags: GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (see table below).

<table>
<thead>
<tr>
<th>GPSLatitude</th>
<th>GPSLatitudeRef</th>
<th>GPSLongitude</th>
<th>GPSLongitudeRef</th>
<th>Geolocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>33°51'21.91&quot;S</td>
<td>S</td>
<td>151°13'11.73&quot;E</td>
<td></td>
<td>33°51'21.91&quot;S 151°13'11.73&quot;E</td>
</tr>
</tbody>
</table>

\texttt{geolocation-within-rectangle [altova:]
\texttt{altova:geolocation-within-rectangle(Geolocation as xs:string, RectCorner-1 as xs:string, RectCorner-2 as xs:string) as xs:boolean}} XQP3.XQ3.1

Determine whether \texttt{Geolocation} (the first argument) is within the rectangle defined by the
second and third arguments, `RectCorner-1` and `RectCorner-2`, which specify opposite corners of the rectangle. All the arguments (Geolocation, `RectCorner-1` and `RectCorner-2`) are given by geolocation input strings (formats listed below). If the Geolocation argument is within the rectangle, then the function returns `true()`; otherwise it returns `false()`.

Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

**Note:** The `image-exif-data` function and the Exif metadata’s `@Geolocation` attribute can be used to supply geolocation input strings.

**Examples**

- `altova:geolocation-within-rectangle("33 -22", "58 -32", "-48 24")` returns `true()`
- `altova:geolocation-within-rectangle("33 -22", "58 -32", "48 24")` returns `false()`
- `altova:geolocation-within-rectangle("33 -22", "58 -32", "48°51'29.6"S 24°17'40.2"W")` returns `true()`

**Geolocation input string formats:**

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

**Note:** If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (’’) while unit indicators that are escaped are highlighted in blue (””).

- **Degrees, minutes, decimal seconds, with suffixed orientation (N/S, W/E)**
  
  $D°M'S.S'SS''N/S \quad D°M'S.S'SSS''W/E$

  **Example:** 33°55'11.11"N  22°44'55.25"W

- **Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (N/W) is optional**
  
  $+/-D°M'S.S'SS''  +/-D°M'S.SSS''$

  **Example:** 33°55'11.11"  -22°44'55.25"W

- **Degrees, decimal minutes, with suffixed orientation (N/S, W/E)**
  
  $D°M.MM'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'M'I
- **Decimal degrees, with suffixed orientation (N/S, W/E)**
  \[ \text{D.DDN/S D.DDW/E} \]
  \[ \text{Example: 33.33N 22.22W} \]

- **Decimal degrees, with prefixed sign (+/-); the plus sign for (N/W) is optional**
  \[ \text{+/-D.DD +/-D.DD} \]
  \[ \text{Example: 33.33 -22.22} \]

**Examples of format-combinations:**
- 33.33N -22°44'55.25"
- 33.33 22°44'55.25"W
- 33.33 22.45

*Altova Exif Attribute: Geolocation*

The Altova XPath/XQuery Engine generates the custom attribute Geolocation from standard Exif metadata tags. Geolocation is a concatenation of four Exif tags:

GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (see table below).

<table>
<thead>
<tr>
<th>GPSLatitude</th>
<th>GPSLatitude Ref</th>
<th>GPSLongitude</th>
<th>GPSLongitude Ref</th>
<th>Geolocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>33 51 21.91</td>
<td>S</td>
<td>151 13 11.73</td>
<td>E</td>
<td>33°51'21.91&quot;S 151°13'11.73&quot;E</td>
</tr>
</tbody>
</table>

19.1.2.1.4 *XPath/XQuery Functions: Image-Related*

The following image-related XPath/XQuery extension functions are supported in the current version of MapForce and can be used in (i) XPath expressions in an XSLT context, or (ii) XQuery expressions in an XQuery document.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the *Altova extension functions namespace*, [http://www.altova.com/xslt-extensions](http://www.altova.com/xslt-extensions), and are indicated in this section with the prefix *altova:*, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

<table>
<thead>
<tr>
<th>XPath functions (used in XPath expressions in XSLT):</th>
<th>XP1 XP2 XP3.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLT functions (used in XPath expressions in XSLT):</td>
<td>XSLT1 XSLT2 XSLT3</td>
</tr>
</tbody>
</table>
### XQuery functions (used in XQuery expressions in XQuery):

<table>
<thead>
<tr>
<th>XQ1</th>
<th>XQ3.1</th>
</tr>
</thead>
</table>

#### suggested-image-file-extension [altova:]

**altova:suggested-image-file-extension(\texttt{Base64String as string}) AS string?**

\texttt{XP3.1 XQ3.1}

Takes the Base64 encoding of an image file as its argument and returns the file extension of the image as recorded in the Base64-encoding of the image. The returned value is a suggestion based on the image type information available in the encoding. If this information is not available, then an empty string is returned. This function is useful if you wish to save a Base64 image as a file and wish to dynamically retrieve an appropriate file extension.

**Examples**

- \texttt{altova:suggested-image-file-extension(/MyImages/MobilePhone/Image20141130.01)} returns \texttt{'jpg'}
- \texttt{altova:suggested-image-file-extension($XML1/Staff/Person/@photo)} returns \texttt{''}

In the examples above, the nodes supplied as the argument of the function are assumed to contain a Base64-encoded image. The first example retrieves \texttt{jpg} as the file's type and extension. In the second example, the submitted Base64 encoding does not provide usable file extension information.

#### image-exif-data [altova:]

**altova:image-exif-data(\texttt{Base64BinaryString as string}) AS element?**

\texttt{XP3.1 XQ3.1}

Takes a Base64-encoded JPEG image as its argument and returns an element called \texttt{Exif} that contains the Exif metadata of the image. The Exif metadata is created as attribute-value pairs of the \texttt{Exif} element. The attribute names are the Exif data tags found in the Base64 encoding. The list of Exif-specification tags is given below. If a vendor-specific tag is present in the Exif data, this tag and its value will also be returned as an attribute-value pair.

Additional to the standard Exif metadata tags (see list below), Altova-specific attribute-value pairs are also generated. These Altova Exif attributes are listed below.

**Examples**

- To access any one attribute, use the function like this:
  \texttt{image-exif-data(/MyImages/Image20141130.01)/@GPSLatitude}
  \texttt{image-exif-data(/MyImages/Image20141130.01)/@Geolocation}
- To access all the attributes, use the function like this:
  \texttt{image-exif-data(/MyImages/Image20141130.01)/@*}
- To access the names of all the attributes, use the following expression:
  \texttt{for $i in image-exif-data(/MyImages/Image20141130.01)/@* return name($i)}

  This is useful to find out the names of the attributes returned by the function.

**Altova Exif Attribute: Geolocation**

The Altova XPath/XQuery Engine generates the custom attribute \texttt{Geolocation} from
standard Exif metadata tags. Geolocation is a concatenation of four Exif tags:
GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (see table below).

<table>
<thead>
<tr>
<th>GPSLatitude</th>
<th>GPSLatitudeRef</th>
<th>GPSLongitude</th>
<th>GPSLongitudeRef</th>
<th>Geolocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>33°51'21.91&quot;</td>
<td>S</td>
<td>151°13'11.73&quot;</td>
<td>E</td>
<td>33°51'21.91&quot;S 151°13'11.73&quot;E</td>
</tr>
</tbody>
</table>

Altova Exif Attribute: OrientationDegree

The Altova XPath/XQuery Engine generates the custom attribute OrientationDegree from the Exif metadata tag Orientation.

OrientationDegree translates the standard Exif tag Orientation from an integer value (1, 8, 3, or 6) to the respective degree values of each (0, 90, 180, 270), as shown in the figure below. Note that there are no translations of the Orientation values of 2, 4, 5, 7. (These orientations are obtained by flipping image 1 across its vertical center axis to get the image with a value of 2, and then rotating this image in 90-degree jumps clockwise to get the values of 7, 4, and 5, respectively).

Listing of standard Exif meta tags
- ImageWidth
- ImageLength
- BitsPerSample
• Compression
• PhotometricInterpretation
• Orientation
• SamplesPerPixel
• PlanarConfiguration
• YCbCrSubSampling
• YCbCrPositioning
• XResolution
• YResolution
• ResolutionUnit
• StripOffsets
• RowsPerStrip
• StripByteCounts
• JPEGInterchangeFormat
• JPEGInterchangeFormatLength
• TransferFunction
• WhitePoint
• PrimaryChromaticities
• YCbCrCoefficients
• ReferenceBlackWhite
• DateTime
• ImageDescription
• Make
• Model
• Software
• Artist
• Copyright

---------------------------

• ExifVersion
• FlashpixVersion
• ColorSpace
• ComponentsConfiguration
• CompressedBitsPerPixel
• PixelXDimension
• PixelYDimension
• MakerNote
• UserComment
• RelatedSoundFile
• DateTimeOriginal
• DateTimeDigitized
• SubSecTime
• SubSecTimeOriginal
• SubSecTimeDigitized
• ExposureTime
• FNumber
• ExposureProgram
• SpectralSensitivity
• ISOSpeedRatings
• OECF
• ShutterSpeedValue
• ApertureValue
• BrightnessValue
• ExposureBiasValue
• MaxApertureValue
• SubjectDistance
• MeteringMode
• LightSource
• Flash
• FocalLength
• SubjectArea
• FlashEnergy
• SpatialFrequencyResponse
• FocalPlaneXResolution
• FocalPlaneYResolution
• FocalPlaneResolutionUnit
• SubjectLocation
• ExposureIndex
• SensingMethod
• FileSource
• SceneType
• CFAPattern
• CustomRendered
• ExposureMode
• WhiteBalance
• DigitalZoomRatio
• FocalLengthIn35mmFilm
• SceneCaptureType
• GainControl
• Contrast
• Saturation
• Sharpness
• DeviceSettingDescription
• SubjectDistanceRange
• ImageUniqueID

-------------------------------

• GPSVersionID
• GPSLatitudeRef
• GPSLatitude
• GPSLongitudeRef
• GPSLongitude
• GPSAltitudeRef
• GPSAltitude
• GPSTimeStamp
• GPSSatellites
• GPSStatus
• GPSMeasureMode
• GPSDOP
• GPSSpeedRef
• GPSSpeed
• GPSTrackRef
• GPSTrack
• GPSImgDirectionRef
• GPSImgDirection
• GPSMapDatum
• GPSDestLatitudeRef
• GPSDestLatitude
• GPSDestLongitudeRef
• GPSDestLongitude
19.1.2.1.5 XPath/XQuery Functions: Numeric

Altova’s numeric extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova’s XPath 3.0 and XQuery 3.0 engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the Altova extension functions namespace, http://www.altova.com/xslt-extensions, and are indicated in this section with the prefix altova:, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

<table>
<thead>
<tr>
<th>XPath functions (used in XPath expressions in XSLT):</th>
<th>XP1 XP2 XP3.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLT functions (used in XPath expressions in XSLT):</td>
<td>XSLT1 XSLT2 XSLT3</td>
</tr>
<tr>
<td>XQuery functions (used in XQuery expressions in XQuery):</td>
<td>XQ1 XQ3.1</td>
</tr>
</tbody>
</table>

Auto-numbering functions

generate-auto-number [altova:]

\[ \text{altova:generate-auto-number(ID as xs:string, StartsWith as xs:double, Increment as xs:double, ResetOnChange as xs:string) as xs:integer} \]

Generates a number each time the function is called. The first number, which is generated the first time the function is called, is specified by the StartsWith argument. Each subsequent call to the function generates a new number, this number being incremented over the previously generated number by the value specified in the Increment argument. In effect, the altova:generate-auto-number function creates a counter having a name specified by the ID argument, with this counter being incremented each time the function is called. If the value of the ResetOnChange argument changes from that of the previous function call, then the value of the number to be generated is reset to the StartsWith value. Auto-numbering can also be reset by using the altova:reset-auto-number function.
Examples

- `altova:generate-auto-number("ChapterNumber", 1, 1, "SomeString")` will return one number each time the function is called, starting with 1, and incrementing by 1 with each call to the function. As long as the fourth argument remains "SomeString" in each subsequent call, the incrementing will continue. When the value of the fourth argument changes, the counter (called ChapterNumber) will reset to 1. The value of ChapterNumber can also be reset by a call to the `altova:reset-auto-number` function, like this: `altova:reset-auto-number("ChapterNumber")`.

reset-auto-number [altova:]

`altova:reset-auto-number(ID as xs:string) as xs:string`  
This function resets the number of the auto-numbering counter named in the ID argument. The number is reset to the number specified by the StartsWith argument of the `altova:generate-auto-number` function that created the counter named in the ID argument.

Examples

- `altova:reset-auto-number("ChapterNumber")` resets the number of the auto-numbering counter named ChapterNumber that was created by the `altova:generate-auto-number` function. The number is reset to the value of the StartsWith argument of the `altova:generate-auto-number` function that created ChapterNumber.

Numeric functions

hex-string-to-integer [altova:]

`altova:hex-string-to-integer(HexString as xs:string) as xs:integer`  
Takes a string argument that is the Base-16 equivalent of an integer in the decimal system (Base-10), and returns the decimal integer.

Examples

- `altova:hex-string-to-integer('1')` returns 1
- `altova:hex-string-to-integer('9')` returns 9
- `altova:hex-string-to-integer('A')` returns 10
- `altova:hex-string-to-integer('B')` returns 11
- `altova:hex-string-to-integer('F')` returns 15
- `altova:hex-string-to-integer('G')` returns an error
- `altova:hex-string-to-integer('10')` returns 16
- `altova:hex-string-to-integer('01')` returns 1
- `altova:hex-string-to-integer('20')` returns 32
- `altova:hex-string-to-integer('21')` returns 33
- `altova:hex-string-to-integer('5A')` returns 90
- `altova:hex-string-to-integer('USA')` returns an error

integer-to-hex-string [altova:]

[ Top ]
altova:integer-to-hex-string(Integer as xs:integer) as xs:string  XP3.1  XQ3.1
Takes an integer argument and returns its Base-16 equivalent as a string.

Examples
- altova:integer-to-hex-string(1) returns '1'
- altova:integer-to-hex-string(9) returns '9'
- altova:integer-to-hex-string(10) returns 'A'
- altova:integer-to-hex-string(11) returns 'B'
- altova:integer-to-hex-string(15) returns 'F'
- altova:integer-to-hex-string(16) returns '10'
- altova:integer-to-hex-string(32) returns '20'
- altova:integer-to-hex-string(33) returns '21'
- altova:integer-to-hex-string(90) returns '5A'

Number-formatting functions

generate-auto-number [altova:]

altova:generate-auto-number(ID as xs:string, StartsWith as xs:double, Increment as xs:double, ResetOnChange as xs:string) as xs:integer  XP1 XP2 XQ1 XP3.1 XQ3.1
Generates a number each time the function is called. The first number, which is generated the first time the function is called, is specified by the StartsWith argument. Each subsequent call to the function generates a new number, this number being incremented over the previously generated number by the value specified in the Increment argument. In effect, the altova:generate-auto-number function creates a counter having a name specified by the ID argument, with this counter being incremented each time the function is called. If the value of the ResetOnChange argument changes from that of the previous function call, then the value of the number to be generated is reset to the StartsWith value. Auto-numbering can also be reset by using the altova:reset-auto-number function.

Examples
- altova:generate-auto-number("ChapterNumber", 1, 1, "SomeString") will return one number each time the function is called, starting with 1, and incrementing by 1 with each call to the function. As long as the fourth argument remains "SomeString" in each subsequent call, the incrementing will continue. When the value of the fourth argument changes, the counter (called ChapterNumber) will reset to 1. The value of ChapterNumber can also be reset by a call to the altova:reset-auto-number function, like this: altova:reset-auto-number("ChapterNumber").

19.1.2.1.6  XPath/XQuery Functions: Sequence

Altova's sequence extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova's XPath 3.0 and XQuery 3.0 engines. They are available in XPath/XQuery contexts.
Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the Altova extension functions namespace, http://www.altova.com/xslt-extensions, and are indicated in this section with the prefix altova, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

<table>
<thead>
<tr>
<th>XPath functions (used in XPath expressions in XSLT):</th>
<th>XP1</th>
<th>XP2</th>
<th>XP3.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLT functions (used in XPath expressions in XSLT):</td>
<td>XSLT1</td>
<td>XSLT2</td>
<td>XSLT3</td>
</tr>
<tr>
<td>XQuery functions (used in XQuery expressions in XQuery):</td>
<td>XQ1</td>
<td>XQ3.1</td>
<td></td>
</tr>
</tbody>
</table>

**attributes [altova:]

```xml
altova:attributes (AttributeName as xs:string) as attribute()* XP3.1 XQ3.1
```

Returns all attributes that have a local name which is the same as the name supplied in the input argument, AttributeName. The search is case-sensitive and conducted along the attribute:: axis. This means that the context node must be the parent element node.

**Examples**

- `altova:attributes("MyAttribute")` returns `MyAttribute()`*

```xml
altova:attributes (AttributeName as xs:string, SearchOptions as xs:string) as attribute()* XP3.1 XQ3.1
```

Returns all attributes that have a local name which is the same as the name supplied in the input argument, AttributeName. The search is case-sensitive and conducted along the attribute:: axis. The context node must be the parent element node. The second argument is a string containing option flags. Available flags are:

- **r** = switches to a regular-expression search; AttributeName must then be a regular-expression search string;
- **f** = If this option is specified, then AttributeName provides a full match; otherwise AttributeName need only partially match an attribute name to return that attribute. For example: if **f** is not specified, then MyAtt will return MyAttribute;
- **i** = switches to a case-insensitive search;
- **p** = includes the namespace prefix in the search; AttributeName should then contain the namespace prefix, for example: `altova:MyAttribute`.

The flags can be written in any order. Invalid flags will generate errors. One or more flags can be omitted. The empty string is allowed, and will produce the same effect as the function having only one argument (previous signature). However, an empty sequence is not allowed as the second argument.

**Examples**

- `altova:attributes("MyAttribute", "rfip")` returns `MyAttribute()`*
- `altova:attributes("MyAttribute", "pri")` returns `MyAttribute()`*
- `altova:attributes("MyAtt", "rip")` returns `MyAttribute()`*
- `altova:attributes("MyAttributes", "rfip")` returns no match
- `altova:attributes("MyAttribute", "")` returns `MyAttribute()`*
Appendices Engine information

- `altova:attributes("MyAttribute", "Rip")` returns an unrecognized-flag error.
- `altova:attributes("MyAttribute", )` returns a missing-second-argument error.

### elements [altova:]

```xml
altova:elements(ElementName as xs:string) as element()* XP3.1 XQ3.1
```

Returns all elements that have a local name which is the same as the name supplied in the input argument, `ElementName`. The search is case-sensitive and conducted along the child::* axis. The context node must be the parent node of the element/s being searched for.

**Examples**

- `altova:elements("MyElement")` returns `MyElement()` *

```xml
altova:elements(ElementName as xs:string, SearchOptions as xs:string) as element()* XP3.1 XQ3.1
```

Returns all elements that have a local name which is the same as the name supplied in the input argument, `ElementName`. The search is case-sensitive and conducted along the child::* axis. The context node must be the parent node of the element/s being searched for. The second argument is a string containing option flags. Available flags are:
- `r` = switches to a regular-expression search; `ElementName` must then be a regular-expression search string;
- `f` = If this option is specified, then `ElementName` provides a full match; otherwise `ElementName` need only partially match an element name to return that element. For example: if `f` is not specified, then `MyElem` will return `MyElement`;
- `i` = switches to a case-insensitive search;
- `p` = includes the namespace prefix in the search; `ElementName` should then contain the namespace prefix, for example: `altova:MyElement`.

The flags can be written in any order. Invalid flags will generate errors. One or more flags can be omitted. The empty string is allowed, and will produce the same effect as the function having only one argument (**previous signature**). However, an empty sequence is not allowed.

**Examples**

- `altova:elements("MyElement", "rip")` returns `MyElement()` *
- `altova:elements("MyElement", "pri")` returns `MyElement()` *
- `altova:elements("MyElement", "")` returns `MyElement()` *
- `altova:attributes("MyElem", "rip")` returns `MyElement()` *
- `altova:attributes("MyElements", "rfip")` returns no match
- `altova:elements("MyElement", "Rip")` returns an unrecognized-flag error.

### find-first [altova:]

```xml
altova:find-first((Sequence as item())*, (Condition( Sequence-Item as xs:boolean)) as item())? XP3.1 XQ3.1
```

This function takes two arguments. The first argument is a sequence of one or more items of any datatype. The second argument, `Condition`, is a reference to an XPath function that takes one argument (has an arity of 1) and returns a boolean. Each item of `Sequence` is submitted, in turn, to the function referenced in `Condition`. (**Remember:** This function takes a single argument.) The first `Sequence` item that causes the function in `Condition` to
evaluate to \texttt{true()} is returned as the result of \texttt{altova:find-first}, and the iteration stops.

\textbf{Examples}

- \texttt{altova:find-first(5 to 10, function($a) \ {$a \ mod \ 2 = 0\}) \ returns} \ x{s:integer \ 6}

  The \texttt{Condition} argument references the XPath 3.0 inline function, \texttt{function()}, which declares an inline function named \texttt{$a$} and then defines it. Each item in the \texttt{Sequence} argument of \texttt{altova:find-first} is passed, in turn, to \texttt{$a$} as its input value. The input value is tested on the condition in the function definition ($a \mod \ 2 = 0$). The first input value to satisfy this condition is returned as the result of \texttt{altova:find-first} (in this case \texttt{6}).

- \texttt{altova:find-first((1 to 10), (function($a) \ {$a+3=7\}) \ returns} \ x{s:integer \ 4}

\textbf{Further examples}

If the file \texttt{C:\Temp\Customers.xml} exists:

- \texttt{altova:find-first( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) \ returns} \ x{s:string \ C:\Temp\Customers.xml}

If the file \texttt{C:\Temp\Customers.xml} does not exist, and \texttt{http://www.altova.com/index.html} exists:

- \texttt{altova:find-first( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) \ returns} \ x{s:string \ http://www.altova.com/index.html}

If the file \texttt{C:\Temp\Customers.xml} does not exist, and \texttt{http://www.altova.com/index.html} also does not exist:

- \texttt{altova:find-first( ("C:\Temp\ Customers.xml", "http://www.altova.com/ index.html"), (doc-available#1) \ returns} no result

\textbf{Notes about the examples given above}

- The XPath 3.0 function, \texttt{doc-available}, takes a single string argument, which is used as a URI, and returns \texttt{true} if a document node is found at the submitted URI. (The document at the submitted URI must therefore be an XML document.)
- The \texttt{doc-available} function can be used for \texttt{Condition}, the second argument of \texttt{altova:find-first}, because it takes only one argument (arity=1), because it takes an \texttt{item()} as input (a string which is used as a URI), and returns a boolean value.
- Notice that the \texttt{doc-available} function is only referenced, not called. The \#1 suffix that is attached to it indicates a function with an arity of 1. In its entirety \texttt{doc-available\#1} simply means: \textit{Use the doc-available() function that has arity=1, passing to it as its single argument, in turn, each of the items in the first sequence. As a result, each of the two strings will be passed to doc-available(), which uses the string as a URI and tests whether a document node exists at the URI. If}
one does, the `doc-available()` evaluates to `true()` and that string is returned as the result of the `altova:find-first` function. Note about the `doc-available()` function: Relative paths are resolved relative to the the current base URI, which is by default the URI of the XML document from which the function is loaded.

### find-first-combination [altova:]

```xml
altova:find-first-combination((Seq-01 as item()*), (Seq-02 as item()*),
(Condition( Seq-01-Item, Seq-02-Item as xs:boolean)) AS item()* XP3.1 XQ3.1
```

This function takes three arguments:

- The first two arguments, `Seq-01` and `Seq-02`, are sequences of one or more items of any datatype.
- The third argument, `Condition`, is a reference to an XPath function that takes two arguments (has an arity of 2) and returns a boolean.

The items of `Seq-01` and `Seq-02` are passed in ordered pairs (one item from each sequence making up a pair) as the arguments of the function in `Condition`. The pairs are ordered as follows.

**If**  
`Seq-01 = X1, X2, X3 ... Xn`  
`And Seq-02 = Y1, Y2, Y3 ... Yn`  
**Then**  
`(X1 Y1), (X1 Y2), (X1 Y3) ... (X1 Yn), (X2 Y1), (X2 Y2) ... (Xn Yn)`

The first ordered pair that causes the `Condition` function to evaluate to `true()` is returned as the result of `altova:find-first-combination`. Note that: (i) If the `Condition` function iterates through the submitted argument pairs and does not once evaluate to `true()`, then `altova:find-first-combination` returns `No results`; (ii) The result of `altova:find-first-combination` will always be a pair of items (of any datatype) or no item at all.

**Examples**

- `altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a +$b = 32})` returns the sequence of `xs:integers (11, 21)`
- `altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a +$b = 33})` returns the sequence of `xs:integers (11, 22)`
- `altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a +$b = 34})` returns the sequence of `xs:integers (11, 23)`

### find-first-pair [altova:]

```xml
altova:find-first-pair((Seq-01 as item()*), (Seq-02 as item()*),
(Condition( Seq-01-Item, Seq-02-Item as xs:boolean)) AS item()* XP3.1 XQ3.1
```

This function takes three arguments:

- The first two arguments, `Seq-01` and `Seq-02`, are sequences of one or more items of any datatype.
- The third argument, `Condition`, is a reference to an XPath function that takes two arguments (has an arity of 2) and returns a boolean.
The items of Seq-01 and Seq-02 are passed in ordered pairs as the arguments of the function in Condition. The pairs are ordered as follows.

If Seq-01 = X1, X2, X3 ... Xn
And Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X2 Y2), (X3 Y3) ... (Xn Yn)

The first ordered pair that causes the Condition function to evaluate to true() is returned as the result of altova:find-first-pair. Note that: (i) If the Condition function iterates through the submitted argument pairs and does not once evaluate to true(), then altova:find-first-pair returns No results; (ii) The result of altova:find-first-pair will always be a pair of items (of any datatype) or no item at all.

**Examples**

- `altova:find-first-pair(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32})` returns the sequence of xs:integer (11, 21)
- `altova:find-first-pair(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33})` returns No results

Notice from the two examples above that the ordering of the pairs is: (11, 21) (12, 22) (13, 23) ... (20, 30). This is why the second example returns No results (because no ordered pair gives a sum of 33).

**find-first-pair-pos [altova:]**

altova:find-first-pair-pos(Seq-01 as item(), Seq-02 as item(), Condition(Seq-01-Item, Seq-02-Item as xs:boolean) as xs:integer) XP3.1 XQ3.1

This function takes three arguments:

- The first two arguments, Seq-01 and Seq-02, are sequences of one or more items of any datatype.
- The third argument, Condition, is a reference to an XPath function that takes two arguments (has an arity of 2) and returns a boolean.

The items of Seq-01 and Seq-02 are passed in ordered pairs as the arguments of the function in Condition. The pairs are ordered as follows.

If Seq-01 = X1, X2, X3 ... Xn
And Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X2 Y2), (X3 Y3) ... (Xn Yn)

The index position of the first ordered pair that causes the Condition function to evaluate to true() is returned as the result of altova:find-first-pair-pos. Note that if the Condition function iterates through the submitted argument pairs and does not once evaluate to true(), then altova:find-first-pair-pos returns No results.

**Examples**

- `altova:find-first-pair-pos(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32})` returns 1
- `altova:find-first-pair-pos(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33})` returns No results
Notice from the two examples above that the ordering of the pairs is: (11, 21) (12, 22) (13, 23)...(20, 30). In the first example, the first pair causes the Condition function to evaluate to true(), and so its index position in the sequence, 1, is returned. The second example returns No results because no pair gives a sum of 33.

**find-first-pos [altova:]**

```
altova:find-first-pos((Sequence as item())*, (Condition( Sequence-Item as xs:boolean)) as XS xs:integer  XP3.1  XQ3.1
```

This function takes two arguments. The first argument is a sequence of one or more items of any datatype. The second argument, Condition, is a reference to an XPath function that takes one argument (has an arity of 1) and returns a boolean. Each item of sequence is submitted, in turn, to the function referenced in Condition. (Remember: This function takes a single argument.) The first Sequence item that causes the function in Condition to evaluate to true() has its index position in Sequence returned as the result of `altova:find-first-pos`, and the iteration stops.

**Examples**

- `altova:find-first-pos(5 to 10, function($a) {$a mod 2 = 0})` returns `xs:integer 2`
  The Condition argument references the XPath 3.0 inline function, `function()`, which declares an inline function named $a and then defines it. Each item in the Sequence argument of `altova:find-first-pos` is passed, in turn, to $a as its input value. The input value is tested on the condition in the function definition ($a mod 2 = 0). The index position in the sequence of the first input value to satisfy this condition is returned as the result of `altova:find-first-pos` (in this case 2, since 6, the first value (in the sequence) to satisfy the condition, is at index position 2 in the sequence).

- `altova:find-first-pos((2 to 10), (function($a) {$a+3=7}))` returns `xs:integer 3`

**Further examples**

1. If the file `C:\Temp\Customers.xml` exists:
   - `altova:find-first-pos( "C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) ) returns 1`

2. If the file `C:\Temp\Customers.xml` does not exist, and `http://www.altova.com/index.html` exists:
   - `altova:find-first-pos( "C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) ) returns 2`

3. If the file `C:\Temp\Customers.xml` does not exist, and `http://www.altova.com/index.html` also does not exist:
   - `altova:find-first-pos( "C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) ) returns 2`
Notes about the examples given above

- The XPath 3.0 function, `doc-available`, takes a single string argument, which is used as a URI, and returns `true` if a document node is found at the submitted URI. (The document at the submitted URI must therefore be an XML document.)
- The `doc-available` function can be used for `Condition`, the second argument of `altova:find-first-pos`, because it takes only one argument (arity=1), because it takes an `item()` as input (a string which is used as a URI), and returns a boolean value.
- Notice that the `doc-available` function is only referenced, not called. The `#1` suffix that is attached to it indicates a function with an arity of 1. In its entirety `doc-available#1` simply means: Use the `doc-available()` function that has arity=1, passing to it as its single argument, in turn, each of the items in the first sequence. As a result, each of the two strings will be passed to `doc-available()`, which uses the string as a URI and tests whether a document node exists at the URI. If one does, the `doc-available()` function evaluates to `true()` and the index position of that string in the sequence is returned as the result of the `altova:find-first-pos` function. Note about the `doc-available()` function: Relative paths are resolved relative to the the current base URI, which is by default the URI of the XML document from which the function is loaded.

```
altova:for-each-attribute-pair (Seq1 as element()?, Seq2 as element()?, 
Function as function()?) as item()*
```

The first two arguments identify two elements, the attributes of which are used to build attribute pairs, where one attribute of a pair is obtained from the first element and the other attribute is obtained from the second element. Attribute pairs are selected on the basis of having the same name, and the pairs are ordered alphabetically (on their names) into a set. If, for one attribute no corresponding attribute on the other element exists, then the pair is "disjoint", meaning that it consists of one member only. The function item (third argument `Function`) is applied separately to each pair in the sequence of pairs (joint and disjoint), resulting in an output that is a sequence of items.

**Examples**

```
altova:for-each-attribute-pair (/Example/Test-A, /Example/Test-B, 
function($a, $b){$a+$b}) returns ...
```

```
(2, 4, 6) if
<Test-A att1="1" att2="2" att3="3" />
<Test-B att1="1" att2="2" att3="3" />

(2, 4, 6) if
<Test-A att2="2" att1="1" att3="3" />
<Test-B att3="3" att2="2" att1="1" />

(2, 6) if
<Test-A att4="4" att1="1" att3="3" />
<Test-B att3="3" att2="2" att1="1" />
```
Note: The result (2, 6) is obtained by way of the following action: (1+1, ()+2, 3+3, 4+()). If one of the operands is the empty sequence, as in the case of items 2 and 4, then the result of the addition is an empty sequence.

* altova:for-each-attribute-pair(/Example/Test-A, /Example/Test-B, concat#2) returns ...

(11, 22, 33) if
  <Test-A att1="1" att2="2" att3="3" />  
  <Test-B att1="1" att2="2" att3="3" /> 

(11, 2, 33, 4) if
  <Test-A att4="4" att1="1" att3="3" />  
  <Test-B att3="3" att2="2" att1="1" /> 

for-each-combination [altova:]

altova:for-each-combination(FirstSequence as item()*, SecondSequence as item()*, Function($i,$j){$i || $j}) as item()*  XP3.1 XQ3.1

The items of the two sequences in the first two arguments are combined so that each item of the first sequence is combined, in order, once with each item of the second sequence. The function given as the third argument is applied to each combination in the resulting sequence, resulting in an output that is a sequence of items (see example).

Examples

* altova:for-each-combination(('a', 'b', 'c'), ('1', '2', '3'), function($i, $j){$i || $j}) returns ('a1', 'a2', 'a3', 'b1', 'b2', 'b3', 'c1', 'c2', 'c3')

for-each-matching-attribute-pair [altova:]

altova:for-each-matching-attribute-pair(Seq1 as element()?, Seq2 as element()?, Function as function()) as item()* XP3.1 XQ3.1

The first two arguments identify two elements, the attributes of which are used to build attribute pairs, where one attribute of a pair is obtained from the first element and the other attribute is obtained from the second element. Attribute pairs are selected on the basis of having the same name, and the pairs are ordered alphabetically (on their names) into a set. If, for one attribute no corresponding attribute on the other element exists, then no pair is built. The function item (third argument Function) is applied separately to each pair in the sequence of pairs, resulting in an output that is a sequence of items.

Examples

* altova:for-each-matching-attribute-pair(/Example/Test-A, /Example/Test-B, function($a, $b){$a+b}) returns ...

(2, 4, 6) if
  <Test-A att1="1" att2="2" att3="3" />  
  <Test-B att1="1" att2="2" att3="3" /> 

(2, 4, 6) if
substitute-empty [altova:]

`<altova:substitute-empty (FirstSequence as item(), SecondSequence as item())>`

If `FirstSequence` is empty, returns `SecondSequence`. If `FirstSequence` is not empty, returns `FirstSequence`.

Examples

- `altova:substitute-empty ((1,2,3), (4,5,6))` returns `(1,2,3)`
- `altova:substitute-empty ((), (4,5,6))` returns `(4,5,6)`

19.1.2.1.7 XPath/XQuery Functions: String

Altova’s string extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova’s XPath 3.0 and XQuery 3.0 engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the Altova extension functions namespace, `http://www.altova.com/xslt-extensions`, and are indicated in this section with the prefix `altova:`, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT): XP1 XP2 XP3.1
### camel-case [altova:]

**altova:camel-case**(InputString as xs:string) as xs:string  

Returns the input string InputString in CamelCase. The string is analyzed using the regular 
expression `\s` (which is a shortcut for the whitespace character). The first non-whitespace 
character after a whitespace or sequence of consecutive whitespaces is capitalized. The first 
character in the output string is capitalized.

#### Examples

- altova:camel-case("max") returns Max
- altova:camel-case("max max") returns Max Max
- altova:camel-case("file01.xml") returns File01.xml
- altova:camel-case("file01.xml file02.xml") returns File01.xml File02.xml
- altova:camel-case("file01.xml file02.xml") returns File01.xml File02.xml
- altova:camel-case("file01.xml -file02.xml") returns File01.xml - file02.xml

**altova:camel-case**(InputString as xs:string, SplitChars as xs:string, IsRegex as xs:boolean) as xs:string  

Converts the input string InputString to camel case by using SplitChars to determine the 
character/s that trigger the next capitalization. SplitChars is used as a regular expression 
when IsRegex = true(), or as plain characters when IsRegex = false(). The first 
character in the output string is capitalized.

#### Examples

- altova:camel-case("setname getname", "set|get", true()) returns setName getName
- altova:camel-case("altova\documents\testcases", ",", false()) returns Altova\Documents\Testcases

### char [altova:]

**altova:char**(Position as xs:integer) as xs:string  

Returns a string containing the character at the position specified by the Position 
argument, in the string obtained by converting the value of the context item to xs:string. 
The result string will be empty if no character exists at the index submitted by the Position 
argument.

#### Examples

If the context item is 1234ABCD:

- altova:char(2) returns 2
- altova:char(5) returns A
- altova:char(9) returns the empty string.
- altova:char(-2) returns the empty string.
<p>altova:char(InputString as xs:string, Position as xs:integer) as xs:string

Returns a string containing the character at the position specified by the Position argument, in the string submitted as the InputString argument. The result string will be empty if no character exists at the index submitted by the Position argument.

Examples
- altova:char("2014-01-15", 5) returns -
- altova:char("USA", 1) returns U
- altova:char("USA", 10) returns the empty string.
- altova:char("USA", -2) returns the empty string.</p>

<< create-hash-from-string[altova:] >>

altova:create-hash-from-string(InputString as xs:string) as xs:string
altova:create-hash-from-string(InputString as xs:string, HashAlgo as xs:string) as xs:string

Generates a hash string from InputString by using the hashing algorithm specified by the HashAlgo argument. The following hashing algorithms may be specified (in upper or lower case): MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512. If the second argument is not specified (see the first signature above), then the SHA-256 hashing algorithm is used.

Examples
- altova:create-hash-from-string('abc') returns a hash string generated by using the SHA-256 hashing algorithm.
- altova:create-hash-from-string('abc', 'md5') returns a hash string generated by using the MD5 hashing algorithm.
- altova:create-hash-from-string('abc', 'MD5') returns a hash string generated by using the MD5 hashing algorithm.

<< first-chars [altova:] >>

altova:first-chars(X-Number as xs:integer) as xs:string

Returns a string containing the first X-Number of characters of the string obtained by converting the value of the context item to xs:string.

Examples
- If the context item is 1234ABCD:
  - altova:first-chars(2) returns 12
  - altova:first-chars(5) returns 1234A
  - altova:first-chars(9) returns 1234ABCD

altova:first-chars(InputString as xs:string, X-Number as xs:integer) as xs:string

Returns a string containing the first X-Number of characters of the string submitted as the InputString argument.

Examples
- altova:first-chars("2014-01-15", 5) returns 2014-
• `altova:first-chars("USA", 1)` returns U

---

**format-string [altova:]**

```xml
altova:format-string(InputString as xs:string, FormatSequence as item()*) as xs:string
```

The input string (first argument) contains positional parameters (%1, %2, etc). Each parameter is replaced by the string item that is located at the corresponding position in the format sequence (submitted as the second argument). So the first item in the format sequence replaces the positional parameter %1, the second item replaces %2, and so on. The function returns this formatted string that contains the replacements. If no string exists for a positional parameter, then the positional parameter itself is returned. This happens when the index of a positional parameter is greater than the number of items in the format sequence.

**Examples**

- `altova:format-string('Hello %1, %2, %3', ('Jane','John','Joe'))` returns "Hello Jane, John, Joe"
- `altova:format-string('Hello %1, %2, %3', ('Jane','John','Joe', 'Tom'))` returns "Hello Jane, John, Joe"
- `altova:format-string('Hello %1, %2, %4', ('Jane','John','Joe', 'Tom'))` returns "Hello Jane, John, Tom"
- `altova:format-string('Hello %1, %2, %4', ('Jane','John','Joe'))` returns "Hello Jane, John, %4"

---

**last-chars [altova:]**

```xml
altova:last-chars(X-Number as xs:integer) as xs:string
```

Returns a string containing the last X-Number of characters of the string obtained by converting the value of the context item to xs:string.

**Examples**

If the context item is 1234ABCD:

- `altova:last-chars(2)` returns CD
- `altova:last-chars(5)` returns 4ABCD
- `altova:last-chars(9)` returns 1234ABCD

---

```xml
altova:last-chars(InputString as xs:string, X-Number as xs:integer) as xs:string
```

Returns a string containing the last X-Number of characters of the string submitted as the InputString argument.

**Examples**

- `altova:last-chars("2014-01-15", 5)` returns 01-15
- `altova:last-chars("USA", 10)` returns USA

---

**pad-string-left [altova:]**

```xml
altova:pad-string-left(StringToPad as xs:string, StringLength as xs:integer, PadCharacter as xs:string) as xs:string
```
The PadCharacter argument is a single character. It is padded to the left of the string to increase the number of characters in StringToPad so that this number equals the integer value of the StringLength argument. The StringLength argument can have any integer value (positive or negative), but padding will occur only if the value of StringLength is greater than the number of characters in StringToPad. If StringToPad has more characters than the value of StringLength, then StringToPad is left unchanged.

**Examples**

- `altova:pad-string-left('AP', 1, 'Z')` returns 'AP'
- `altova:pad-string-left('AP', 2, 'Z')` returns 'AP'
- `altova:pad-string-left('AP', 3, 'Z')` returns 'ZAP'
- `altova:pad-string-left('AP', 4, 'Z')` returns 'ZZAP'
- `altova:pad-string-left('AP', -3, 'Z')` returns 'AP'
- `altova:pad-string-left('AP', 3, 'YZ')` returns a pad-character-too-long error

---

**pad-string-right [altova:]**

```xml
<xs:schema

altova:pad-string-right(StringToPad as xs:string, StringLength as xs:integer, PadCharacter as xs:string) as xs:string
```

The PadCharacter argument is a single character. It is padded to the right of the string to increase the number of characters in StringToPad so that this number equals the integer value of the StringLength argument. The StringLength argument can have any integer value (positive or negative), but padding will occur only if the value of StringLength is greater than the number of characters in StringToPad. If StringToPad has more characters than the value of StringLength, then StringToPad is left unchanged.

**Examples**

- `altova:pad-string-right('AP', 1, 'Z')` returns 'AP'
- `altova:pad-string-right('AP', 2, 'Z')` returns 'AP'
- `altova:pad-string-right('AP', 3, 'Z')` returns 'APZ'
- `altova:pad-string-right('AP', 4, 'Z')` returns 'APZZ'
- `altova:pad-string-right('AP', -3, 'Z')` returns 'AP'
- `altova:pad-string-right('AP', 3, 'YZ')` returns a pad-character-too-long error

---

**repeat-string [altova:]**

```xml
<xs:schema

altova:repeat-string(InputString as xs:string, Repeats as xs:integer) as xs:string
```

Generates a string that is composed of the first InputString argument repeated Repeats number of times.

**Examples**

- `altova:repeat-string("Altova ", 3)` returns "Altova #Altova #Altova #"

---

**substring-after-last [altova:]**

```xml
<xs:schema

altova:substring-after-last(MainString as xs:string, CheckString as xs:string) as xs:string
```

If CheckString is found in MainString, then the substring that occurs after CheckString in MainString is returned. If CheckString is not found in MainString, then the empty string is

---

© 2019 Altova Gmbh
returned. If CheckString is an empty string, then MainString is returned in its entirety. If there is more than one occurrence of CheckString in MainString, then the substring after the last occurrence of CheckString is returned.

**Examples**

- altova:substring-after-last('ABCDEFGH', 'B') returns 'CDEFGH'
- altova:substring-after-last('ABCDEFGH', 'BC') returns 'DEFGH'
- altova:substring-after-last('ABCDEFGH', 'BD') returns ''
- altova:substring-after-last('ABCDEFGH', 'Z') returns ''
- altova:substring-after-last('ABCDEFGH', '') returns 'ABCDEFGH'
- altova:substring-after-last('ABC-ABCD', 'B') returns 'CD'
- altova:substring-after-last('ABC-ABCD-ABCD', 'BCD') returns ''

**substring-before-last [altova:]**

\[
\text{altova:substring-before-last} \left( \text{MainString as } \text{xs:string}, \text{CheckString as } \text{xs:string} \right) \text{ as } \text{xs:string}
\]

If CheckString is found in MainString, then the substring that occurs before CheckString in MainString is returned. If CheckString is not found in MainString, or if CheckString is an empty string, then the empty string is returned. If there is more than one occurrence of CheckString in MainString, then the substring before the last occurrence of CheckString is returned.

**Examples**

- altova:substring-before-last('ABCDEFGH', 'B') returns 'A'
- altova:substring-before-last('ABCDEFGH', 'BC') returns 'A'
- altova:substring-before-last('ABCDEFGH', 'BD') returns ''
- altova:substring-before-last('ABCDEFGH', 'Z') returns ''
- altova:substring-before-last('ABCDEFGH', '') returns ''
- altova:substring-before-last('ABC-ABCD', 'B') returns 'ABC-ABCD-

**substring-pos [altova:]**

\[
\text{altova:substring-pos} \left( \text{StringToCheck as } \text{xs:string}, \text{StringToFind as } \text{xs:string} \right) \text{ as } \text{xs:integer}
\]

Returns the character position of the first occurrence of StringToFind in the string StringToCheck. The character position is returned as an integer. The first character of StringToCheck has the position 1. If StringToFind does not occur within StringToCheck, the integer 0 is returned. To check for the second or a later occurrence of StringToCheck, use the next signature of this function.

**Examples**

- altova:substring-pos('Altova', 'to') returns 3
- altova:substring-pos('Altova', 'to') returns 3
- altova:substring-pos('Altova', 'tv') returns 0
- altova:substring-pos('AltovaAltova', 'to') returns 3

\[
\text{altova:substring-pos} \left( \text{StringToCheck as } \text{xs:string}, \text{StringToFind as } \text{xs:string}, \text{Integer as } \text{xs:integer} \right) \text{ as } \text{xs:integer}
\]
Returns the character position of StringToFind in the string, StringToCheck. The search for StringToFind starts from the character position given by the Integer argument; the character substring before this position is not searched. The returned integer, however, is the position of the found string within the entire string, StringToCheck. This signature is useful for finding the second or a later position of a string that occurs multiple times with the StringToCheck. If StringToFind does not occur within StringToCheck, the integer 0 is returned.

**Examples**

```
• altova:substring-pos('Altova', 'to', 1) returns 3
• altova:substring-pos('Altova', 'to', 3) returns 3
• altova:substring-pos('Altova', 'to', 4) returns 0
• altova:substring-pos('Altova-Altova', 'to', 0) returns 3
• altova:substring-pos('Altova-Altova', 'to', 4) returns 10
```

**trim-string [altova:]**

```
altova:trim-string([InputString] as xs:string) as xs:string
```

This function takes an xs:string argument, removes any leading and trailing whitespace, and returns a "trimmed" xs:string.

**Examples**

```
• altova:trim-string(" Hello World ") returns "Hello World"
• altova:trim-string("Hello World ") returns "Hello World"
• altova:trim-string(" Hello World") returns "Hello World"
• altova:trim-string("Hello World") returns "Hello World"
• altova:trim-string("Hello World") returns "Hello World"
```

**trim-string-left [altova:]**

```
altova:trim-string-left([InputString] as xs:string) as xs:string
```

This function takes an xs:string argument, removes any leading whitespace, and returns a left-trimmed xs:string.

**Examples**

```
• altova:trim-string-left(" Hello World ") returns "Hello World "
• altova:trim-string-left("Hello World ") returns "Hello World "
• altova:trim-string-left(" Hello World") returns "Hello World"
• altova:trim-string-left("Hello World") returns "Hello World"
• altova:trim-string-left("Hello World") returns "Hello World"
```

**trim-string-right [altova:]**

```
altova:trim-string-right([InputString] as xs:string) as xs:string
```

This function takes an xs:string argument, removes any trailing whitespace, and returns a right-trimmed xs:string.

**Examples**

```
• altova:trim-string-right(" Hello World ") returns " Hello World"
• altova:trim-string-right("Hello World ") returns "Hello World"
```
19.1.2.1.8 XPath/XQuery Functions: Miscellaneous

The following general purpose XPath/XQuery extension functions are supported in the current version of MapForce and can be used in (i) XPath expressions in an XSLT context, or (ii) XQuery expressions in an XQuery document.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the Altova extension functions namespace, http://www.altova.com/xslt-extensions, and are indicated in this section with the prefix altova:, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

| XPath functions (used in XPath expressions in XSLT): | XP1 XP2 XP3.1 |
| XSLT functions (used in XPath expressions in XSLT): | XSLT1 XSLT2 XSLT3 |
| XQuery functions (used in XQuery expressions in XQuery): | XQ1 XQ3.1 |

- get-temp-folder [altova:]

  `altova:get-temp-folder()` as xs:string  XP2 XQ1 XP3.1 XQ3.1

  This function takes no argument. It returns the path to the temporary folder of the current user.

  **Examples**

  - `altova:get-temp-folder()` would return, on a Windows machine, something like C:\Users\<UserName>\AppData\Local\Temp\ as an xs:string.

- generate-guid [altova:]

  `altova:generate-guid()` as xs:string  XP2 XQ1 XP3.1 XQ3.1

  Generates a unique string GUID string.

  **Examples**

  - `altova:generate-guid()` returns (for example) 85F971DA-17F3-4E4E-994E-99137873ACCD
### high-res-timer [altova:]

```
altova:high-res-timer() as xs:double  XP3.1 XQ3.1
```

Returns a system high-resolution timer value in seconds. A high-resolution timer, when present on a system, enables high precision time measurements when these are required (for example, in animations and for determining precise code-execution time). This function provides the resolution of the system's high-res timer.

**Examples**

- `altova:high-res-timer()` returns something like '1.16766146154566E6'

### parse-html [altova:]

```
altova:parse-html(HTMLText as xs:string) as node()  XP3.1 XQ3.1
```

The HTMLText argument is a string that contains the text of an HTML document. The function creates an HTML tree from the string. The submitted string may or may not contain the HTML element. In either case, the root element of the tree is an element named `HTML`. It is best to make sure that the HTML code in the submitted string is valid HTML.

**Examples**

- `altova:parse-html("<html><head/><body><h1>Header</h1></body></html>")` creates an HTML tree from the submitted string

### sleep[altova:]

```
altova:sleep(Millisecs as xs:integer) as empty-sequence()  XP2 XQ1 XP3.1 XQ3.1
```

Suspends execution of the current operation for the number of milliseconds given by the Milliseconds argument.

**Examples**

- `altova:sleep(1000)` suspends execution of the current operation for 1000 milliseconds.

[ Top ]

### 19.1.2.1.9 Chart Functions

The chart functions listed below enable you to create, generate, and save charts as images. They are supported in the current version of your Altova product in the manner described below. However, note that in future versions of your product, support for one or more of these functions might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

The chart functions are **XPath functions** (not XSLT functions), and organized into two groups:

- Functions for generating and saving charts
- Functions for creating charts
Note: Chart functions are supported only in Altova’s Server products and the Enterprise Editions of Altova products.

Note: Supported image formats for charts in server editions are jpg, png, and bmp. The best option is png because it is lossless and compressed. In Enterprise editions, the supported formats are jpg, png, bmp, and gif.

Functions for generating and saving charts

These functions take the chart object (obtained with the chart creation functions) and either generate an image or save an image to file.

```
altova:generate-chart-image ($chart, $width, $height, $encoding) as atomic
```

where

- `$chart` is the chart extension item obtained with the `altova:create-chart` function
- `$width` and `$height` must be specified with a length unit
- `$encoding` may be `binarytobase64` or `binarytobase16`

The function returns the chart image in the specified encoding.

```
altova:generate-chart-image ($chart, $width, $height, $encoding, $imagetype) as atomic
```

where

- `$chart` is the chart extension item obtained with the `altova:create-chart` function
- `$width` and `$height` must be specified with a length unit
- `$encoding` may be `base64Binary` or `hexBinary`
- `$imagetype` may be one of the following image formats: png, gif, bmp, jpg, jpeg. Note that gif is not supported on server products. Also see note at top of page.

The function returns the chart image in the specified encoding and image format.

```
altova:save-chart-image ($chart, $filename, $width, $height) as empty() (Windows only)
```

where

- `$chart` is the chart extension item obtained with the `altova:create-chart` function
- `$filename` is the path to and name of the file to which the chart image is to be saved
- `$width` and `$height` must be specified with a length unit

The function saves the chart image to the file specified in `$filename`.

```
altova:save-chart-image ($chart, $filename, $width, $height, $imagetype) as empty() (Windows only)
```

where
• $chart is the chart extension item obtained with the altova:create-chart function
• $filename is the path to and name of the file to which the chart image is to be saved
• $width and $height must be specified with a length unit
• $imagetype may be one of the following image formats: png, gif, bmp, jpg, jpeg. Note that gif is not supported on server products. Also see note at top of page.

The function saves the chart image to the file specified in $filename in the image format specified.

Functions for creating charts
The following functions are used to create charts.

altova:create-chart($chart-config, $chart-data-series*) as chart extension item

where

• $chart-config is the chart-config extension item obtained with the altova:create-chart-config function or or via the altova:create-chart-config-from-xml function
• $chart-data-series is the chart-data-series extension item obtained with the altova:create-chart-data-series function or altova:create-chart-data-series-from-rows function

The function returns a chart extension item, which is created from the data supplied via the arguments.

altova:create-chart-config($type-name, $title) as chart-config extension item

where

• $type-name specifies the type of chart to be created: Pie, Pie3d, BarChart, BarChart3d, BarChart3dGrouped, LineChart, ValueLineChart, RoundGauge, BarGauge
• $title is the name of the chart

The function returns a chart-config extension item containing the configuration information of the chart.

altova:create-chart-config-from-xml($xml-struct) as chart-config extension item

where

• $xml-struct is the XML structure containing the configuration information of the chart

The function returns a chart-config extension item containing the configuration information of the chart. This information is supplied in an XML data fragment.

altova:create-chart-data-series($series-name?, $x-values*, $y-values*) as chart-data-series extension item
where

- \$series-name specifies the name of the series
- \$x-values gives the list of X-Axis values
- \$y-values gives the list of Y-Axis values

The function returns a chart-data-series extension item containing the data for building the chart: that is, the names of the series and the Axes data.

\[ \text{altova:create-chart-data-row}(x, y_1, y_2, y_3, \ldots) \text{ as chart-data-x-Ny-row extension item} \]

where

- \( x \) is the value of the X-Axis column of the chart data row
- \( y_N \) are the values of the Y-Axis columns

The function returns a chart-data-x-Ny-row extension item, which contains the data for the X-Axis column and Y-Axis columns of a single series.

\[ \text{altova:create-chart-data-series-from-rows}(%03D{series-names as xs:string*, }%03D{row*}) \text{ as chart-data-series extension item} \]

where

- \$series-name is the name of the series to be created
- \$row is the chart-data-x-Ny-row extension item that is to be created as a series

The function returns a chart-data-series extension item, which contains the data for the X-Axis and Y-Axes of the series.

\[ \text{altova:create-chart-layer}(\text{chart-config, chart-data-series*}) \text{ as chart-layer extension item} \]

where

- \$chart-config is the chart-config extension item obtained with the \text{altova:create-chart-config} function or or via the \text{altova:create-chart-config-from-xml} function
- \$chart-data-series is the chart-data-series extension item obtained with the \text{altova:create-chart-data-series} function or \text{altova:create-chart-data-series-from-rows} function

The function returns a chart-layer extension item, which contains chart-layer data.

\[ \text{altova:create-multi-layer-chart}(\text{chart-config, chart-data-series*, chart-layer*}) \]
chart-config function or or via the altova:create-chart-config-from-xml function

- $chart-data-series is the chart-data-series extension item obtained with the
  altova:create-chart-data-series function or altova:create-chart-data-series-from-rows function

- $chart-layer is the chart-layer extension item obtained with the altova:create-chart-layer function

The function returns a multi-layer-chart item.

\[
\text{altova:create-multi-layer-chart}($chart-config, $chart-data-series*, $chart-layer*, \text{xs:boolean } \text{mergecategoryvalues})
\]

where

- $chart-config is the chart-config extension item obtained with the altova:create-chart-config function or or via the altova:create-chart-config-from-xml function

- $chart-data-series is the chart-data-series extension item obtained with the
  altova:create-chart-data-series function or altova:create-chart-data-series-from-rows function

- $chart-layer is the chart-layer extension item obtained with the altova:create-chart-layer function

The function returns a multi-layer-chart item.

19.1.2.1.9.1 Chart Data XML Structure

Given below is the XML structure of chart data, how it might appear for the Altova extension functions for charts. This affects the appearance of the specific chart. Not all elements are used for all chart kinds, e.g. the <Pie> element is ignored for bar charts.

Note: Chart functions are supported only in the Enterprise and Server Editions of Altova products.

<chart-config>
  <General
    SettingsVersion="1"  must be provided
    ChartKind="BarChart"  Pie, Pie3d, BarChart, StackedBarChart, BarChart3d,
    BarChart3dGrouped, LineChart, ValueLineChart, AreaChart, StackedAreaChart, RoundGauge,
    BarGauge, CandleStick
    BKColor="#ffffff"  Color
    BKColorGradientEnd="#ffffff"  Color. In case of a gradient, BKColor and
    BKColorGradientEnd define the gradient's colors
    BKMode="#ffffff"  Solid, HorzGradient, VertGradient
    BKFile="Path+Filename"  String. If file exists, its content is drawn over the
    background.
    BKFileMode="#ffffff"  Stretch, ZoomToFit, Center, Tile
    ShowBorder="1"  Bool
    PlotBorderColor="#000000"  Color
    PlotBKColor="#ffffff"  Color

© 2019 Altova GmbH
Altova MapForce 2019 Enterprise Edition
<CandleStick>
    FillHighClose="0"  Boolean. If 0, the body is left empty. If 1, FillColorHighClose is used for the candle body
    FillColorHighClose="#ffffff"  Color. For the candle body when close > open
    FillHighOpenWithSeriesColor="1"  Boolean. If true, the series color is used to fill the candlebody when open > close
    FillColorHighOpen="#000000"  Color. For the candle body when open > close and FillHighOpenWithSeriesColor is false
</CandleStick>

<Colors>
    User-defined color scheme: By default this element is empty except for the style and has no Color attributes
    UseSubsequentColors="1"  Boolean. If 0, then color in overlay is used. If 1, then subsequent colors from previous chart layer is used
    Style="User"  Possible values are: "Default", "Grayscale", "Colorful", "Pastel", "User"
    Colors="#52aca0"  Color: only added for user defined color set
    Colors1="#d3c15d"  Color: only added for user defined color set
    Colors2="#8971d8"  Color: only added for user defined color set
    ...  ColorsN=""  Up to ten colors are allowed in a set: from Colors to Colors9
</Colors>

<Pie>
    ShowLabels="1"  Boolean
    OutlineColor="#404040"  Color
    ShowOutline="1"  Boolean
    StartAngle="0."  Double
    Clockwise="1"  Boolean
    Draw2dHighlights="1"  Boolean
    Transparency="0"  Int (0 to 255: 0 is opaque, 255 is fully transparent)
    DropShadowColor="#c0c0c0"  Color
    DropShadowSize="5.%"  PercentOrPixel
    PieHeight="10.%"  PercentOrPixel. Pixel values might be different in the result because of 3d tilting
    Tilt="40.0"  Double (10 to 90: The 3d tilt in degrees of a 3d pie)
    ShowDropShadow="1"  Boolean
    ChartToLabelMargin="10.%"  PercentOrPixel
    AddValueToLabel="0"  Boolean
    AddPercentToLabel="0"  Boolean
    AddPercentToLabels.DecimalDigits="0"  UINT (0 – 2)
</Pie>

<LabelFont>
    Color="#000000"
    Name="Arial"
    Bold="0"
    Italic="0"
    Underline="0"
    MinFontHeight="10.pt"
    Size="4.5%"
</LabelFont>

</XY>
<XAxis
  AutoRange="1"  Bool
  AutoRangeIncludesZero="1"  Bool
  RangeFrom="0."  Double: manual range
  RangeTill="1."  Double: manual range
  LabelToAxisMargin="3.3%"  PercentOrPixel
  AxisLabel=""  String
  AxisColor="#000000"  Color
  AxisGridColor="#666666"  Color
  ShowGrid="1"  Bool
  UseAutoTick="1"  Bool
  ManualTickInterval="1."  Double
  AxisToChartMargin="0.0px"  PercentOrPixel
  TickSize="3.0px"  PercentOrPixel
  ShowTicks="1"  Bool
  ShowValues="1"  Bool
  AxisPosition="LeftOrBottom"  Enums: "LeftOrBottom",
  "RightOrTop", "AtValue"

  AxisPositionAtValue = "0"  Double
>
  <ValueFont
    Color="#000000"
    Name="Tahoma"
    Bold="0"
    Italic="0"
    Underline="0"
    MinFontHeight="10.0pt"
    Size="3.3%" />
</XAxis>

<YAxis
  Axis (same as for XAxis)
  AutoRange="1"
  AutoRangeIncludesZero="1"
  RangeFrom="0.0"
  RangeTill="1.0"
  LabelToAxisMargin="3.3%"
  AxisLabel=""
  AxisColor="#000000"
  AxisGridColor="#666666"
  ShowGrid="1"
  UseAutoTick="1"
  ManualTickInterval="1.0"
  AxisToChartMargin="0.0px"
  TickSize="3.0px"
  ShowTicks="1"  Bool
  ShowValues="1"  Bool
  AxisPosition="LeftOrBottom"  Enums: "LeftOrBottom",
  "RightOrTop", "AtValue"

  AxisPositionAtValue = "0"  Double
>
  <ValueFont
    Color="#000000"
    Name="Tahoma"
    Bold="0"
    Italic="0"
    Underline="0"
MinFontHeight="10.pt"
Size="3.%"/>

</YAxis>

</XY>

<XY3d
AxisAutoSize="1"  // Boolean: If false, XSize and YSize define the aspect ratio of x and y axis. If true, aspect ratio is equal to chart window
XSize="100.%"  // PercentOrPixel. Pixel values might be different in the result because of 3d tilting and zooming to fit chart
YSize="100.%"  // PercentOrPixel. Pixel values might be different in the result because of 3d tilting and zooming to fit chart
SeriesMargin="30.%"  // PercentOrPixel. Pixel values might be different in the result because of 3d tilting and zooming to fit chart
Tilt="20."  // Double. -90 to +90 degrees
Rot="20."  // Double. -359 to +359 degrees
FoV="50."  // Double. Field of view: 1-120 degree
>
<ZAxis
AutoRange="1"
AutoRangeIncludesZero="1"
RangeFrom="0."
RangeTill="1."
LabelToAxisMargin="3.%"
AxisLabel=""
AxisColor="#000000"
AxisGridColor="#e6e6e6"
ShowGrid="1"
UseAutoTick="1"
ManualTickInterval="1."
AxisToChartMargin="0.px"
TickSize="3.px" >
<ValueFont
Color="#000000"
Name="Tahoma"
Bold="0"
Italic="0"
Underline="0"
MinFontHeight="10.pt"
Size="3.%"/>
</ZAxis>

</XY3d>

<Gauge
MinVal="0."  // Double
MaxVal="100."  // Double
MinAngle="225"  // UINT: -359-359
SweepAngle="270"  // UINT: 1-359
BorderToTick="1.%"  // PercentOrPixel
MajorTickWidth="3.px"  // PercentOrPixel
MajorTickLength="4.%"  // PercentOrPixel
MinorTickWidth="1.px"  // PercentOrPixel
MinorTickLength="3.%"  // PercentOrPixel
BorderColor="#a0a0a0"  // Color
FillColor="#303535"  // Color
Example: Chart Functions

The example XSLT document below shows how Altova extension functions for charts can be used. Given further below are an XML document and a screenshot of the output image generated when the XML document is processed with the XSLT document using the XSLT 2.0 Engine.

Note: Chart functions are supported only in the Enterprise and Server Editions of Altova products.
Note: For more information about how chart data tables are created, see the documentation of Altova’s XMLSpy and StyleVision products.

XSLT document
This XSLT document (listing below) uses Altova chart extension functions to generate a pie chart. It can be used to process the XML document listed further below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:altovaext="http://www.altova.com/xslt-extensions"
    exclude-result-prefixes="#all">
    <xsl:output version="4.0" method="html" indent="yes" encoding="UTF-8"/>
    <xsl:template match="/">
        <html>
            <head>
                <title>
                    <xsl:text>HTML Page with Embedded Chart</xsl:text>
                </title>
            </head>
            <body>
                <xsl:for-each select="/Data/Region[1]">
                    <xsl:variable name="extChartConfig" as="item()*">
                        <xsl:variable name="ext-chart-settings" as="item()*">
                            <chart-config>
                                <General
                                    SettingsVersion="#1"
                                    ChartKind="#Pie3d"
                                    BKColor="#ffffff"
                                    ShowBorder="#1"
                                    PlotBorderColor="#000000"
                                    PlotBKColor="#ffffff"
                                    Title="#@id#"
                                    ShowLegend="#1"
                                    OutsideMargin="#3.2%"
                                    TitleToPlotMargin="#3.3%"
                                    LegendToPlotMargin="#6.3%"
                                >
                                    <TitleFont
                                        Color="#023d7d"
                                        Name="#Tahoma"
                                        Bold="#1"
                                        Italic="#0"
                                        Underline="#0"
                                        MinFontHeight="#10.pt"
                                        Size="#8.3%"
                                    />
                                </General>
                            </chart-config>
                        </xsl:variable>
                    </xsl:sequence select="altovaext:create-chart-config-from-
```
XML document

This XML document can be processed with the XSLT document above. Data in the XML document is used to generate the pie chart shown in the screenshot below.
The pie chart shown below is generated when the XML document listed above is processed with the XSLT document.

19.1.2.2 Miscellaneous Extension Functions

There are several ready-made functions in programming languages such as Java and C# that are not available as XQuery/XPath functions or as XSLT functions. A good example would be the math functions available in Java, such as sin() and cos(). If these functions were available to the designers of XSLT stylesheets and XQuery queries, it would increase the application area of stylesheets and queries and greatly simplify the tasks of stylesheet creators. The XSLT and
XQuery engines used in a number of Altova products support the use of extension functions in Java and .NET, as well as MSXSL scripts for XSLT. This section describes how to use extension functions and MSXSL scripts in your XSLT stylesheets and XQuery documents. The available extension functions are organized into the following sections:

- Java Extension Functions
- .NET Extension Functions
- MSXSL Scripts for XSLT

The two main issues considered in the descriptions are: (i) how functions in the respective libraries are called; and (ii) what rules are followed for converting arguments in a function call to the required input format of the function, and what rules are followed for the return conversion (function result to XSLT/XQuery data object).

Requirements

For extension functions support, a Java Runtime Environment (for access to Java functions) and .NET Framework 2.0 (minimum, for access to .NET functions) must be installed on the machine running the XSLT transformation or XQuery execution, or must be accessible for the transformations.

19.1.2.2.1 Java Extension Functions

A Java extension function can be used within an XPath or XQuery expression to invoke a Java constructor or call a Java method (static or instance).

A field in a Java class is considered to be a method without any argument. A field can be static or instance. How to access fields is described in the respective sub-sections, static and instance.

This section is organized into the following sub-sections:

- Java: Constructors
- Java: Static Methods and Static Fields
- Java: Instance Methods and Instance Fields
- Datatypes: XPath/XQuery to Java
- Datatypes: Java to XPath/XQuery

Note the following

- If you are using an Altova desktop product, the Altova application attempts to detect the path to the Java virtual machine automatically, by reading (in this order): (i) the Windows registry, and (ii) the JAVA_HOME environment variable. You can also add a custom path in the Options dialog of the application; this entry will take priority over any other Java VM path detected automatically.
- If you are running an Altova server product on a Windows machine, the path to the Java virtual machine will be read first from the Windows registry; if this is not successful the JAVA_HOME environment variable will be used.
- If you are running an Altova server product on a Linux or macOS machine, then make sure that the path to the Java virtual machine is stored in the JAVA_HOME environment variable. The path must point to the jvm.dll file in the \bin\server or \bin\client directory.
Form of the extension function

The extension function in the XPath/XQuery expression must have the form prefix:fname().

- The prefix: part identifies the extension function as a Java function. It does so by associating the extension function with an in-scope namespace declaration, the URI of which must begin with java: (see below for examples). The namespace declaration should identify a Java class, for example: xmlns:myns="java:java.lang.Math". However, it could also simply be: xmlns:myns="java" (without a colon), with the identification of the Java class being left to the fname() part of the extension function.

- The fname() part identifies the Java method being called, and supplies the arguments for the method (see below for examples). However, if the namespace URI identified by the prefix: part does not identify a Java class (see preceding point), then the Java class should be identified in the fname() part, before the class and separated from the class by a period (see the second XSLT example below).

Note: The class being called must be on the classpath of the machine.

XSLT example

Here are two examples of how a static method can be called. In the first example, the class name (java.lang.Math) is included in the namespace URI and, therefore, must not be in the fname() part. In the second example, the prefix: part supplies the prefix java: while the fname() part identifies the class as well as the method.

```
             select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jmath="java"
              select="jmath:java.lang.Math.cos(3.14)" />
```

The method named in the extension function (cos() in the example above) must match the name of a public static method in the named Java class (java.lang.Math in the example above).

XQuery example

Here is an XQuery example similar to the XSLT example above:

```
<cosine xmlns:jMath="java:java.lang.Math">
  {jMath:cos(3.14)}
</cosine>
```

User-defined Java classes

If you have created your own Java classes, methods in these classes are called differently according to: (i) whether the classes are accessed via a JAR file or a class file, and (ii) whether these files (JAR or class) are located in the current directory (the same directory as the XSLT or XQuery document) or not. How to locate these files is described in the sections User-Defined Class Files and User-Defined Jar Files. Note that paths to class files not in the current directory and to all JAR files must be specified.
If access is via a class file, then there are two possibilities:

- The class file is in a package. The XSLT or XQuery file is in the same folder as the Java package. (See example below.)
- The class file is not packaged. The XSLT or XQuery file is in the same folder as the class file. (See example below.)
- The class file is in a package. The XSLT or XQuery file is at some random location. (See example below.)
- The class file is not packaged. The XSLT or XQuery file is at some random location. (See example below.)

Consider the case where the class file is not packaged and is in the same folder as the XSLT or XQuery document. In this case, since all classes in the folder are found, the file location does not need to be specified. The syntax to identify a class is:

```
java:classname
```

where

```
java: indicates that a user-defined Java function is being called; (Java classes in the current directory will be loaded by default)
classname is the name of the required method's class
```

The class is identified in a namespace URI, and the namespace is used to prefix a method call.

### Class file packaged, XSLT/XQuery file in same folder as Java package

The example below calls the `getVehicleType()` method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is also in the folder `JavaProject`.

```xml
<xsl:stylesheet version="2.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:fn="http://www.w3.org/2005/xpath-functions"
 xmlns:car="java:com.altova.extfunc.Car" >
 <xsl:output exclude-result-prefixes="fn car xsl fo xs"/>
</xsl:stylesheet>
```

### Class file referenced, XSLT/XQuery file in same folder as class file

The example below calls the `getVehicleType()` method of the `Car` class. Let us say that: (i) the
Car class file is in the following folder: JavaProject/com/altova/extfunc, and (ii) that this folder is the current folder in the example below. The XSLT file is also in the folder JavaProject/com/altova/extfunc.

```xml
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:fn="http://www.w3.org/2005/xpath-functions"
    xmlns:car="java:Car">
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
    <a>
        <xsl:value-of select="car:getVehicleType()"/>
    </a>
</xsl:template>
</xsl:stylesheet>
```

Class file packaged, XSLT/XQuery file at any location

The example below calls the `getCarColor()` method of the Car class of the com.altova.extfunc package. The com.altova.extfunc package is in the folder JavaProject. The XSLT file is at any location. In this case, the location of the package must be specified within the URI as a query string. The syntax is:

```
java:classname[?path=uri-of-package]
```

where

java: indicates that a user-defined Java function is being called
uri-of-package is the URI of the Java package
classname is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call. The example below shows how to access a class file that is located in another directory than the current directory.

```xml
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:fn="http://www.w3.org/2005/xpath-functions"

<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
    <xsl:variable name="myCar" select="car:new('red')"/>
    <a><xsl:value-of select="car:getCarColor($myCar)"/></a>
</xsl:template>
</xsl:stylesheet>
```
Class file referenced, XSLT/XQuery file at any location

The example below calls the `getCarColor()` method of the `Car` class. Let us say that the `Car` class file is in the folder `C:/JavaProject/com/altova/extfunc`, and the XSLT file is at any location. The location of the class file must then be specified within the namespace URI as a query string. The syntax is:

\[
\text{java:classname[?path=<uri-of-classfile>]}
\]

where

- `java:` indicates that a user-defined Java function is being called
- `uri-of-classfile` is the URI of the folder containing the class file
- `classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call. The example below shows how to access a class file that is located in another directory than the current directory.

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:car="java:Car?path=file:///C:/JavaProject/com/altova/
     extfunc/"
>
  <xsl:output exclude-result-prefixes="fn car xsl xs"/>

  <xsl:template match="/"
    >
    <xsl:variable name="myCar" select="car:new('red')" />
    <a><xsl:value-of select="car:getCarColor($myCar)"/></a>
  </xsl:template>
</xsl:stylesheet>
```

**Note:** When a path is supplied via the extension function, the path is added to the ClassLoader.

### 19.1.2.1.2 User-Defined Jar Files

If access is via a JAR file, the URI of the JAR file must be specified using the following syntax:

\[
\text{xmlns:classNS="java:classname?path=jar:uri-of-jarfile!/"}
\]

The method is then called by using the prefix of the namespace URI that identifies the class: `classNS:method()`

**In the above:**

- `java:` indicates that a Java function is being called
- `classname` is the name of the user-defined class
? is the separator between the classname and the path
path=jar: indicates that a path to a JAR file is being given
uri-of-jarfile is the URI of the jar file
!/ is the end delimiter of the path
classNS:method() is the call to the method

Alternatively, the classname can be given with the method call. Here are two examples of the syntax:

```text
xmlns:ns1="java:docx.layout.pages?path=jar:file:///c:/projects/docs/docx.jar!/"
s1:main()

xmlns:ns2="java?path=jar:file:///c:/projects/docs/docx.jar!/"
s2:docx.layout.pages.main()
```

Here is a complete XSLT example that uses a JAR file to call a Java extension function:

```xml
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:fn="http://www.w3.org/2005/xpath-functions"
xmlns:car="java?path=jar:file:///C:/test/Car1.jar!/>
<xsl:output exclude-result-prefixes="fn car xsl xs"/>
<xsl:template match="/"
    <xsl:variable name="myCar" select="car:Car1.new('red')" />
    <a><xsl:value-of select="car:Car1.getCarColor($myCar)"/></a>
</xsl:template>
<xsl:template match="car"/>
</xsl:stylesheet>

**Note:** When a path is supplied via the extension function, the path is added to the ClassLoader.

### 19.1.2.2.1.3 Java: Constructors

An extension function can be used to call a Java constructor. All constructors are called with the pseudo-function `new()`.

If the result of a Java constructor call can be *implicitly converted to XPath/XQuery datatypes*, then the Java extension function will return a sequence that is an XPath/XQuery datatype. If the result of a Java constructor call cannot be converted to a suitable XPath/XQuery datatype, then the constructor creates a wrapped Java object with a type that is the name of the class returning that Java object. For example, if a constructor for the class `java.util.Date` is called (`java.util.Date.new()`), then an object having a type `java.util.Date` is returned. The lexical format of the returned object may not match the lexical format of an XPath datatype and the value would therefore need to be converted to the lexical format of the required XPath datatype and then to the required XPath datatype.
There are two things that can be done with a Java object created by a constructor:

- It can be assigned to a variable:
  ```xml
  <xsl:variable name="currentdate" select="date:new()"
  xmlns:date="java:java.util.Date" />
  ```

- It can be passed to an extension function (see Instance Method and Instance Fields):
  ```xml
  <xsl:value-of select="date:toString(date:new())"
  xmlns:date="java:java.util.Date" />
  ```

### 19.1.2.2.1.4 Java: Static Methods and Static Fields

A static method is called directly by its Java name and by supplying the arguments for the method. Static fields (methods that take no arguments), such as the constant-value fields \( \varepsilon \) and \( \mathrm{PI} \), are accessed without specifying any argument.

#### XSLT examples

Here are some examples of how static methods and fields can be called:

```xml
  select="jMath:cos(3.14)" />
```

```xml
  select="jMath:cos(jMath:PI())" />
```

```xml
  select="jMath:E() * jMath:cos(3.14)" />
```

Notice that the extension functions above have the form `prefix:fname()`. The prefix in all three cases is `jMath:`, which is associated with the namespace URI `java:java.lang.Math`. (The namespace URI must begin with `java:`. In the examples above it is extended to contain the class name (`java.lang.Math`).) The `fname()` part of the extension functions must match the name of a public class (e.g. `java.lang.Math`) followed by the name of a public static method with its argument/s (such as `cos(3.14)`) or a public static field (such as `PI()`).

In the examples above, the class name has been included in the namespace URI. If it were not contained in the namespace URI, then it would have to be included in the `fname()` part of the extension function. For example:

```xml
<xsl:value-of xmlns:java="java:"
  select="java:java.lang.Math.cos(3.14)" />
```

#### XQuery example

A similar example in XQuery would be:

```xml
<cosine xmlns:jMath="java:java.lang.Math">
  {jMath:cos(3.14)}
</cosine>
```
19.1.2.2.1.5  Java: Instance Methods and Instance Fields

An instance method has a Java object passed to it as the first argument of the method call. Such a Java object typically would be created by using an extension function (for example a constructor call) or a stylesheets parameter/variable. An XSLT example of this kind would be:

```xml
<xsl:stylesheet version="1.0" exclude-result-prefixes="date"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:date="java:java.util.Date"
    xmlns:jlang="java:java.lang">
    <xsl:param name="CurrentDate" select="date:new()"/>
    <xsl:template match="/">
        <enrollment institution-id="Altova School"
            date="{date:toString($CurrentDate)}" type="
            {jlang:Object.toString(jlang:Object.getClass( date:new() ))}"/>
    </xsl:template>
</xsl:stylesheet>
```

In the example above, the value of the node `enrollment/@type` is created as follows:

1. An object is created with a constructor for the class `java.util.Date` (with the `date:new()` constructor).
2. This Java object is passed as the argument of the `jlang.Object.getClass` method.
3. The object obtained by the `getClass` method is passed as the argument to the `jlang.Object.toString` method.

The result (the value of `@type`) will be a string having the value: `java.util.Date`.

An instance field is theoretically different from an instance method in that it is not a Java object per se that is passed as an argument to the instance field. Instead, a parameter or variable is passed as the argument. However, the parameter/variable may itself contain the value returned by a Java object. For example, the parameter `CurrentDate` takes the value returned by a constructor for the class `java.util.Date`. This value is then passed as an argument to the instance method `date:toString` in order to supply the value of `/enrollment/@date`.

19.1.2.2.1.6  Datatypes: XPath/XQuery to Java

When a Java function is called from within an XPath/XQuery expression, the datatype of the function's arguments is important in determining which of multiple Java classes having the same name is called.

In Java, the following rules are followed:

- If there is more than one Java method with the same name, but each has a different number of arguments than the other/s, then the Java method that best matches the number of arguments in the function call is selected.
• The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding Java datatype. If the supplied XPath/XQuery type can be converted to more than one Java type (for example, `xs:integer`), then that Java type is selected which is declared for the selected method. For example, if the Java method being called is `fx(decimal)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to Java's `decimal` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to Java datatypes.

<table>
<thead>
<tr>
<th>XPath/XQuery Datatype</th>
<th>Java Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xs:string</code></td>
<td><code>java.lang.String</code></td>
</tr>
<tr>
<td><code>xs:boolean</code></td>
<td><code>boolean</code> (primitive), <code>java.lang.Boolean</code></td>
</tr>
<tr>
<td><code>xs:integer</code></td>
<td><code>int</code>, <code>long</code>, <code>short</code>, <code>byte</code>, <code>float</code>, <code>double</code>, and the wrapper classes of these, such as <code>java.lang.Integer</code></td>
</tr>
<tr>
<td><code>xs:float</code></td>
<td><code>float</code> (primitive), <code>java.lang.Float</code>, <code>double</code> (primitive)</td>
</tr>
<tr>
<td><code>xs:double</code></td>
<td><code>double</code> (primitive), <code>java.lang.Double</code></td>
</tr>
<tr>
<td><code>xs:decimal</code></td>
<td><code>float</code> (primitive), <code>java.lang.Float</code>, <code>double</code> (primitive), <code>java.lang.Double</code></td>
</tr>
</tbody>
</table>

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the Java type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct Java method based on the supplied information. For example, consider the following case.

• The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`. However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`. Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`. Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error. However, note that in some cases, it might be possible to create the required Java type by using a Java constructor.

**Datatypes: Java to XPath/XQuery**

When a Java method returns a value, the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, Java's `java.lang.Boolean` and `boolean` datatypes are converted to `xsd:boolean`.
One-dimensional arrays returned by functions are expanded to a sequence. Multi-dimensional arrays will not be converted, and should therefore be wrapped.

When a wrapped Java object or a datatype other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a Java method (e.g. `toString`) to convert the Java object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as` expression).

19.1.2.2.2 .NET Extension Functions

If you are working on the .NET platform on a Windows machine, you can use extension functions written in any of the .NET languages (for example, C#). A .NET extension function can be used within an XPath or XQuery expression to invoke a constructor, property, or method (static or instance) within a .NET class.

A property of a .NET class is called using the syntax `get_PropertyName()`.

This section is organized into the following sub-sections:

- .NET: Constructors
- .NET: Static Methods and Static Fields
- .NET: Instance Methods and Instance Fields
- Datatypes: XPath/XQuery to .NET
- Datatypes: .NET to XPath/XQuery

Form of the extension function

The extension function in the XPath/XQuery expression must have the form `prefix:fname()`.

- The `prefix:` part is associated with a URI that identifies the .NET class being addressed.
- The `fname()` part identifies the constructor, property, or method (static or instance) within the .NET class, and supplies any argument/s, if required.
- The URI must begin with `clitype:` (which identifies the function as being a .NET extension function).
- The `prefix:fname()` form of the extension function can be used with system classes and with classes in a loaded assembly. However, if a class needs to be loaded, additional parameters containing the required information will have to be supplied.

Parameters

To load an assembly, the following parameters are used:

- `asm` The name of the assembly to be loaded.
- `ver` The version number (maximum of four integers separated by periods).
- `sn` The key token of the assembly’s strong name (16 hex digits).
- `from` A URI that gives the location of the assembly (DLL) to be loaded. If the
URI is relative, it is relative to the XSLT or XQuery document. If this parameter is present, any other parameter is ignored.

**partialname**  
The partial name of the assembly. It is supplied to Assembly.LoadWith.PartialName(), which will attempt to load the assembly. If partialname is present, any other parameter is ignored.

**loc**  
The locale, for example, en-US. The default is neutral.

If the assembly is to be loaded from a DLL, use the from parameter and omit the sn parameter. If the assembly is to be loaded from the Global Assembly Cache (GAC), use the sn parameter and omit the from parameter.

A question mark must be inserted before the first parameter, and parameters must be separated by a semi-colon. The parameter name gives its value with an equals sign (see example below).

**Examples of namespace declarations**

An example of a namespace declaration in XSLT that identifies the system class System.Environment:

```xml
xmlns:myns="clitype:System.Environment"
```

An example of a namespace declaration in XSLT that identifies the class to be loaded as Trade.Forward.Scrip:

```xml
xmlns:myns="clitype:Trade.Forward.Scrip?asm=forward;version=10.6.2.1"
```

An example of a namespace declaration in XQuery that identifies the system class MyManagedDLL.testClass. Two cases are distinguished:

1. When the assembly is loaded from the GAC:
   ```xml
   declare namespace cs="clitype:MyManagedDLL.testClass?asm=MyManagedDLL;ver=1.2.3.4;loc=neutral;sn=b9f091b72dcccfa8";
   ```

2. When the assembly is loaded from the DLL (complete and partial references below):
   ```xml
   declare namespace cs="clitype:MyManagedDLL.testClass?from=file:///C:/Altova
   Projects/extFunctions/MyManagedDLL.dll;
   ```

   ```xml
   declare namespace cs="clitype:MyManagedDLL.testClass?
   from=MyManagedDLL.dll;
   ```

**XSLT example**

Here is a complete XSLT example that calls functions in system class System.Math:

```xml
<xsl:stylesheet version="2.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:fn="http://www.w3.org/2005/xpath-functions">
 <xsl:output method="xml" omit-xml-declaration="yes" />
 <xsl:template match="/">
   <math xmlns:math="clitype:System.Math">
```

The namespace declaration on the element `math` associates the prefix `math:` with the URI `clitype:System.Math`. The `clitype:` beginning of the URI indicates that what follows identifies either a system class or a loaded class. The `math:` prefix in the XPath expressions associates the extension functions with the URI (and, by extension, the class) `System.Math`. The extension functions identify methods in the class `System.Math` and supply arguments where required.

**XQuery example**

Here is an XQuery example fragment similar to the XSLT example above:

```xml
<math xmlns:math="clitype:System.Math">
  {math:Sqrt(9)}
</math>
```

As with the XSLT example above, the namespace declaration identifies the .NET class, in this case a system class. The XQuery expression identifies the method to be called and supplies the argument.

### 19.1.2.2.1 .NET: Constructors

An extension function can be used to call a .NET constructor. All constructors are called with the pseudo-function `new()`. If there is more than one constructor for a class, then the constructor that most closely matches the number of arguments supplied is selected. If no constructor is deemed to match the supplied argument/s, then a 'No constructor found' error is returned.

**Constructors that return XPath/XQuery datatypes**

If the result of a .NET constructor call can be implicitly converted to XPath/XQuery datatypes, then the .NET extension function will return a sequence that is an XPath/XQuery datatype.

**Constructors that return .NET objects**

If the result of a .NET constructor call cannot be converted to a suitable XPath/XQuery datatype, then the constructor creates a wrapped .NET object with a type that is the name of the class returning that object. For example, if a constructor for the class `System.DateTime` is called (with `System.DateTime.new()`), then an object having a type `System.DateTime` is returned.

The lexical format of the returned object may not match the lexical format of a required XPath datatype. In such cases, the returned value would need to be: (i) converted to the lexical format of the required XPath datatype; and (ii) cast to the required XPath datatype.

There are three things that can be done with a .NET object created by a constructor:
• It can be used within a variable:
  `<xsl:variable name="currentdate" select="date:new(2008, 4, 29)" xmlns:date="clitype:System.DateTime" />`

• It can be passed to an extension function (see Instance Method and Instance Fields):
  `<xsl:value-of select="date:ToString(date:new(2008, 4, 29))" xmlns:date="clitype:System.DateTime" />`

• It can be converted to a string, number, or boolean:
  `<xsl:value-of select="xs:integer(data:get_Month(date:new(2008, 4, 29)))" xmlns:date="clitype:System.DateTime" />`

19.1.2.2.2 .NET: Static Methods and Static Fields

A static method is called directly by its name and by supplying the arguments for the method. The name used in the call must exactly match a public static method in the class specified. If the method name and the number of arguments that were given in the function call matches more than one method in a class, then the types of the supplied arguments are evaluated for the best match. If a match cannot be found unambiguously, an error is reported.

Note: A field in a .NET class is considered to be a method without any argument. A property is called using the syntax get_PropertyName().

Examples

An XSLT example showing a call to a method with one argument (System.Math.Sin(arg)):

An XSLT example showing a call to a field (considered a method with no argument)
(System.Double.MaxValue()):

An XSLT example showing a call to a property (syntax is get_PropertyName())
(System.String()):
  `<xsl:value-of select="string:get_Length('my string')" xmlns:string="clitype:System.String"/>

An XQuery example showing a call to a method with one argument (System.Math.Sin(arg)):
  `<sin xmlns:math="clitype:System.Math">
    { math:Sin(30) }
  </sin>

19.1.2.2.3 .NET: Instance Methods and Instance Fields

An instance method has a .NET object passed to it as the first argument of the method call. This .NET object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:
In the example above, a System.DateTime constructor (new(2008, 4, 29)) is used to create a .NET object of type System.DateTime. This object is created twice, once as the value of the variable releasedate, a second time as the first and only argument of the System.DateTime.ToString() method. The instance method System.DateTime.ToString() is called twice, both times with the System.DateTime constructor (new(2008, 4, 29)) as its first and only argument. In one of these instances, the variable releasedate is used to get the .NET object.

Instance methods and instance fields

The difference between an instance method and an instance field is theoretical. In an instance method, a .NET object is directly passed as an argument; in an instance field, a parameter or variable is passed instead—though the parameter or variable may itself contain a .NET object. For example, in the example above, the variable releasedate contains a .NET object, and it is this variable that is passed as the argument of ToString() in the second date element constructor. Therefore, the ToString() instance in the first date element is an instance method while the second is considered to be an instance field. The result produced in both instances, however, is the same.

19.1.2.2.4 Datatypes: XPath/XQuery to .NET

When a .NET extension function is used within an XPath/XQuery expression, the datatypes of the function's arguments are important for determining which one of multiple .NET methods having the same name is called.

In .NET, the following rules are followed:

- If there is more than one method with the same name in a class, then the methods
available for selection are reduced to those that have the same number of arguments as the function call.

- The XPath/XQuery string, number, and boolean datatypes (see list below) are implicitly converted to a corresponding .NET datatype. If the supplied XPath/XQuery type can be converted to more than one .NET type (for example, `xs:integer`), then that .NET type is selected which is declared for the selected method. For example, if the .NET method being called is `fx(double)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to .NET's `double` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to .NET datatypes.

<table>
<thead>
<tr>
<th>XPath/XQuery Type</th>
<th>.NET Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xs:string</code></td>
<td><code>StringValue</code>, <code>string</code></td>
</tr>
<tr>
<td><code>xs:boolean</code></td>
<td><code>BooleanValue</code>, <code>bool</code></td>
</tr>
<tr>
<td><code>xs:integer</code></td>
<td><code>IntegerValue</code>, <code>decimal</code>, <code>long</code>, <code>integer</code>, <code>short</code>, <code>byte</code>, <code>double</code>, <code>float</code></td>
</tr>
<tr>
<td><code>xs:float</code></td>
<td><code>FloatValue</code>, <code>float</code>, <code>double</code></td>
</tr>
<tr>
<td><code>xs:double</code></td>
<td><code>DoubleValue</code>, <code>double</code></td>
</tr>
<tr>
<td><code>xs:decimal</code></td>
<td><code>DecimalValue</code>, <code>decimal</code>, <code>double</code>, <code>float</code></td>
</tr>
</tbody>
</table>

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the .NET type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct .NET method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`. However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error.

### 19.1.2.2.5 Datatypes: .NET to XPath/XQuery

When a .NET method returns a value and the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, .NET's `decimal` datatype is converted to `xsd:decimal`. 
When a .NET object or a datatye other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a .NET method (for example System.DateTime.ToString()) to convert the .NET object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as expression`).

### 19.1.2.2.3 MSXSL Scripts for XSLT

The `<msxsl:script>` element contains user-defined functions and variables that can be called from within XPath expressions in the XSLT stylesheet. The `<msxsl:script>` is a top-level element, that is, it must be a child element of `<xsl:stylesheet>` or `<xsl:transform>`.

The `<msxsl:script>` element must be in the namespace urn:schemas-microsoft-com:xslt (see example below).

#### Scripting language and namespace

The scripting language used within the block is specified in the `<msxsl:script>` element's language attribute and the namespace to be used for function calls from XPath expressions is identified with the implements-prefix attribute (see below).

```xml
<msxsl:script language="scripting-language" implements-prefix="user-namespace-prefix">
    function-1 or variable-1
    ...
    function-n or variable-n
</msxsl:script>
```

The `<msxsl:script>` element interacts with the Windows Scripting Runtime, so only languages that are installed on your machine may be used within the `<msxsl:script>` element. The .NET Framework 2.0 platform or higher must be installed for MSXSL scripts to be used. Consequently, the .NET scripting languages can be used within the `<msxsl:script>` element.

The language attribute accepts the same values as the language attribute on the HTML `<script>` element. If the language attribute is not specified, then Microsoft JScript is assumed as the default.

The implements-prefix attribute takes a value that is a prefix of a declared in-scope namespace. This namespace typically will be a user namespace that has been reserved for a function library. All functions and variables defined within the `<msxsl:script>` element will be in the namespace identified by the prefix specified in the implements-prefix attribute. When a function is called from within an XPath expression, the fully qualified function name must be in the same namespace as the function definition.

#### Example

Here is an example of a complete XSLT stylesheet that uses a function defined within a `<msxsl:script>` element.
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:msxsl="urn:schemas-microsoft-com:xslt"
   xmlns:user="http://mycompany.com/mynamespace">
  <msxsl:script language="VBScript" implements-prefix="user">
    <![CDATA[
      ' Input: A currency value: the wholesale price
      ' Returns: The retail price: the input value plus 20% margin, rounded to the nearest cent
      dim a as integer = 13
      Function AddMargin(WholesalePrice) as integer
        AddMargin = WholesalePrice * 1.2 + a
      End Function
    ]]>
  </msxsl:script>

  <xsl:template match="/">
    <html>
      <body>
        <p>
          <b>Total Retail Price = $<xsl:value-of select="user:AddMargin(50)" /></b>
        </p>
        <p>
          <b>Total Wholesale Price = $<xsl:value-of select="50" /></b>
        </p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>

Datatypes
The values of parameters passed into and out of the script block are limited to XPath datatypes. This restriction does not apply to data passed among functions and variables within the script block.

Assemblies
An assembly can be imported into the script by using the msxsl:assembly element. The assembly is identified via a name or a URI. The assembly is imported when the stylesheet is compiled. Here is a simple representation of how the msxsl:assembly element is to be used.

<msxsl:script>
  <msxsl:assembly name="myAssembly.assemblyName" />
  <msxsl:assembly href="pathToAssembly" />

  ...
</msxsl:script>
The assembly name can be a full name, such as:

"system.Math, Version=3.1.4500.1 Culture=neutral PublicKeyToken=a46b3f648229c514"

or a short name, such as "myAssembly.Draw".

Namespaces
Namespaces can be declared with the `<msxsl:using>` element. This enables assembly classes to be written in the script without their namespaces, thus saving you some tedious typing. Here is how the `<msxsl:using>` element is used so as to declare namespaces.

```xml
<msxsl:script>
  <msxsl:using namespace="myAssemblyNS.NamespaceName" />
  ...
</msxsl:script>
```

The value of the `namespace` attribute is the name of the namespace.
19.2 Technical Data

This section contains information on some technical aspects of your software. This information is organized into the following sections:

- OS and Memory Requirements
- Altova Engines
- Unicode Support
- Internet Usage

19.2.1 OS and Memory Requirements

Operating System

Altova software applications are available for the following platforms:

- Windows 7 SP1 with Platform Update, Windows 8, Windows 10
- Windows Server 2008 R2 SP1 with Platform Update or newer

Memory

Since the software is written in C++ it does not require the overhead of a Java Runtime Environment and typically requires less memory than comparable Java-based applications. However, each document is loaded fully into memory so as to parse it completely and to improve viewing and editing speed. As a result, the memory requirement increases with the size of the document.

Memory requirements are also influenced by the unlimited Undo history. When repeatedly cutting and pasting large selections in large documents, available memory can rapidly be depleted.

19.2.2 Altova Engines

XML Validator

When opening an XML document, the application uses its built-in XML validator to check for well-formedness, to validate the document against a schema (if specified), and to build trees and infosets. The XML validator is also used to provide intelligent editing help while you edit documents and to dynamically display any validation error that may occur.

The built-in XML validator implements the Final Recommendation of the W3C’s XML Schema 1.0 and 1.1 specifications. New developments recommended by the W3C’s XML Schema Working Group are continuously being incorporated in the XML validator, so that Altova products give you a state-of-the-art development environment.

XSLT and XQuery Engines

Altova products use the Altova XSLT 1.0, 2.0, and 3.0 Engines and the Altova XQuery 1.0 and 3.1 Engines. If one of these engines is included in the product, then documentation about implementation-specific behavior for each engine is given in the appendices of the documentation.
Note: Altova MapForce generates code using the XSLT 1.0, 2.0 and XQuery 1.0 engines.

19.2.3 Unicode Support

Altova's XML products provide full Unicode support. To edit an XML document, you will also need a font that supports the Unicode characters being used by that document.

Please note that most fonts only contain a very specific subset of the entire Unicode range and are therefore typically targeted at the corresponding writing system. If some text appears garbled, the reason could be that the font you have selected does not contain the required glyphs. So it is useful to have a font that covers the entire Unicode range, especially when editing XML documents in different languages or writing systems. A typical Unicode font found on Windows PCs is Arial Unicode MS.

In the /Examples folder of your application folder you will find an XHTML file called UnicodeUTF-8.html that contains the following sentence in a number of different languages and writing systems:

- When the world wants to talk, it speaks Unicode
- Wenn die Welt miteinander spricht, spricht sie Unicode
- 世界的に話すならUnicodeです

Opening this XHTML file will give you a quick impression of Unicode's possibilities and also indicate what writing systems are supported by the fonts available on your PC.

19.2.4 Internet Usage

Altova applications will initiate Internet connections on your behalf in the following situations:

- If you click the "Request evaluation key-code" in the Registration dialog (Help | Software Activation), the three fields in the registration dialog box are transferred to our web server by means of a regular http (port 80) connection and the free evaluation key-code is sent back to the customer via regular SMTP e-mail.
- In some Altova products, you can open a file over the Internet (File | Open | Switch to URL). In this case, the document is retrieved using one of the following protocol methods and connections: HTTP (normally port 80), FTP (normally port 20/21), HTTPS (normally port 443). You could also run an HTTP server on port 8080. (In the URL dialog, specify the port after the server name and a colon.)
- If you open an XML document that refers to an XML Schema or DTD and the document is specified through a URL, the referenced schema document is also retrieved through a HTTP connection (port 80) or another protocol specified in the URL (see Point 2 above). A schema document will also be retrieved when an XML file is validated. Note that validation might happen automatically upon opening a document if you have instructed the application to do this (in the File tab of the Options dialog (Tools | Options)).
- In Altova applications using WSDL and SOAP, web service connections are defined by the WSDL documents.
- If you are using the Send by Mail command (File | Send by Mail) in XMLSpy, the current selection or file is sent by means of any MAPI-compliant mail program installed on the user's PC.
• As part of Software Activation and LiveUpdate as further described in the Altova Software License Agreement.
19.3 License Information

This section contains information about:

- the distribution of this software product
- software activation and license metering
- the license agreement governing the use of this product

Please read this information carefully. It is binding upon you since you agreed to these terms when you installed this software product.

To view the terms of any Altova license, go to the Altova Legal Information page at the Altova website.

19.3.1 Electronic Software Distribution

This product is available through electronic software distribution, a distribution method that provides the following unique benefits:

- You can evaluate the software free-of-charge for 30 days before making a purchasing decision. *(Note: Altova MobileTogether Designer is licensed free of charge.)*
- Once you decide to buy the software, you can place your order online at the Altova website and get a fully licensed product within minutes.
- When you place an online order, you always get the latest version of our software.
- The product package includes an onscreen help system that can be accessed from within the application interface. The latest version of the user manual is available at www.altova.com in (i) HTML format for online browsing, and (ii) PDF format for download (and to print if you prefer to have the documentation on paper).

30-day evaluation period

After downloading this product, you can evaluate it for a period of up to 30 days free of charge. About 20 days into the evaluation period, the software will start to remind you that it has not yet been licensed. The reminder message will be displayed once each time you start the application. If you would like to continue using the program after the 30-day evaluation period, you must purchase a product license, which is delivered in the form of a license file containing a key code. Unlock the product by entering this key code in the Software Activation dialog of your product.

You can purchase product licenses at the Altova online shop.

Helping Others within Your Organization to Evaluate the Software

If you wish to distribute the evaluation version within your company network, or if you plan to use it on a PC that is not connected to the Internet, you may distribute only the installer file, provided that this file is not modified in any way. Any person who accesses the software installer that you have provided must request their own 30-day evaluation license key code and after expiration of their evaluation period, must also purchase a license in order to be able to continue using the product.
19.3.2 Software Activation and License Metering

As part of Altova’s Software Activation, the software may use your internal network and Internet connection for the purpose of transmitting license-related data at the time of installation, registration, use, or update to an Altova-operated license server and validating the authenticity of the license-related data in order to protect Altova against unlicensed or illegal use of the software and to improve customer service. Activation is based on the exchange of license-related data such as operating system, IP address, date/time, software version, and computer name, along with other information between your computer and an Altova license server.

Your Altova product has a built-in license metering module that further helps you avoid any unintentional violation of the End User License Agreement. Your product is licensed either as a single-user or multi-user installation, and the license-metering module makes sure that no more than the licensed number of users use the application concurrently.

This license-metering technology uses your local area network (LAN) to communicate between instances of the application running on different computers.

Single license
When the application starts up, as part of the license metering process, the software sends a short broadcast datagram to find any other instance of the product running on another computer in the same network segment. If it doesn't get any response, it will open a port for listening to other instances of the application.

Multi-user license
If more than one instance of the application is used within the same LAN, these instances will briefly communicate with each other on startup. These instances exchange key-codes in order to help you to better determine that the number of concurrent licenses purchased is not accidentally violated. This is the same kind of license metering technology that is common in the Unix world and with a number of database development tools. It allows Altova customers to purchase reasonably-priced concurrent-use multi-user licenses.

We have also designed the applications so that they send few and small network packets so as to not put a burden on your network. The TCP/IP ports (2799) used by your Altova product are officially registered with the IANA (see the IANA Service Name Registry for details) and our license-metering module is tested and proven technology.

If you are using a firewall, you may notice communications on port 2799 between the computers that are running Altova products. You are, of course, free to block such traffic between different groups in your organization, as long as you can ensure by other means, that your license agreement is not violated.

If you are online, you will also notice that your Altova software provides many useful functions. These are unrelated to the license-metering technology.
19.3.3 Altova End-User License Agreement

- The Altova End-User License Agreement is available here: https://www.altova.com/legal/eula
- Altova's Privacy Policy is available here: https://www.altova.com/privacy
Chapter 20

Glossary
Glossary

The glossary section includes the list of terms pertaining to MapForce.
20.1 C

Component
In MapForce, the term “component” is what represents visually the structure (schema) of your data, or how data is to be transformed (functions). Components are the central building pieces of any mapping. On the mapping area, components appear as rectangles. The following are examples of MapForce components:

- Constants
- Databases
- Filters
- Conditions
- Flat files (CSV, fixed-length, and other text files)
- Function components
- EDI documents (UN/EDIFACT, ANSI X12, HL7)
- Excel 2007+ files
- Simple input components
- Simple output components
- Variables
- XBRL documents
- XML Schemas and DTDs

Connection
A connection is a line that you can draw between two connectors. By drawing connections, you instruct MapForce to transform data in a specific way (for example, read data from an XML document and write it to another XML document).

Connector
A connector is a small triangle displayed on the left or right side of a component. The connectors displayed on the left of a component provide data entry points to that component. The connectors displayed on the right of a component provide data exit points from that component.
20.2  F

**Fixed Length Field (FLF)**
A common text format where data is conventionally separated into fields which have a fixed length (for example, the first 5 characters of every row represent a transaction ID, and the next 20 characters represent a transaction description).

**FlexText**
FlexText is a module in MapForce Enterprise Edition which enables you to convert data from non-standard or legacy text files of high complexity to other formats supported by MapForce, and vice versa.
20.3 G

Global Resources
Altova Global Resources represent a way to refer to files, folders, or databases so as to make these resources reusable, configurable and available across multiple Altova applications.
Input component

An input component is a MapForce component that enables you to pass simple values to a mapping. Input components are commonly used to pass file names or other string values to a mapping at runtime. Input components should not be confused with source components.
20.5  J

Join component
A Join component is a MapForce component which enables joining two or more structures on the mapping based on custom-defined conditions. It returns the association (joined set) of items that satisfy the condition. Joins are particularly useful to combine data from two structures which share a common field (such as an identity).
20.6 M

MapForce
MapForce is a Windows-based, multi-purpose IDE (integrated development environment) that enables you to transform data from one format to another, or from one schema to another, by means of a visual, "drag-and-drop"-style graphical user interface that does not require writing any program code. In fact, MapForce generates for you the program code which performs the actual data transformation (or data mapping). When you prefer not to generate program code, you can just run the transformation using the MapForce built-in transformation language (available in the MapForce Professional or Enterprise Editions).

Mapping
A MapForce mapping design (or simply "mapping") is the visual representation of how data is to be transformed from one format to another. A mapping consists of components that you add to the MapForce mapping area in order to create your data transformations (for example, convert XML documents from one schema to another). A valid mapping consists of one or several source components connected to one or several target components. You can run a mapping and preview its result directly in MapForce. You can generate code and execute it externally. You can also compile a mapping to a MapForce execution file and automate mapping execution using MapForce Server or FlowForce Server. MapForce saves mappings as files with .mfd extension.

MFF
The file name extension of MapForce function files.

MFD
The file name extension of MapForce design documents (mappings).

MFP
The file name extension of MapForce Project files.

MFT
The file name extension of MapForce FlextText template documents.
20.7  O

Output component

An output component (or "simple output") is a MapForce component which enables you to return a string value from the mapping. Output components represent just one possible type of target components, but should not be confused with the latter.
**20.8 P**

**parent-context**

*parent-context* is an optional argument in some MapForce core aggregation functions such as *min*, *max*, *avg*, *count*. In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.
20.9 S

Source component
A source component is a component from which MapForce reads data. When you run the mapping, MapForce reads the data supplied by the connector of the source component, converts it to the required type, and sends it to the connector of the target component.
Target component

A target component is a component to which MapForce writes data. When you run the mapping, a target component instructs MapForce to either generate a file (or multiple files) or output the result as a string value for further processing in an external program. A target component is the opposite of a source component.
Index

.NET extension functions,
  constructors, 1821
  datatype conversions, .NET to XPath/XQuery, 1824
  datatype conversions, XPath/XQuery to .NET, 1823
  for XSLT and XQuery, 1819
instance methods, instance fields, 1822
overview, 1819
static methods, static fields, 1822

2019, 1513

A

A to Z,
  sort component, 200
abs,
  as MapForce function (in lang | math functions), 1073
  as MapForce function (in xpath2 | numeric functions), 1099
Access database,
  updating based on IF condition, 623
acos,
  as MapForce function (in lang | math functions), 1073
ActiveX,
  integration at application level, 1669
  integration at document level, 1672
  integration prerequisites, 1665
ActiveX controls,
  adding to the Visual Studio Toolbox, 1667
add,
  as MapForce function (in core | math functions), 1015
  custom library, 972
ADO,
  as data connection interface, 348
  setting up a connection, 355
ADO.NET,
  setting up a connection, 360
age,
  as MapForce function (in lang | datetime functions), 1057
Altova extensions,
  chart functions (see chart functions), 1744
Altova XML Parser,
  about, 1828
ANSI X12,
  customizing message structure, 779
Ant,
  Building an Eclipse project with, 1369
  setting the environment variables for, 1472
Any,
  xs:any, 329
API,
  documentation, 1494
  overview, 1495
Application object, 1497
AS2,
  integration with Altova products, 1227
ASC X12, 692, 706
asin,
  as MapForce function (in lang | math functions), 1073
atan,
  as MapForce function (in lang | math functions), 1074
ATTLIST,
  DTD namespace URIs, 320
auto-format,
  as MapForce function (in edifact functions), 1054
Automated,
  processing, 1214
Automatic,
  loading of libraries, 972
auto-number,
  as MapForce function (in core | generator functions), 1009
avg,
  as MapForce function (in core | aggregate functions), 991
Axis2,
  configuring for SOAP Web services, 1124, 1141

B

Background Information, 1828
base-uri,
  as MapForce function (in xpath2 | accessors library), 1092
Batch,
Batch,
  processing automation, 1214

Binary files,
  adding to the mapping, 881
  as mapping components, 880
  component settings, 884
  reading data from, 885
  writing data to, 891

Block comment, 551

Bookmarks,
  bookmark margin, 552
  inserting, 552
  navigating, 552
  removing, 552

boolean,
  as MapForce function (in core | conversion functions), 996

Breakpoints,
  about, 296
  adding, 299
  removing, 299

Breakpoints window,
  about, 296, 305

Browser,
  applying filters, 555
  Database Query, 554
  filtering, 555

Browser view,
  context menu options, 556
  generating SQL statements, 556

build.xml,
  enabling the zip64mode in., 1472

Built-in engine,
  definition, 78
  using, 78

C

C#,
  code, 1354
  code generation settings, 1473
  error handling, 1498
  generating program code, 1355, 1363
  generating Web services in, 1127
  integrate generated code, 1383
  integration of MapForce, 1676
  reference to generated classes, 1442

C++,
  code, 1354
  code generation settings, 1473
  error handling, 1498
  generating program code, 1355, 1359
  integrate generated code, 1385
  reference to generated classes, 1428

Call graphs,
  SPS stylesheet, 1274

Canonical XML,
  digital signature, 341

capitalize,
  as MapForce function (in lang | string functions), 1077

celling,
  as MapForce function (in core | math functions), 1016

Certificate,
  digital signatures, 341

char-from-code,
  as MapForce function (in core | string functions), 1042

charset-decode,
  as MapForce function (in lang | string functions), 1077

charset-encode,
  as MapForce function (in lang | string functions), 1079

Chart functions,
  chart data structure for, 1801
  example, 1806
  listing, 1797

Class ID,
  in MapForce integration, 1683

Code,
  built in types, 1487
  integrating MapForce code, 1380
  SPL, 1475

Code generation,
  sample, 1511
  supported platforms, 1355

Code generation settings,
  defining for a folder in a project, 126
  defining globally for the entire project, 125

Code Generator, 1354

Code point,
  collation, 200

code-from-char,
  as MapForce function (in core | string functions), 1043

Collapse,
  regions, 553

Collation,
Collation,
locale collation, 200
sort component, 200
unicode code point, 200
columnname-to-index,
as MapForce function, 1090
COM API,
documentation, 1494
Comments,
Adding to target files, 327
Complex type,
sorting, 200
Component,
as application menu, 1336
definition of, 1837
deleted items, 117
sort data, 200
Components,
adding to the mapping, 73
aligning, 100
changing settings, 101
overview, 98
processing sequence, 278
searching, 99
concat,
(as function) example of usage, 623
as MapForce function (in core | string functions), 1043
Configure,
mff file, 972
SQL Editor settings, 561
Connection,
as application menu, 1338
definition of, 1837
Connections,
moving to a different component, 112
preserving on root element change, 112
Connector,
definition of, 1837
viewing the history of processed values, 301
Consolidating data,
merging XML files, 333
Constants,
adding to the mapping, 901
contains,
as MapForce function (in core | string functions), 1044
Context window,
about, 296, 303
convert-to-utc,
Database,
querying, 547
writing XML files to, 540

Database connection,
reusing from Global Resources, 376
setting up, 348
setup examples, 377
starting the wizard, 350

Database drivers,
overview, 352

Database exceptions,
handling, 589, 594
logging, 601, 605

Database objects,
adding to the mapping, 429
filtering, 429
removing from the mapping, 429

Database Query,
bookmarks, 552
commenting out text, 551
filtering tables, 555
generating SQL, 554, 558
regions, 553
Result view options, 563
text font options, 564

Database relationships,
defining in mappings, 442
preserving in mappings, 432

Database transactions,
enabling for stored procedures, 577
rolling back, 594
using in mappings, 589

Databases,
add to mapping, 425
as Global Resources, 1248
deleting table data, 483
executing mappings against, 448
inserting data into a table, 456
inserting data into multiple linked tables, 459
merging data into, 467
updating and inserting table data, 467
updating table data, 464
using bulk insert, 491
date-from-datetime,
as MapForce function (in lang | datetime functions), 1058
datet ime-add,
as MapForce function (in lang | datetime functions), 1058
datetime-diff,
Index

Digital certificates,
  trusting on Linux, 1194
  trusting on Mac, 1196
  trusting on Windows, 1197
Digital signature,
  activating, 338
  creating in XML output, 316
  settings, 341
distinct-values,
  as MapForce function (in core | sequence functions), 1023
Distribution,
  of Altova’s software products, 1831
divide,
  as MapForce function (in core | math functions), 1016
divide-integer,
  as MapForce function (in lang | math functions), 1074
document,
  as MapForce function (in xslt | xslt functions library), 1104
Documentation,
  defining SPS stylesheets, 1277
Documenting,
  mappings, 1268
Document-level,
  examples of integration of XMLSpy, 1676
DoTransform.bat,
  execute with RaptorXML Server, 1213
DTD,
  source and target, 320
Duplicate input, 47
  adding, 1336
duration-add,
  as MapForce function (in lang | datetime functions), 1064
duration-from-parts,
  as MapForce function (in lang | datetime functions), 1064
duration-subtract,
  as MapForce function (in lang | datetime functions), 1065

E

Eclipse,
  generating mapping code for, 1366
EDI,
  ASC X12, 706
  customizing validation, 786
  HIPAA X12, 710
  IATA PADIS, 718
  NCPDP SCRIPT, 721
  separators, 694
EDI files,
  as mapping components, 695
EDIFACT, 692
  mapping to XML, 745
  merged entries, 764
  terminology, 694
Edit,
  as application menu, 1330
element-available,
  as MapForce function (in xslt | xslt functions library), 1105
empty,
  as MapForce function (in lang | string functions), 1081
Empty fields,
  in CSV files, 619
  in flat text files, 629
Encoding,
  changing in CSV files, 619
  changing in flat text files, 629
Encoding settings,
  in XML output, 316
End User License Agreement, 1831, 1833
Enumerations,
  in MapForceControl, 1733
Enveloped,
  digital signature, 341
Environment variables,
  ANT_OPS, 1472
equal,
  as MapForce function (in core | logical functions), 1012
equal-or-greater,
  as MapForce function (in core | logical functions), 1012
equal-or-less,
  as MapForce function (in core | logical functions), 1013
Error handling,
  general description, 1498
Evaluation period,
  of Altova’s software products, 1831
Excel 2007+,
  adding as mapping components, 815
  as mapping component, 816
  as mapping source or target, 813
  formatting MapForce-generated output, 834
  generate multiple XML files from, 171
  mapping data to, 244
  mapping from database to, 832
  mapping from RSS feed, 1183
Excel 2007+,
  mapping to XML, 829
  reading mapping data from cell ranges, 820
  selecting cell ranges from the mapping component, 821
  writing mapping data to cell ranges, 820
Excel 2007+ component settings,
  changing, 827
Excel 2007+ worksheets,
  reading data from, 818
  writing data to, 818
Exceptions,
  adding, 238
  example, 239
  throwing when node is missing, 240
exists,
  as MapForce function (in core | sequence functions), 1024
exp,
  as MapForce function (in lang | math functions), 1074
Expand,
  regions, 553
Extension functions for XSLT and XQuery, 1809
Extension Functions in .NET for XSLT and XQuery,
  see under .NET extension functions, 1819
Extension Functions in Java for XSLT and XQuery,
  see under Java extension functions, 1810
Extension Functions in MSXSL scripts, 1825

F
false,
  as MapForce function (in xpath2 | boolean functions), 1093
Field,
  keys in text files, 616
File,
  as application menu, 1327
  as button on a component, 101
  as button on components, 161
File DSN,
  setting up, 367
File names,
  supplying as mapping input parameters, 166
File paths,
  fixing broken references, 136
  in generated code, 137
  of file-based databases, 134
  relative versus absolute, 132, 137
File/String,
  as button on a component, 101
  as button on components, 161
File: (default),
  as name of root node, 161
File: <dynamic>,
  as name of root node, 161
Files,
  multiple from Excel, 171
Fill character,
  in flat text files, 629
Filter,
  data from components, 212
  database objects, 555
  merging XML files, 333
  the Online Browser, 555
Filtering,
  data from components, 212
  database objects on the mapping, 429
  database tables, 212
  in databases, 524
Filters,
  adding to the mapping, 212
find-substring,
  as MapForce function (in lang | string functions), 1081
Firebird,
  Connecting through JDBC, 380
  Connecting through ODBC, 378
first-items,
  as MapForce function (in core | sequence functions), 1025
Fixed,
  length files - mapping, 611
Flat file,
  mapping, 611
FlexText,
  as target component, 653
  changing component settings, 651
  containers, 636
  definition of, 1838
  interface, 636
  overview of, 636
  tutorial, 639
FlexText functions,
  Ignore, 673
  Node, 672
  Repeated split, 654
  Split once, 662
  Store as CSV (delimited), 674
  Store as FLF (fixed length), 681
Index 1853

**FlexText functions,**
  Store value, 684
  Switch, 668

**FLF,**
  definition of, 1838

**floor,**
  as MapForce function (in core | math functions), 1016

**Folder,**
  layout - Database Query, 554

**Foreign Key,**
  in database mappings, 432, 442

**format-date,**
  as MapForce function (in core | conversion functions), 996

**format-dateTime,**
  as MapForce function (in core | conversion functions), 997

**format-guid-string,**
  as MapForce function (in lang | string functions), 1082

**format-number,**
  as MapForce function (in core | conversion functions), 999

**format-time,**
  as MapForce function (in core | conversion functions), 1002

**Function,**
  as application menu, 1339

**function-available,**
  as MapForce function (in xslt | xslt functions library), 1105

**Functions,**
  adding as mapping components, 899
  adding parameters to, 904
  applying to multiple descendant items, 905, 911
  applying to multiple items, 905
  applying to multiple items conditionally, 917
  deleting parameters from, 904
  finding in the Libraries window, 902
  finding occurrences in active mapping, 902
  viewing the argument data type of, 903
  viewing the description of, 903

**Functions used by,** 1274

**G**

**Generate,**
  code from schema, 1354
  digital signature, 338

**Generated code,**
  throwing exceptions from, 238

**generate-id,**
  as MapForce function (in xslt | xslt functions library), 1106

**generate-sequence,**
  as MapForce function (in core | sequence functions), 1025

**get-fileext,**
  as MapForce function (in core | file path functions), 1007

**get-folder,**
  as MapForce function (in core | file path functions), 1007

**Global objects,**
  in SPL, 1479

**Global Resources,**
  creating, 1246
  examples of usage, 1252, 1254, 1255, 1258
  in various execution environments, 1250
  introduction to, 1246

**greater,**
  as MapForce function (in core | logical functions), 1013

**group-adjacent,**
  as MapForce function (in core | sequence functions), 1026

**group-by,**
  as MapForce function (in core | sequence functions), 1028

**group-ending-with,**
  as MapForce function (in core | sequence functions), 1029

**group-into-blocks,**
  as MapForce function (in core | sequence functions), 1031

**group-starting-with,**
  as MapForce function (in core | sequence functions), 1033

**Health Level 7,**
  example, 877

**Help,**
  as application menu, 1348

**Hierarchy,**
  from text files, 616

**HIPAA,**
  generating the X12 999 Implementation Acknowledgement, 709

**HIPAA X12, 692, 710**

**HL7, 692**

**HL7 2.6 to 3.x,**
  example, 877

**hour-from-datetime,**
  as MapForce function (in lang | datetime functions), 1066

**hour-from-duration,**
  as MapForce function (in lang | datetime functions), 1066
HRESULT,
    and error handling, 1498
HTML,
    integration of MapForce, 1685
    mapping documentation, 1268
    preview mapping output as, 1263
HTML example,
    of MapForceControl integration, 1683, 1684, 1685
HTTPS,
    calling Web services through, 1192

I
IATA PADIS, 692, 718
IBM DB2,
    connecting through ODBC, 381
    reading from XML type fields, 537
    writing to XML type fields, 537
IBM DB2 for i,
    connecting through ODBC, 386
IBM Informix,
    connecting through JDBC, 389
Icons,
    in Messages window of Database Query, 558
    in Results window of Database Query, 558
If-Else conditions,
    adding to the mapping, 212
Ignore,
    as FlexText function, 673
IIS,
    configuring for SOAP Web services, 1127
Impact analysis,
    SPS stylesheet, 1274
implicit-timezone,
    as MapForce function (in xpath2 | context functions), 1095
index-to-columnname,
    as MapForce function, 1091
Inline XBRL,
    mapping data from, 840
Input component,
    definition of, 1840
Input parameters,
    in user-defined functions, 934, 937
Insert,
    as application menu, 1331
    block comment, 551
    bookmarks, 552
    comments, 551
    line comment, 551
    regions, 553
    SQL WHERE component, 521
Instance,
    changing the path reference to, 132
Integrate,
    into C#, 1383
    into C++, 1385
    into Java, 1381
Integrate MapForce code, 1380
Integrating,
    MapForce in applications, 1664
Internet usage,
    in Altova products, 1829
is-not-null,
    as MapForce function (in db functions), 1052
is-null,
    as MapForce function (in db functions), 1052
is-xsi-nil,
    as MapForce function (in core | node functions), 1018
Item,
    missing, 117
item-at,
    as MapForce function (in core | sequence functions), 1034
items-from-till,
    as MapForce function (in core | sequence functions), 1035

J
Java, 1690
    avoiding exceptions in generated code, 1472
    code, 1354
    generating program code, 1366
    generating Web services in, 1141
    integrate generated code, 1381
    reference to generated classes, 1457
Java extension functions,
    constructors, 1815
    datatype conversions, Java to XPath/XQuery, 1818
    datatype conversions, XPath/XQuery to Java, 1817
    for XSLT and XQuery, 1810
    instance methods, instance fields, 1817
    overview, 1810
    static methods, static fields, 1816
Java extension functions,
user-defined class files, 1812
user-defined JAR files, 1814

JavaScript,
error handling, 1498

JDBC,
as data connection interface, 348
connect to Teradata, 422
handling references in generated code, 1368
setting up a connection (Windows), 370

JScript,
code-generation sample, 1511

JSON,
mapping example, 803
mapping from additional properties, 803
mapping from Web services, 1178

JSON files,
as source or target components, 797

Keeping data,
when using value-map, 233

Keeping data unchanged,
passing through a value-map, 233

Key,
fields in text files, 616
sort key, 200

last,
as MapForce function (in xpath2 | context functions), 1095
last-items,
as MapForce function (in core | sequence functions), 1035

Layout,
Browser, 554

Leading zeros,
Preserving during conversion, 735

leapyear,
as MapForce function (in lang | datetime functions), 1067

left,
as MapForce function (in lang | string functions), 1082

left-trim,
M

Mac,
executing mappings with Web service calls through HTTPS, 1207
transferring client certificates to, 1207
trusting server certificates on, 1196
main-md-filepath,
as MapForce function (in core | file path functions), 1007
MapForce,
API, 1494
basic concepts, 23
integration, 1664
overview, 16
MapForce API, 1494
overview, 1495
MapForce integration,
example of, 1683, 1684, 1685
MapForce plug-in for Eclipse,
about, 1298, 1303, 1309
accessing common menus and functions, 1306
configuring for automatic code generation, 1316
creating a MapForce/Eclipse project, 1309
creating new mappings, 1311
extending functionality, 1319
extension point, 1319
importing mappings into an Eclipse project, 1313
installing, 1299
switching to the MapForce perspective, 1303
working with mappings and projects, 1309
MapForce plug-in for Visual Studio,
about, 1288
accessing common menus and functions, 1293
enabling, 1289
working with mappings and projects, 1291
MapForce samples,
location on disk, 33
MapForce Server,
automating mappings, 1214
compiling mappings for, 1221
throwing exceptions from, 238
MapForceCommand,
in MapForceControl, 1713
MapForceCommands,
in MapForceControl, 1715

MapForceControl, 1716
documentation of, 1664
eexample of integration at application level, 1683, 1684, 1685
eexamples of integration at document level, 1676
integration using C#, 1676
integration using HTML, 1685
integration using Visual Basic, 1700
object reference, 1713
MapForceControlDocument, 1724
MapForceControlPlaceHolder, 1730
Mapping,
creating, 73
debugging, 290
definition of, 1842
Documenting, 1268
flat file format, 611
predefined SPS stylesheets for documentation, 1274
processing sequence, 278
source driven - mixed content, 139
validating, 79
Mapping input,
supplying custom file name as, 166
Supplying multiple files as, 161, 163, 165
Mapping methods,
standard, 139
standard / mixed / copy all, 139
target-driven, 139
Mapping output,
Generating multiple files as, 161, 165
Mappings,
automated processing, 1214
MariaDB,
connect through ODBC, 391
Marked items,
missing items, 117
match-pattern,
as MapForce function (in lang | string functions), 1082
max,
as MapForce function (in core | aggregate functions), 992
as MapForce function (in lang | math functions), 1075
maxLength,
using in node functions, 922
max-string,
as MapForce function (in core | aggregate functions), 993
Memory requirements, 1828
MERGE,
as statement in MapForce-generated SQL, 467
in MapForce-generated SQL, 473
Merged entries, 764
Merging,
XML files, 333
Messages,
icons in Database Query, 558
window - Database Query, 558
Method names in generating code,
reserving, 1472
mfd,
as file extension, 1842
mfd-filepath,
as MapForce function (in core | file path functions), 1007
mff, 972
as file extension, 1842
library file, 972
mff.xsd file, 972
mff file,
configuring, 972
mfp,
as file extension, 1842
mft,
as file extension, 1842
Microsoft Access,
connecting through ADO, 355, 393
Microsoft Excel,
see “Excel 2007+”, 813
Microsoft SharePoint Server,
adding files as components from, 74
Microsoft SQL Server,
connecting through ADO, 396
connecting through ODBC, 399
millisecond-from-datetime,
as MapForce function (in lang | datetime functions), 1067
millisecond-from-duration,
as MapForce function (in lang | datetime functions), 1067
min,
as MapForce function (in core | aggregate functions), 993
as MapForce function (in lang | math functions), 1075
minLength,
using in node functions, 922
min-string,
as MapForce function (in core | aggregate functions), 994
minute-from-datetime,
as MapForce function (in lang | datetime functions), 1068
minute-from-duration,
as MapForce function (in lang | datetime functions), 1068
Missing items, 117
Mixed, 139
content mapping, 139
content mapping example, 145
content mapping method, 139
source-driven mapping, 139
Mixed content,
Mapping, 146
modulus,
as MapForce function (in core | math functions), 1016
month-from-datetime,
as MapForce function (in lang | datetime functions), 1068
month-from-duration,
as MapForce function (in lang | datetime functions), 1068
MSXML,
generating code for, 1473
supported versions, 1355
msxsl:script, 1825
Multiple consecutive elements,
Edifact - X12, 764
Multiple source,
to single target, 333
multiply,
as MapForce function (in core | math functions), 1017
Multiple XML files,
from a Excel, 171
MySQL,
connecting through ODBC, 402

N
Namespace URI,
DTD, 320
Namespace URLs,
and QNames, 323
Namespaces,
and wildcards (xs:any), 329
declaring custom, 335
namespace-uri-form-QName,
as MapForce function (in lang | QName functions), 1023
Navigate,
bookmarks, 552
NCPDP SCRIPT,
as mapping component, 721
negative,
as MapForce function (in lang | logical functions), 1072
nillable,
as attribute in XML schema, 323
Node,
   as FlexText function, 672
Node functions,
   creating, 905
   deleting, 909
   editing, 909
Node names,
   mapping data from/to, 254
node-name,
   as MapForce function (in core | node functions), 1018
   as MapForce function (in xpath2 | accessors library), 1092
node-name function,
   alternatives to using, 254
normalize-space,
   as MapForce function (in core | string functions), 1044
not-equal,
   as MapForce function (in core | logical functions), 1015
not-exists,
   as MapForce function (in core | sequence functions), 1035
now,
   as MapForce function (in lang | datetime functions), 1069
NULL,
   handling in database mappings, 494
NULL values,
   replace in multiple occurrences, 905
Nulls,
   handling in database components, 450
number,
   as MapForce function (in core | conversion functions), 1003
numeric,
   as MapForce function (in lang | logical functions), 1072

O
Object model,
   overview, 1497
ODBC,
   as data connection interface, 348
   connect to MariaDB, 391
   connect to Teradata, 418
   setting up a connection, 367
ODBC Drivers,
   checking availability of, 367
OLE DB,
   as data connection interface, 348
OOXML,
   as default Excel 2007 format, 813
OpenJDK,
   as Java Virtual Machine, 370
Options,
   Result view - Database Query, 563
   text fonts - Database Query, 564
Oracle,
   reading from XML type fields, 537
   writing to XML type fields, 537
Oracle database,
   connecting through JDBC, 408
   connecting through ODBC, 403
Order,
   components are processed, 278
ORDER BY,
   SQL where component, 521
Ordering data,
   sort component, 200
OS,
   for Altova products, 1828
Out of memory errors,
   troubleshooting, 77
Out of memory exceptions,
   resolving, 1472
Output,
   as application menu, 1340
   previewing, 82
   saving, 82
   validating, 81
Output component,
   definition of, 1843
Output parameters,
   in user-defined functions, 934
   in user-defined functions, 937
Overall documentation,
   SPS stylesheet, 1274
Overview,
   of MapForce API, 1495

P
pad-string-left,
   as MapForce function (in lang | string functions), 1083
pad-string-right,
   as MapForce function (in lang | string functions), 1083
Parameters,
Parameters,  
in user-defined functions, 937

parent-context,  
definition of, 1844

parse-date,  
as MapForce function (in core | conversion functions), 1003

parse-dateTime,  
as MapForce function (in core | conversion functions), 1003

parse-number,  
as MapForce function (in core | conversion functions), 1005

Parser,  
built into Altova products, 1828

parse-time,  
as MapForce function (in core | conversion functions), 1006

Passing through data,  
unchanged through value-map, 233

Password,  
digital signature, 341

Paths in generated code,  
making absolute, 95

PDF,  
mapping documentation, 1268  
preview mapping output as, 1263

pi,  
as MapForce function (in lang | math functions), 1075

Platforms,  
for Altova products, 1828

position,  
as MapForce function (in core | sequence functions), 1037

positive,  
as MapForce function (in lang | logical functions), 1073

PostgreSQL,  
connecting natively, 374

connecting through ODBC, 410

pow,  
as MapForce function (in lang | math functions), 1076

Precision,  
using in node functions, 922

Primary Key,  
generating during database insert action, 456

generating for a database component, 451

in database mappings, 432, 442

Priority Context,  
setting on functions, 281

Processing,  
automating mappings, 1214

Processing Instructions,  
Adding to target files, 327

Processing Instructions and Comments,  
mapping, 140

Processing sequence,  
of components in a mapping, 278

Progress OpenEdge database,  
connecting through JDBC, 415

connecting through ODBC, 412

Project,  
as application menu, 1334

Projects,  
closing, 123

creating, 123

opening, 123

searching, 123

Properties,  
value map table, 235

Protocol Buffers,  
as mappig components, 880

Q

QName,  
as MapForce function (in lang | QName functions), 1021

QName support, 323

QName-as-string,  
as MapForce function (in lang | QName functions), 1056

Question mark,  
missing items, 117

Quote character,  
in CSV files, 619

R

radians,  
as MapForce function (in lang | math functions), 1076

random,  
as MapForce function (in lang | math functions), 1076

RaptorXML Server,  
executing a transformation, 1213

Reference, 1326

Regions,  
collapsing, 553

creating, 553

expanding, 553
Regions
inserting, 553
removing, 553

Regular expressions
as parameter to the "match-pattern" function, 987
as parameter to the "tokenize-regexp" function, 987
in MapForce FlexText, 685
in node functions, 917
splitting the FlexText component with, 686
using in FlexText switch conditions, 687

Remove
block comment, 551
bookmarks, 552
comments, 551
line comment, 551
regions, 553
remove-fileext,
as MapForce function (in core | file path functions), 1008
remove-folder,
as MapForce function (in core | file path functions), 1008
remove-timezone,
as MapForce function (in lang | datetime functions), 1069

Repeated split
as FlexText function, 654
using the "delimited (floating) mode, 656
using the "delimited (line based)" mode, 658
using the "delimited (line starts with)" mode, 660
using the "fixed length" mode, 655
repeat-string,
as MapForce function (in lang | string functions), 1083
replace,
as MapForce function (in lang | string functions), 1084
replace-fileext,
as MapForce function (in core | file path functions), 1008
replicate-item,
as MapForce function (in core | sequence functions), 1040
replicate-sequence,
as MapForce function (in core | sequence functions), 1041
resolve-filepath,
as MapForce function (in core | file path functions), 1008
resolve-uri,
as MapForce function (in xpath2 | anyURI functions), 1093

Results
icons in Database Query, 558
window - Database Query, 558

Retaining data
passing through value-map, 233
reversefind-substring,
as MapForce function (in lang | string functions), 1084
right,
as MapForce function (in lang | string functions), 1084
right-trim,
as MapForce function (in lang | string functions), 1085
round,
as MapForce function (in core | math functions), 1017
round-half-to-even,
as MapForce function (in xpath2 | numeric functions), 1099
round-precision,
as MapForce function (in core | math functions), 1017

Rows
from Excel, 171
mapping from - text files, 616

RSS feed
mapping from, 1183
preview mapping output as, 1263

S

SAP IDoc
adding as mapping components, 723
mapping data to/from, 723
mapping to XML, 723

SAP IDocs, 692

Scale
using in node functions, 922

Schema
and XML mapping, 315
changing the path reference to, 132
code generator, 1354
generating for an XML file, 315
schemanativetype, 1476

Scripts in XSLT/XQuery
see under Extension functions, 1809

Search
files in the Projects window, 123
functions in the Libraries window, 902
items within mapping components, 99

second-from-datetime,
as MapForce function (in lang | datetime functions), 1069
second-from-duration,
as MapForce function (in lang | datetime functions), 1070

Section,
Section,  
    CDATA, 328
Select,  
    table data - Database Query, 558
Separators,  
    EDI, 694
Sequence,  
    of processing components, 278
set-empty,  
    as MapForce function (in core | sequence functions), 1042
set-null,  
    as MapForce function (in db functions), 1052
Settings,  
    digital signature, 341
    Result view - Database Query, 563
    text fonts - Database Query, 564
set-xsi-nil,  
    as MapForce function (in core | node functions), 1020
Sign,  
    digital signature, 338
Signature,  
    settings, 341
Simple type,  
    sorting, 200
sin,  
    as MapForce function (in lang | math functions), 1076
Single target,  
    multiple sources, 333
skip-first-items,  
    as MapForce function (in core | sequence functions), 1042
SOAP,  
    creating a Web Service in MapForce, 1114
    defining a fault, 1122
    generating a request, 1114
    Java specifics, 1124
    support notes, 1111
SOAP message,  
    adding a UsernameToken to, 1175
    setting the TTL (time-to-live), 1175
Software product license, 1833
Sort,  
    column icon in Results window, 558
    data in result window, 558
    sort component, 200
    tables Database Query, 554
Sort key,  
    sort component, 200
Sort order,  
changing, 200
Sorting,  
    in databases, 524
Source component,  
    definition of, 1845
Source-driven,  
    - mixed content mapping, 139
Source-driven connections,  
    as opposed to standard (target-driven) connections, 146
SPL, 1475
    code blocks, 1475
    conditions, 1482
    foreach, 1483
    global objects, 1479
    subroutines, 1485
    using files, 1480
    variables, 1478
Split once,  
    as FlexText function, 662
    using the "delimited (floating)" mode, 664
    using the "delimited (line based)" mode, 665
    using the "delimited (line starts with)" mode, 666
    using the "fixed length" mode, 663
SPS,  
    predefined stylesheets for documenting mappings, 1274
    user-defined stylesheets, 1277
SQL, 550
    executing statements, 549, 551
    exporting statements as SQL scripts, 551
    generating statements, 549, 550
    importing SQL scripts, 551
    load from scripts, 549
    writing statements, 550
SQL Editor,  
    bookmark margin, 552
    commenting out text, 551
    creating regions, 553
    inserting bookmarks, 552
    inserting comments, 551
    inserting regions, 553
    removing bookmarks, 552
    removing comments, 551
    removing regions, 553
    settings - general, 561
    using bookmarks, 552
    using regions, 553
SQL Server,  
    connecting through ADO, 355
© 2019 Altova GmbH
SQL Server,
  connecting through ADO.NET, 360
  reading from XML type fields, 537
  writing to XML type fields, 537

SQL WHERE,
  component - insert, 521
  ORDER BY, 521

SQL WHERE/ORDER,
  as MapForce component, 524

SQLite,
  changing database path to absolute in generated code, 137
  mapping data from, 244
  mapping data to, 249, 501
  using an absolute or relative path, 134
  writing XML files to, 540

sqrt,
  as MapForce function (in lang | math functions), 1076

Standard,
  mapping method, 139

starts-with,
  as MapForce function (in core | string functions), 1045

static-node-annotation,
  as MapForce function (in core | node functions), 1020

static-node-name,
  as MapForce function (in core | node functions), 1020

Store as CSV (delimited),
  as FlexText function, 674

Store as FLF (fixed length),
  as FlexText function, 681

Store value,
  as FlexText function, 684

Stored procedures,
  adding to a mapping, 568
  calling from a mapping, 565, 570, 573, 577, 582, 585
  support notes, 565

string,
  as MapForce function (in core | conversion functions), 1006
  as MapForce function (in xpath2 | accessors library), 1092
  parsing data from, 242, 244
  serializing data to, 242, 249

string-as-QName,
  as MapForce function (in lang | QName functions), 1056

string-compare,
  as MapForce function (in lang | string functions), 1085

string-compare-ignore-case,
  as MapForce function (in lang | string functions), 1085

string-join,
  as MapForce function (in core | aggregate functions), 994

string-length,
  as MapForce function (in core | string functions), 1045

Stylesheets,
  defining for documentation, 1277

Stylevision,
  defining SPS stylesheets for mappings, 1277

substitute-missing,
  as MapForce function (in core | sequence functions), 1042

substitute-missing-with-xsi-nil,
  as MapForce function (in core | node functions), 1020

substitute-null,
  as MapForce function (in core | node functions), 1020

substring,
  as MapForce function (in core | string functions), 1045

substring-after,
  as MapForce function (in core | string functions), 1045

substring-before,
  as MapForce function (in core | string functions), 1046

subtract,
  as MapForce function (in core | math functions), 1018

sum,
  as MapForce function (in core | aggregate functions), 995

Switch,
  as FlexText function, 668

Sybase,
  connecting through JDBC, 417

System DSN,
  setting up, 367

system-property,
  as MapForce function (in xslt | xslt functions library), 1106

T

Table,
  lookup - value map, 230

Table data,
  sorting, 200

tan,
  as MapForce function (in lang | math functions), 1077

Target,
  FlexText component, 653

Target component,
  definition of, 1846

Target-driven connections,
  as opposed to source-driven connections, 146

Target-driven mapping, 139
Technical Information, 1828
Teradata,
  connect through JDBC, 422
  connect through ODBC, 418
Terminology,
  EDIFACT, 694
Text,
  files - defining key fields, 616
  mapping text files, 611
Text files,
  adding or removing fields in,, 629
  as source component, 629
  as target component, 629
  mapping data from, 623
  previewing data from,, 629
  setting the encoding of,, 629
  setting the fill character, 623
  setting the fixed field size, 623

time-from-datetime,
  as MapForce function (in lang | datetime functions), 1070

time-to-xlsx,
  as MapForce function, 1091
timezone,
  as MapForce function (in lang | datetime functions), 1070
to-date,
  as MapForce function (in edifact functions), 1054
to-datetime,
  as MapForce function (in edifact functions), 1054
to-duration,
  as MapForce function (in edifact functions), 1056
tokenize,
  as MapForce function (in core | string functions), 1046

tokenize-by-length,
  as MapForce function (in core | string functions), 1048
tokenize-regexp,
  as MapForce function (in core | string functions), 1050
Tomcat,
  configuring for SOAP Web services, 1124, 1141
Tools,
  as application menu, 1345
to-time,
  as MapForce function (in edifact functions), 1056
TRADACOMS, 692
  as mapping component, 726, 732
  configuration files, 736
  converting to XML, 737
  validation rules, 733
Transform,
User-defined functions,
  importing into a mapping, 946
  inline versus regular, 942
  navigating, 944
  overview, 932

Web services,
  adding a UsernameToken, 1175
  adding REST-style calls, 1178
  as MapForce components, 1150
  calling from MapForce, 1150
  calling through HTTPS, 1192
  configuring the time-out period, 1170
  enabling preemptive authentication, 1173
  enabling WS-Security, 1175
  example (REST-style), 1178
  example (WSDL-style), 1188
  mapping from RSS feed, 1183
  mapping to JSON, 1178
  signing with HTTPS digital certificates, 1173
  supplying basic HTTP authentication credentials, 1173

Web services (REST),
  adding to the mapping, 1152

Web services (SOAP),
  generating with MapForce, 1108, 1127, 1141
  implementing, 1108
  prerequisites, 1108

Web services (WSDL),
  adding to the mapping, 1169

WebDAV Server,
  adding files as components from, 74

weekday,
  as MapForce function (in lang | datetime functions), 1070

weeknumber,
  as MapForce function (in lang | datetime functions), 1071

WHERE,
  SQL WHERE component, 521

Wildcards,
  xs:any - xs:anyAttribute, 329

Windows,
  executing mappings with Web service calls through HTTPS, 1208
  support for Altova products, 1828
  trusting server certificates on, 1197

Word,
  mapping documentation, 1268

Word 2007+,
  preview mapping output as, 1263

Worksheet,
  to XML output files, 171

Wrapper classes,
  in generated code, 1473

WSDL,
  calling a Web service from a mapping, 1188
WSDL,
creating a SOAP Web Service from, 1114
support notes, 1111
validating, 1114

X

X12,
generating the X12 997 Functional Acknowledgement, 708
generating the X12 999 Implementation Acknowledgement, 709
merged entries, 764
XBRL taxonomy packages,
adding, 842
xbrl-measure-currency,
as MapForce function, 1090
xbrl-measure-pure,
as MapForce function, 1090
xbrl-measure-shares,
as MapForce function, 1090
Xerces,
generating code for, 1473
supported versions, 1355
xlsx-to-date,
as MapForce function, 1091
xlsx-to-datetime,
as MapForce function, 1092
xlsx-to-time,
as MapForce function, 1092
XML,
as mapping target, 611
converting from TRADACOMS, 737
mapping data from CSV to, 611
mapping from Excel 2007+ to, 829
mapping from UN/EDIFACT, 745
signature settings, 341
writing to database field, 540
XML data,
reading from database fields, 537
writing to database fields, 537
XML declaration,
suppressing from output, 316
XML files,
generate from database records, 169
generate from Excel, 171
generate from single XML source, 167
XML output,
changing encoding settings, 316
changing instance file name, 316
changing schema, 316
creating digital signature, 316
XML Parser,
about, 1828
XML to XML, 315
XMLSpy,
as SOAP client, 1114
XMLSpy command table, 1703
XQuery,
adding custom functions, 962
Extension functions, 1809
importing modules, 962
previewing generated code, 93
xs: any (xs:anyAttribute), 329
xs: nil,
as attribute in XML instance, 323
xs: type,
mapping to derived types, 321
XSLT,
adding custom functions, 956
Extension functions, 1809
previewing the generated code, 92
removing custom functions, 956
template namespace, 956
Y

year-from-datetime,
as MapForce function (in lang | datetime functions), 1071
year-from-duration,
as MapForce function (in lang | datetime functions), 1071
Z

Z to A,
sort component, 200
zip64mode,
enabling in the build.xml file, 1472