Table of contents

Altova XMLSpy 2019 Professional Edition 2

XMLSpy Tutorial 4

1 XMLSpy Interface 5
1.1 The Views ................................................................. 7
1.2 The Windows .............................................................. 9
1.3 Menus and Toolbars ..................................................... 11
1.4 Text View Settings ..................................................... 13

2 XML Schemas: Basics 17
2.1 Creating a New XML Schema File .................................. 18
2.2 Defining Namespaces ................................................... 20
2.3 Defining a Content Model ............................................. 22
2.4 Adding Elements with Drag-and-Drop .......................... 27
2.5 Configuring the Content Model View ............................ 29
2.6 Completing the Basic Schema ..................................... 31

3 XML Schemas: Advanced 34
3.1 Working with Complex Types and Simple Types ............ 35
3.2 Referencing Global Elements ....................................... 43
3.3 Attributes and Attribute Enumerations ........................ 46

4 XML Schemas: XMLSpy Features 49
4.1 Schema Navigation .................................................... 50
4.2 Schema Documentation ............................................. 53
2 General Information 96
2.1 Windows File Paths ................................................................. 97
2.2 About RaptorXML Server ..................................................... 98

3 Interface and Environment 99
3.1 The Graphical User Interface (GUI) ........................................... 100
  3.1.1 Main Window ................................................................. 101
  3.1.2 Project Window ............................................................. 102
  3.1.3 Info Window ................................................................. 104
  3.1.4 Entry Helpers ............................................................... 104
  3.1.5 Output Window: Messages .............................................. 105
  3.1.6 Output Window: XPath/XQuery ....................................... 107
    – Evaluate Mode............................................................... 108
    – Debug Mode ................................................................ 113
    – XPath and XQuery Specification Information .................... 118
  3.1.7 Output Window: XSL Outline .......................................... 119
  3.1.8 Output Window: HTTP .................................................. 120
  3.1.9 Output Window: Find in Files ......................................... 121
  3.1.10 Output Window: Find in Schemas .................................. 122
  3.1.11 Menu Bar, Toolbars, Status Bar .................................... 123
3.2 The Application Environment .................................................. 125
  3.2.1 Settings and Customization .............................................. 125
  3.2.2 Tutorials, Projects, Examples .......................................... 127
  3.2.3 XMLSpy Features and Help, and Altova Products ............... 128

4 Editing Views 130
4.1 Text View ........................................................................... 131
  4.1.1 Formatting in Text View ................................................ 132
  4.1.2 Displaying the Document .............................................. 133
  4.1.3 Editing in Text View ...................................................... 136
  4.1.4 Navigating the Document .............................................. 139
  4.1.5 Find and Replace .......................................................... 141
  4.1.6 Validating an XML Document ........................................ 148
  4.1.7 Entry Helpers in Text View ............................................ 149
  4.1.8 Split View ..................................................................... 150
  4.1.9 Text View Shortcuts ...................................................... 152
4.2 Grid View ........................................................................... 155
  4.2.1 Editing in Grid View ...................................................... 156
  4.2.2 Grid View Tables ......................................................... 158
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.3</td>
<td>Entry Helpers in Grid View</td>
<td>161</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Grid View Shortcuts</td>
<td>162</td>
</tr>
<tr>
<td>4.3</td>
<td>Schema View</td>
<td>165</td>
</tr>
<tr>
<td>4.3.1</td>
<td>XSD Mode: XSD 1.0 or 1.1</td>
<td>167</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Schema Overview</td>
<td>171</td>
</tr>
<tr>
<td></td>
<td>GUI Mechanisms</td>
<td>172</td>
</tr>
<tr>
<td></td>
<td>Global Components</td>
<td>175</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Content Model View</td>
<td>183</td>
</tr>
<tr>
<td></td>
<td>Content Model Objects</td>
<td>185</td>
</tr>
<tr>
<td></td>
<td>Editing in Content Model View</td>
<td>191</td>
</tr>
<tr>
<td></td>
<td>Conditional Type Assignment</td>
<td>197</td>
</tr>
<tr>
<td></td>
<td>Open Content Models</td>
<td>201</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Attributes, Assertions, and Identity Constraints</td>
<td>204</td>
</tr>
<tr>
<td></td>
<td>Attributes, Attribute Groups, Attribute Wildcards</td>
<td>205</td>
</tr>
<tr>
<td></td>
<td>Assertions</td>
<td>208</td>
</tr>
<tr>
<td></td>
<td>Identity Constraints</td>
<td>212</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Entry Helpers in Schema View</td>
<td>219</td>
</tr>
<tr>
<td></td>
<td>Components</td>
<td>220</td>
</tr>
<tr>
<td></td>
<td>Details</td>
<td>223</td>
</tr>
<tr>
<td></td>
<td>Facets</td>
<td>225</td>
</tr>
<tr>
<td>4.3.6</td>
<td>Validation and Smart Fixes</td>
<td>229</td>
</tr>
<tr>
<td>4.3.7</td>
<td>Assertion Messages</td>
<td>230</td>
</tr>
<tr>
<td>4.3.8</td>
<td>Base Type Modification</td>
<td>233</td>
</tr>
<tr>
<td>4.3.9</td>
<td>Smart Restrictions</td>
<td>234</td>
</tr>
<tr>
<td>4.3.10</td>
<td>xml:base, xml:lang, xml:id, xml:space</td>
<td>239</td>
</tr>
<tr>
<td>4.3.11</td>
<td>Back and Forward: Moving through Positions</td>
<td>240</td>
</tr>
<tr>
<td>4.4</td>
<td>Authentic View</td>
<td>242</td>
</tr>
<tr>
<td>4.5</td>
<td>Browser View</td>
<td>243</td>
</tr>
<tr>
<td>4.6</td>
<td>Archive View</td>
<td>244</td>
</tr>
<tr>
<td>5</td>
<td>XML</td>
<td>246</td>
</tr>
<tr>
<td>5.1</td>
<td>Creating, Opening, and Saving XML Documents</td>
<td>247</td>
</tr>
<tr>
<td>5.2</td>
<td>Assigning Schemas and Validating</td>
<td>249</td>
</tr>
<tr>
<td>5.3</td>
<td>Editing XML in Text View</td>
<td>251</td>
</tr>
<tr>
<td>5.4</td>
<td>Editing XML in Grid View</td>
<td>253</td>
</tr>
<tr>
<td>5.5</td>
<td>Editing XML in Authentic View</td>
<td>256</td>
</tr>
<tr>
<td>5.6</td>
<td>Entry Helpers for XML Documents</td>
<td>258</td>
</tr>
<tr>
<td>5.7</td>
<td>Processing with XSLT and XQuery</td>
<td>260</td>
</tr>
<tr>
<td>5.8</td>
<td>PDF Fonts</td>
<td>262</td>
</tr>
<tr>
<td>5.9</td>
<td>Additional Features</td>
<td>265</td>
</tr>
</tbody>
</table>
6 DTDs and XML Schemas

6.1 DTDs ................................................................. 268
6.2 XML Schemas .................................................. 270
6.3 Schema Subsets ............................................... 271
6.4 Catalogs in XMLSpy .......................................... 275
6.5 Working with SchemaAgent ............................... 280
   6.5.1 Connecting to SchemaAgent Server .............. 281
   6.5.2 Opening Schemas Found in the Search Path ..... 283
   6.5.3 Using IIRs .................................................. 284
   6.5.4 Viewing Schemas in SchemaAgent ............... 289
   6.5.5 SchemaAgent Validation ............................. 289
6.6 Find in Schemas ................................................ 291
   6.6.1 Search Term .............................................. 293
   6.6.2 Components ............................................. 294
   6.6.3 Properties ................................................ 296
   6.6.4 Scope ...................................................... 299
   6.6.5 Find and Replace Commands ..................... 300
   6.6.6 Results and Information ............................. 302
   6.6.7 Finding and Renaming Globals ................... 303

7 XSLT

7.1 XSLT Documents .............................................. 306
7.2 XSLT Processing .............................................. 308
7.3 XSL Outline .................................................... 310
   7.3.1 XSL Outline Window ............................... 311
   7.3.2 Info Window .......................................... 314

8 XQuery

8.1 Editing XQuery Documents ................................. 319
   8.1.1 XQuery Documents .................................. 320
   8.1.2 XQuery Entry Helpers .............................. 321
   8.1.3 XQuery Syntax Coloring .............................. 321
   8.1.4 XQuery Intelligent Editing ......................... 323
8.2 XQuery Evaluation .......................................... 326
8.3 XQuery Validation ........................................... 327
8.4 XQuery/Update Execution ................................. 328
8.5 XQuery Update Facility .................................. 330
8.5.1 Previewing and Applying Updates.................................................................330
8.5.2 Update Operations and Syntax.................................................................333
   – Delete Nodes.........................................................................................333
   – Insert Nodes.........................................................................................333
   – Rename Node.........................................................................................334
   – Replace Node.........................................................................................334
   – Replace Value of Node...........................................................................335
   – The fn:put Function..............................................................................335
8.6 XQuery and XML Databases ........................................................................336

9 XSLT/XQuery Debugger and Profiler .............................................................340
9.1 XSLT and XQuery Debugger ........................................................................341
   9.1.1 Mechanism and Interface....................................................................342
   9.1.2 Commands and Toolbar Icons..........................................................344
   9.1.3 Debugger Settings.............................................................................347
   9.1.4 Starting a Debugging Session.............................................................348
   9.1.5 Information Windows..........................................................................349
      – Context Window..................................................................................351
      – Variables Window................................................................................351
      – XPath-Watch Window..........................................................................352
      – Call Stack Window..............................................................................352
      – Messages Window...............................................................................353
      – Templates Window...............................................................................353
      – Info Window.......................................................................................354
      – Trace Window.....................................................................................354
      – Arranging the Info Windows ................................................................355
   9.1.6 Breakpoints ......................................................................................356
   9.1.7 Tracepoints ......................................................................................359
   9.1.8 Debugger Shortcuts ............................................................................364

10 Authentic ......................................................................................................366
10.1 Authentic View Tutorial ...........................................................................368
   10.1.1 Opening an XML Document in Authentic View...............................369
   10.1.2 The Authentic View Interface............................................................370
   10.1.3 Node Operations .............................................................................373
   10.1.4 Entering Data in Authentic View......................................................376
   10.1.5 Entering Attribute Values.................................................................378
   10.1.6 Adding Entities ...............................................................................378
   10.1.7 Printing the Document....................................................................379
10.2 Authentic View Interface ..........................................................................381
   10.2.1 Overview of the GUI..........................................................................381
   10.2.2 Authentic View Toolbar Icons..........................................................382
10.3 Editing in Authentic View ................................................................. 395
  10.3.1 Basic Editing ............................................................................. 395
  10.3.2 Tables in Authentic View .......................................................... 400
    – SPS Tables .................................................................................. 400
    – CALS/HTML Tables ................................................................. 401
    – CALS/HTML Table Editing Icons ............................................ 405
  10.3.3 Editing a DB .............................................................................. 408
    – Navigating a DB Table ............................................................. 408
    – DB Queries .............................................................................. 409
    – Modifying a DB Table .............................................................. 413
  10.3.4 Working with Dates ................................................................. 414
    – Date Picker ............................................................................. 415
    – Text Entry ............................................................................... 416
  10.3.5 Defining Entities ..................................................................... 416
  10.3.6 XML Signatures ..................................................................... 418
  10.3.7 Images in Authentic View ...................................................... 419
  10.3.8 Keystrokes in Authentic View ................................................ 420
  10.4 Authentic Scripting ...................................................................... 421

11 HTML and CSS .............................................................................. 423
  11.1 HTML .......................................................................................... 424
  11.2 CSS ............................................................................................. 426

12 JSON, JSON Schema ........................................................................ 431
  12.1 JSON Data ................................................................................. 433
  12.2 JSON Schema ............................................................................ 436
  12.3 JSON Documents in Text View ................................................... 438
  12.4 JSON Documents in Grid View .................................................. 443
  12.5 Validating JSON Documents ...................................................... 447
  12.6 JSON Transformations with XSLT/XQuery .................................. 449
  12.7 Generating JSON Schema from a JSON Instance ....................... 454
  12.8 Generating a JSON Instance from a JSON Schema .................... 457
  12.9 Converting between JSON and XML ........................................ 458
  12.10 JSON Schema View ................................................................... 459
    12.10.1 JSON Schema Version ....................................................... 460
    12.10.2 Adding Global Definitions ................................................. 461
12.10.3 Entry Helpers: Overview, Details, Constraints .................................................................463
12.10.4 Global and Local Definitions ............................................................................................466
12.10.5 Design View ....................................................................................................................467
12.10.6 Objects and Properties ....................................................................................................469
12.10.7 Unspecified Properties ....................................................................................................472
12.10.8 Objects and Dependencies ..............................................................................................475
12.10.9 Arrays ...............................................................................................................................478
12.10.10 Atomic Types ................................................................................................................480
12.10.11 Type Selectors (Any, Multiple, etc) ..............................................................................481
12.10.12 Operators .....................................................................................................................483
12.10.13 Conditionals ................................................................................................................485
12.10.14 Configuring Design View ..............................................................................................486
12.10.15 Generating JSON Schema Documentation ..................................................................488

13 HTTP .........................................................................................................................................490
13.1 Sending the Request ............................................................................................................491
13.2 Importing a Request to Send ...............................................................................................496
13.3 Receiving the Response ........................................................................................................498

14 Office Open XML, ZIP, EPUB .................................................................................................501
14.1 Working with OOXML Files .................................................................................................503
14.2 OOXML Example Files .........................................................................................................505
14.3 ZIP Files ................................................................................................................................507
14.4 EPUB Files ..........................................................................................................................509

15 Databases ..................................................................................................................................513
15.1 Connecting to a Database .......................................................................................................515
15.1.1 Starting the Database Connection Wizard .......................................................................516
15.1.2 Database Drivers Overview .............................................................................................518
15.1.3 Setting up an ADO Connection ........................................................................................521
  – Connecting to an Existing Microsoft Access Database ....................................................523
  – Creating a New Microsoft Access Database .......................................................................523
  – Setting up the SQL Server Data Link Properties ...............................................................524
  – Setting up the Microsoft Access Data Link Properties .....................................................525
15.1.4 Setting up an ADO.NET Connection ..............................................................................527
  – Creating a Connection String in Visual Studio ...................................................................529
  – Sample ADO.NET Connection Strings ..............................................................................532
  – ADO.NET Support Notes ....................................................................................................533
15.1.5 Setting up an ODBC Connection ....................................................................................534
- Viewing the Available ODBC Drivers
- Configuring the CLASSPATH

15.1.6 Setting up a JDBC Connection
- Connecting to an Existing SQLite Database
- Creating a New SQLite Database
- Foreign Key Constraints

15.1.7 Setting up a PostgreSQL Connection
15.1.8 Setting up a SQLite Connection

15.1.9 Using a Connection from Global Resources

15.1.10 Database Connection Examples
- Connecting to Firebird (ODBC)
- Connecting to Firebird (JDBC)
- Connecting to IBM DB2 (ODBC)
- Connecting to IBM DB2 for i (ODBC)
- Connecting to IBM Informix (JDBC)
- Connecting to MariaDB (ODBC)
- Connecting to Microsoft Access (ADO)
- Connecting to Microsoft SQL Server (ADO)
- Connecting to Microsoft SQL Server (ODBC)
- Connecting to MySQL (ODBC)
- Connecting to Oracle (ODBC)
- Connecting to Oracle (JDBC)
- Connecting to PostgreSQL (ODBC)
- Connecting to Progress OpenEdge (ODBC)
- Connecting to Progress OpenEdge (JDBC)
- Connecting to Sybase (JDBC)
- Connecting to Teradata (ODBC)
- Connecting to Teradata (JDBC)

15.2 Supported Databases

---

16 Altova Global Resources 593

16.1 Defining Global Resources
16.1.1 Files
16.1.2 Folders
16.1.3 Databases

16.2 Using Global Resources
16.2.1 Assigning Files and Folders
16.2.2 Assigning Databases
16.2.3 Changing the Active Configuration

---

17 Projects 611

17.1 Creating and Editing Projects
18 RaptorXML Server

18.1 Adding Servers and Server Configurations .......................................................... 619
18.2 Validating with RaptorXML Server ..................................................................... 623
18.3 Validation Options ............................................................................................... 624
   18.3.1 Common Options ......................................................................................... 624
   18.3.2 XML with DTD .......................................................................................... 625
   18.3.3 DTD ........................................................................................................... 625
   18.3.4 XML with W3C Schema ............................................................................ 626
   18.3.5 W3C Schema ........................................................................................... 626
   18.3.6 XBRL Instance ......................................................................................... 627
   18.3.7 XBRL Taxonomy ....................................................................................... 628
   18.3.8 XSLT ......................................................................................................... 629
   18.3.9 XQuery ..................................................................................................... 630
   18.3.10 JSON ...................................................................................................... 631
   18.3.11 JSON Schema ................................................................................-------- 631
   18.3.12 EDGAR .................................................................................................... 632
18.4 XSLT and XQuery with RaptorXML Server ..................................................... 634

19 File/Directory Comparisons

19.1 File Comparisons ............................................................................................... 637
19.2 Directory Comparisons ...................................................................................... 638

20 Source Control

20.1 Setting Up Source Control .................................................................................. 642
20.2 Supported Source Control Systems .................................................................... 643
20.3 Local Workspace Folder .................................................................................... 645
20.4 Application Project ............................................................................................. 646
20.5 Add to Source Control ....................................................................................... 648
20.6 Working with Source Control ............................................................................ 650
   20.6.1 Add to, Remove from Source Control ....................................................... 650
   20.6.2 Check Out, Check In ................................................................................ 651
   20.6.3 Getting Files as Read-Only ....................................................................... 653
   20.6.4 Copying and Sharing from Source Control ............................................. 655
   20.6.5 Changing Source Control ....................................................................... 658
20.7 Source Control with Git ..................................................................................... 660
   20.7.1 Enabling Git Source Control with GIT SCC Plug-in .................................. 661
21 XMLSpy in Visual Studio 665
21.1 Installing the XMLSpy Plugin .................................................................666
21.2 Differences with XMLSpy Standalone ..................................................668
21.3 XMLSpy's Debuggers in Visual Studio ....................................................670

22 XMLSpy in Eclipse 671
22.1 Installing the XMLSpy Plugin for Eclipse ...............................................672
22.2 XMLSpy Entry Points in Eclipse ..............................................................678
22.3 XMLSpy's Debugger Perspectives ...........................................................682

23 Menu Commands 683
23.1 File Menu ..................................................................................................684
  23.1.1 New ......................................................................................................684
  23.1.2 Open .....................................................................................................688
  23.1.3 Reload ..................................................................................................693
  23.1.4 Encoding ..............................................................................................693
  23.1.5 Close, Close All, Close All But Active ................................................694
  23.1.6 Save, Save As, Save All .......................................................................694
  23.1.7 Send by Mail ........................................................................................699
  23.1.8 Print ......................................................................................................700
  23.1.9 Print Preview, Print Setup ...................................................................702
  23.1.10 Recent Files, Exit ...............................................................................703
23.2 Edit Menu ..................................................................................................704
  23.2.1 Undo, Redo ..........................................................................................705
  23.2.2 Cut, Copy, Paste, Delete .....................................................................705
  23.2.3 Copy as XML Text ..............................................................................706
  23.2.4 Copy as Structured Text .....................................................................706
  23.2.5 Copy XPath ........................................................................................710
  23.2.6 Copy XPointer .....................................................................................710
  23.2.7 Insert ....................................................................................................710
  23.2.8 Pretty-Print ..........................................................................................714
  23.2.9 Strip Whitespaces ...............................................................................715
  23.2.10 Select All ............................................................................................715
  23.2.11 Find, Find Next ..................................................................................715
  23.2.12 Replace ...............................................................................................718
  23.2.13 Find in Files .......................................................................................719
23.3.1 New Project ................................................................. 725
23.3.2 Open Project ............................................................. 726
23.3.3 Reload Project ......................................................... 726
23.3.4 Close Project .........................................................   726
23.3.5 Save Project, Save Project As .................................. 726
23.3.6 Source Control .......................................................... 726
   – Open from Source Control ........................................ 727
   – Enable Source Control ............................................ 728
   – Get Latest Version ................................................ 729
   – Get, Get Folders .................................................... 729
   – Check Out, Check In .............................................. 730
   – Undo Check Out ................................................... 732
   – Add to Source Control .......................................... 733
   – Remove from Source Control ................................ 734
   – Share from Source Control .................................. 734
   – Show History ....................................................... 736
   – Show Differences ................................................. 737
   – Show Properties ................................................ 738
   – Refresh Status ..................................................... 739
   – Source Control Manager ..................................... 739
   – Change Source Control ....................................... 739
23.3.7 Add Files to Project ................................................... 740
23.3.8 Add Global Resource to Project ................................. 740
23.3.9 Add URL to Project ................................................ 741
23.3.10 Add Active File to Project ....................................... 741
23.3.11 Add Active And Related Files to Project .................... 741
23.3.12 Add Project Folder to Project .................................. 741
23.3.13 Add External Folder to Project ................................ 742
23.3.14 Add External Web Folder to Project ......................... 744
23.3.15 Script Settings ...................................................... 748
23.3.16 Properties .............................................................. 749
23.3.17 Most Recently Used Projects ................................. 751

23.4 Insert ............................................................................. 752
   – Insert Attribute ..................................................... 753
   – Insert Element ..................................................... 753
   – Insert Text .......................................................... 754
   – Insert CDATA ..................................................... 754
   – Insert Comment .................................................. 754
   – Insert XML ........................................................ 754
   – Insert Processing Instruction ............................... 755
   – Insert XInclude .................................................. 755
<table>
<thead>
<tr>
<th>23.4.2</th>
<th>Append</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Append Attribute</td>
<td>761</td>
</tr>
<tr>
<td>- Append Element</td>
<td>761</td>
</tr>
<tr>
<td>- Append Text</td>
<td>762</td>
</tr>
<tr>
<td>- Append CDATA</td>
<td>762</td>
</tr>
<tr>
<td>- Append Comment</td>
<td>762</td>
</tr>
<tr>
<td>- Append XML</td>
<td>762</td>
</tr>
<tr>
<td>- Append Processing Instruction</td>
<td>763</td>
</tr>
<tr>
<td>- Append XInclude</td>
<td>763</td>
</tr>
<tr>
<td>- Append DOCTYPE</td>
<td>765</td>
</tr>
<tr>
<td>- Append ExternalID</td>
<td>766</td>
</tr>
<tr>
<td>- Append ELEMENT</td>
<td>766</td>
</tr>
<tr>
<td>- Append ATTLIST</td>
<td>767</td>
</tr>
<tr>
<td>- Append ENTITY</td>
<td>767</td>
</tr>
<tr>
<td>- Append NOTATION</td>
<td>767</td>
</tr>
<tr>
<td>- Append Encoded External File</td>
<td>767</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>23.4.3</th>
<th>Add Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Attribute</td>
<td>769</td>
</tr>
<tr>
<td>- Element</td>
<td>769</td>
</tr>
<tr>
<td>- Text</td>
<td>769</td>
</tr>
<tr>
<td>- CDATA</td>
<td>769</td>
</tr>
<tr>
<td>- Comment</td>
<td>769</td>
</tr>
<tr>
<td>- XML</td>
<td>770</td>
</tr>
<tr>
<td>- Processing Instruction</td>
<td>770</td>
</tr>
<tr>
<td>- XInclude</td>
<td>770</td>
</tr>
<tr>
<td>- DOCTYPE</td>
<td>773</td>
</tr>
<tr>
<td>- ExternalID</td>
<td>773</td>
</tr>
<tr>
<td>- ELEMENT</td>
<td>773</td>
</tr>
<tr>
<td>- ATTLIST</td>
<td>774</td>
</tr>
<tr>
<td>- ENTITY</td>
<td>774</td>
</tr>
<tr>
<td>- NOTATION</td>
<td>774</td>
</tr>
<tr>
<td>- Encoded External File</td>
<td>774</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>23.4.4</th>
<th>Convert To</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Attribute</td>
<td>775</td>
</tr>
<tr>
<td>- Element</td>
<td>775</td>
</tr>
<tr>
<td>- Text</td>
<td>775</td>
</tr>
<tr>
<td>- CDATA</td>
<td>776</td>
</tr>
<tr>
<td>- Comment</td>
<td>776</td>
</tr>
</tbody>
</table>
23.4.5 Table
- Display as Table ................................................................. 777
- Insert Row ........................................................................... 778
- Append Row .......................................................................... 778
- Ascending Sort .................................................................... 779
- Descending Sort .................................................................... 779

23.4.6 Move Left ........................................................................ 780
23.4.7 Move Right ...................................................................... 780
23.4.8 Enclose in Element ............................................................. 780
23.4.9 Evaluate XPath ................................................................ 781
23.4.10 Check Well-Formedness ................................................... 781
23.4.11 Validate XML ................................................................. 783
23.4.12 Validate XML on Server (high-performance) ................... 787
23.4.13 Update Entry Helpers .................................................... 788
23.4.14 Namespace Prefix ........................................................... 788

23.5 DTD/Schema Menu ............................................................... 790
23.5.1 Assign DTD ..................................................................... 790
23.5.2 Assign Schema ................................................................. 791
23.5.3 Include Another DTD ...................................................... 791
23.5.4 Go to DTD ....................................................................... 791
23.5.5 Go to Schema ................................................................. 792
23.5.6 Go to Definition ............................................................... 792
23.5.7 Generate DTD/Schema ..................................................... 792
23.5.8 Flatten DTD .................................................................... 794
23.5.9 Convert DTD to Schema ................................................... 794
23.5.10 Flatten Schema ............................................................... 796
23.5.11 Convert Schema to DTD .................................................. 796
23.5.12 Convert to UML ................................................................. 797
23.5.13 Generate XML from DB, Excel, EDI with MapForce ........ 799
23.5.14 Design HTML/PDF/Word Output with StyleVision ........ 799
23.5.15 Generate Sample XML/JSON File .................................... 799
23.5.16 Flush Memory Cache ..................................................... 803

23.6 Schema Design Menu ......................................................... 804
23.6.1 Schema Settings ............................................................... 804
23.6.2 Save Diagram ................................................................. 807
23.6.3 Generate Documentation ................................................ 807
- Documentation Options ......................................................... 809
### 23.7 XSL/XQuery Menu

- **XSL Transformation** .......................................................................................... 827
- **XSL-FO Transformation** ...................................................................................... 827
- **XSL Parameters / XQuery Variables** ................................................................... 829
- **XQuery/Update Execution** .................................................................................... 832
- **Assign XSL** ........................................................................................................... 833
- **Assign XSL-FO** .................................................................................................... 833
- **Assign Sample XML File** ..................................................................................... 833
- **Go to XSL** ............................................................................................................. 834
- **Start Debugger / Go** ............................................................................................ 834
- **Stop Debugger** ...................................................................................................... 834
- **Restart Debugger** ................................................................................................ 835
- **End Debugger Session** ........................................................................................ 835
- **Step Into** ............................................................................................................... 835
- **Step Out** ................................................................................................................. 835
- **Step Over** ............................................................................................................... 835
- **Show Current Execution Node** ............................................................................. 836
- **Insert/Remove Breakpoint** .................................................................................. 836
- **Insert/Remove Tracepoint** .................................................................................... 836
- **Enable/Disable Breakpoint** .................................................................................. 836
- **Enable/Disable Tracepoint** .................................................................................. 837
- **Breakpoints/Tracepoints** ...................................................................................... 837
- **Debug Windows** .................................................................................................... 838
- **Debug Settings** ...................................................................................................... 838

### 23.8 Authentic Menu

- **New Document** ..................................................................................................... 839
- **Edit Database Data** ................................................................................................. 840
- **Assign a StyleVision Stylesheet** .......................................................................... 841
23.8.4 Edit StyleVision Stylesheet ...............................................................842
23.8.5 Select New Row with XML Data for Editing .................................842
23.8.6 XML Signature ..............................................................................843
23.8.7 Define XML Entities .................................................................844
23.8.8 View Markup .................................................................................846
23.8.9 RichEdit .......................................................................................847
23.8.10 Append/Insert/Duplicate/Delete Row ........................................847
23.8.11 Move Row Up/Down ..................................................................848
23.8.12 Generate HTML, RTF, PDF, Word 2007+ Document ................848
23.8.13 Trusted Locations ......................................................................848

23.9 DB Menu .......................................................................................850

23.9.1 Query Database ...........................................................................850
  – Data Sources ..................................................................................852
  – Browser Pane: Viewing the DB Objects ........................................855
  – Query Pane: Description and Features .........................................858
  – Query Pane: Working with Queries .............................................861
  – Results and Messages ...............................................................862
23.9.2 IBM DB2 ...................................................................................864
  – Manage XML Schemas ..............................................................865
  – Assign XML Schema ..................................................................868
23.9.3 SQL Server ................................................................................869
  – Manage XML Schemas ..............................................................870
23.9.4 Oracle XML DB ..........................................................................872
  – Manage XML Schemas ..............................................................872
  – Browse Oracle XML Documents ...............................................875

23.10 Convert Menu ..............................................................................877

23.10.1 Import Text File .........................................................................877
23.10.2 Import Database Data ..............................................................880
23.10.3 Import Microsoft Word Document ..........................................885
23.10.4 Create XML Schema from DB Structure ................................885
23.10.5 DB Import Based on XML Schema ..........................................889
23.10.6 Create DB Structure from XML Schema ................................890
23.10.7 Export to Text Files ..................................................................893
23.10.8 Export to a Database ..............................................................896
23.10.9 Convert XML Instance to/from JSON ....................................899
23.10.10 Convert XML Schema to/from JSON Schema ......................902

23.11 View Menu ..................................................................................904

23.11.1 Text View ..................................................................................904
23.11.2 Enhanced Grid View ..............................................................904
23.11.3 Schema Design View .............................................................905
23.11.4 Authentic View .......................................................................905
23.11.5 Browser View ..........................................................................906
23.11.6 Expand .....................................................................................906
23.11.7 Collapse ..................................................................................906
23.11.8 Expand Fully ...............................................................906
23.11.9 Collapse Unselected .....................................................907
23.11.10 Optimal Widths ..........................................................907
23.11.11 Word Wrap ...............................................................907
23.11.12 Go to Line/Character ..................................................907
23.11.13 Go to File .................................................................908
23.11.14 Text View Settings .....................................................908

23.12 Browser Menu ..............................................................911
23.12.1 Back ...........................................................................911
23.12.2 Forward .......................................................................911
23.12.3 Stop ............................................................................911
23.12.4 Refresh ........................................................................912
23.12.5 Fonts ...........................................................................912
23.12.6 Separate Window .........................................................912

23.13 Tools Menu .................................................................913
23.13.1 Spelling .......................................................................913
23.13.2 Spelling Options ...........................................................917
23.13.3 Scripting Editor .............................................................921
23.13.4 Macros ........................................................................922
23.13.5 Comparisons ...............................................................922
   – Compare Open File With ..................................................922
   – Compare Directories .......................................................926
   – Compare Options ..........................................................929
23.13.6 User-defined Tools .......................................................931
23.13.7 Global Resources ........................................................931
23.13.8 Active Configuration ..................................................932
23.13.9 Manage Raptor Servers ..............................................932
23.13.10 Raptor Servers and Configurations ..............................936
23.13.11 Customize .................................................................936
   – Commands ......................................................................937
   – Toolbars .........................................................................938
   – Tools ..............................................................................940
   – Keyboard .......................................................................941
   – Menu .............................................................................945
   – Macros ...........................................................................947
   – Plug-Ins ..........................................................................948
   – Options ...........................................................................950
   – Customize Context Menu .................................................950
23.13.12 Restore Toolbars and Windows .................................954

23.13.13 Options .................................................................954
   – File ..............................................................................955
   – File Types ......................................................................958
   – Encoding .......................................................................961
   – Editing ..........................................................................962
   – View .............................................................................963
Programmers' Reference 994

1 Scripting Editor 996

1.1 Overview .............................................. 998
   1.1.1 Scripting Projects in XMLSpy .................. 998
   1.1.2 The Scripting Editor GUI ..................... 999
   1.1.3 Components of a Scripting Project ......... 1003

1.2 Creating a Scripting Project ...................... 1006

1.3 Global Declarations .............................. 1008

1.4 Forms ............................................... 1010
1.4.1 Creating a New Form ................................................................. 1010
1.4.2 Form Design and Form Objects .............................................. 1011
1.4.3 Form Events ................................................................. 1013
1.5 Events ......................................................................................... 1016
1.6 Macros ......................................................................................... 1020
  1.6.1 Creating and Editing a Macro ............................................. 1020
  1.6.2 Running a Macro ............................................................. 1022
  1.6.3 Debugging a Macro ........................................................ 1024
1.7 Programming Points .................................................................. 1025
  1.7.1 Built-in Commands ..................................................... 1026
    – Form usage and commands ............................................. 1033
1.8 Migrating to Scripting Editor 2010 and Later ..................... 1035

2 IDE Plugins .................................................................................. 1038
  2.1 Registration of IDE PlugIns ................................................... 1039
  2.2 ActiveX Controls ..................................................................... 1040
  2.3 Configuration XML .................................................................. 1041
  2.4 ATL sample files .................................................................. 1044
    2.4.1 Interface description (IDL) ........................................... 1044
    2.4.2 Class definition ............................................................... 1046
    2.4.3 Implementation .............................................................. 1047
  2.5 IXMLSpyPlugIn .................................................................. 1050
    2.5.1 OnCommand ............................................................. 1050
    2.5.2 OnUpdateCommand ................................................... 1051
    2.5.3 OnEvent ..................................................................... 1052
    2.5.4 GetUIModifications ..................................................... 1054
    2.5.5 GetDescription ............................................................ 1054

3 Application API ......................................................................... 1056
  3.1 Overview ................................................................................ 1058
    3.1.1 Object Model .............................................................. 1058
    3.1.2 Programming Languages .............................................. 1059
      – JScript ........................................................................ 1060
        Start Application ........................................................ 1061
        Simple Document Access .............................................. 1061
        Iteration ................................................................ 1062
        Error Handling .......................................................... 1063
        Events ................................................................. 1063
        Import and Export of Data .................................. 1064
        Import from Database ............................................ 1065
### 3.2 Interfaces

<table>
<thead>
<tr>
<th>3.2.1 Application</th>
<th>1094</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>1096</td>
</tr>
<tr>
<td>OnBeforeOpenDocument</td>
<td>1096</td>
</tr>
<tr>
<td>OnBeforeOpenProject</td>
<td>1096</td>
</tr>
<tr>
<td>OnDocumentOpened</td>
<td>1097</td>
</tr>
<tr>
<td>OnProjectOpened</td>
<td>1098</td>
</tr>
<tr>
<td>ActiveDocument</td>
<td>1098</td>
</tr>
<tr>
<td>AddMacroMenuItem</td>
<td>1098</td>
</tr>
<tr>
<td>AddXSLT_XQParameter</td>
<td>1099</td>
</tr>
<tr>
<td>Application</td>
<td>1099</td>
</tr>
<tr>
<td>ClearMacroMenu</td>
<td>1099</td>
</tr>
<tr>
<td>CreateXMLSchemaFromDBStructure</td>
<td>1099</td>
</tr>
<tr>
<td>CurrentProject</td>
<td>1100</td>
</tr>
<tr>
<td>Dialogs</td>
<td>1100</td>
</tr>
<tr>
<td>Documents</td>
<td>1100</td>
</tr>
<tr>
<td>Edition</td>
<td>1101</td>
</tr>
<tr>
<td>FindInFiles</td>
<td>1101</td>
</tr>
<tr>
<td>GetDatabaseImportElementList</td>
<td>1101</td>
</tr>
<tr>
<td>GetDatabaseSettings</td>
<td>1102</td>
</tr>
<tr>
<td>GetDatabaseTables</td>
<td>1102</td>
</tr>
<tr>
<td>GetExportSettings</td>
<td>1103</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3.1.5 Obsolete: Authentic View Editing operations</th>
<th>1092</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>1080</td>
</tr>
<tr>
<td>Java</td>
<td>1081</td>
</tr>
<tr>
<td>Example Java Project</td>
<td>1082</td>
</tr>
<tr>
<td>Application Startup and Shutdown</td>
<td>1086</td>
</tr>
<tr>
<td>Simple Document Access</td>
<td>1087</td>
</tr>
<tr>
<td>Iterations</td>
<td>1087</td>
</tr>
<tr>
<td>Use of Out-Parameters</td>
<td>1087</td>
</tr>
<tr>
<td>Event Handlers</td>
<td>1088</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3.1.4 Obsolete: Authentic View Row operations</th>
<th>1092</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>1080</td>
</tr>
<tr>
<td>Java</td>
<td>1081</td>
</tr>
<tr>
<td>Example Java Project</td>
<td>1082</td>
</tr>
<tr>
<td>Application Startup and Shutdown</td>
<td>1086</td>
</tr>
<tr>
<td>Simple Document Access</td>
<td>1087</td>
</tr>
<tr>
<td>Iterations</td>
<td>1087</td>
</tr>
<tr>
<td>Use of Out-Parameters</td>
<td>1087</td>
</tr>
<tr>
<td>Event Handlers</td>
<td>1088</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3.1.3 The DOM and XMLData</th>
<th>1089</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>1080</td>
</tr>
<tr>
<td>Java</td>
<td>1081</td>
</tr>
<tr>
<td>Example Java Project</td>
<td>1082</td>
</tr>
<tr>
<td>Application Startup and Shutdown</td>
<td>1086</td>
</tr>
<tr>
<td>Simple Document Access</td>
<td>1087</td>
</tr>
<tr>
<td>Iterations</td>
<td>1087</td>
</tr>
<tr>
<td>Use of Out-Parameters</td>
<td>1087</td>
</tr>
<tr>
<td>Event Handlers</td>
<td>1088</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3.1.2 Interfaces</th>
<th>1094</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>1096</td>
</tr>
<tr>
<td>OnBeforeOpenDocument</td>
<td>1096</td>
</tr>
<tr>
<td>OnBeforeOpenProject</td>
<td>1096</td>
</tr>
<tr>
<td>OnDocumentOpened</td>
<td>1097</td>
</tr>
<tr>
<td>OnProjectOpened</td>
<td>1098</td>
</tr>
<tr>
<td>ActiveDocument</td>
<td>1098</td>
</tr>
<tr>
<td>AddMacroMenuItem</td>
<td>1098</td>
</tr>
<tr>
<td>AddXSLT_XQParameter</td>
<td>1099</td>
</tr>
<tr>
<td>Application</td>
<td>1099</td>
</tr>
<tr>
<td>ClearMacroMenu</td>
<td>1099</td>
</tr>
<tr>
<td>CreateXMLSchemaFromDBStructure</td>
<td>1099</td>
</tr>
<tr>
<td>CurrentProject</td>
<td>1100</td>
</tr>
<tr>
<td>Dialogs</td>
<td>1100</td>
</tr>
<tr>
<td>Documents</td>
<td>1100</td>
</tr>
<tr>
<td>Edition</td>
<td>1101</td>
</tr>
<tr>
<td>FindInFiles</td>
<td>1101</td>
</tr>
<tr>
<td>GetDatabaseImportElementList</td>
<td>1101</td>
</tr>
<tr>
<td>GetDatabaseSettings</td>
<td>1102</td>
</tr>
<tr>
<td>GetDatabaseTables</td>
<td>1102</td>
</tr>
<tr>
<td>GetExportSettings</td>
<td>1103</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3.1.1 Application</th>
<th>1094</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>1096</td>
</tr>
<tr>
<td>OnBeforeOpenDocument</td>
<td>1096</td>
</tr>
<tr>
<td>OnBeforeOpenProject</td>
<td>1096</td>
</tr>
<tr>
<td>OnDocumentOpened</td>
<td>1097</td>
</tr>
<tr>
<td>OnProjectOpened</td>
<td>1098</td>
</tr>
<tr>
<td>ActiveDocument</td>
<td>1098</td>
</tr>
<tr>
<td>AddMacroMenuItem</td>
<td>1098</td>
</tr>
<tr>
<td>AddXSLT_XQParameter</td>
<td>1099</td>
</tr>
<tr>
<td>Application</td>
<td>1099</td>
</tr>
<tr>
<td>ClearMacroMenu</td>
<td>1099</td>
</tr>
<tr>
<td>CreateXMLSchemaFromDBStructure</td>
<td>1099</td>
</tr>
<tr>
<td>CurrentProject</td>
<td>1100</td>
</tr>
<tr>
<td>Dialogs</td>
<td>1100</td>
</tr>
<tr>
<td>Documents</td>
<td>1100</td>
</tr>
<tr>
<td>Edition</td>
<td>1101</td>
</tr>
<tr>
<td>FindInFiles</td>
<td>1101</td>
</tr>
<tr>
<td>GetDatabaseImportElementList</td>
<td>1101</td>
</tr>
<tr>
<td>GetDatabaseSettings</td>
<td>1102</td>
</tr>
<tr>
<td>GetDatabaseTables</td>
<td>1102</td>
</tr>
<tr>
<td>GetExportSettings</td>
<td>1103</td>
</tr>
</tbody>
</table>
- GetTextImportElementList ................................................................. 1103
- GetTextImportExportSettings .............................................................. 1104
- GetXSLT_XQParameterCount ............................................................... 1104
- GetXSLT_XQParameterName ............................................................... 1105
- GetXSLT_XQParameterXPath .............................................................. 1105
- ImportFromDatabase ........................................................................... 1105
- ImportFromSchema ............................................................................ 1106
- ImportFromText .................................................................................. 1107
- ImportFromWord .................................................................................. 1108
- IsAPISupported ................................................................................. 1108
- MajorVersion ..................................................................................... 1108
- MinorVersion ..................................................................................... 1108
- NewProject ......................................................................................... 1109
- OpenProject ....................................................................................... 1109
- Parent ............................................................................................... 1109
- Quit ..................................................................................................... 1110
- ReloadSettings .................................................................................. 1110
- RemoveXSLT_XQParameter ............................................................... 1110
- RunMacro ........................................................................................... 1111
- ScriptingEnvironment ........................................................................ 1111
- ServicePackVersion .......................................................................... 1111
- ShowApplication ............................................................................... 1111
- ShowFindInFiles .............................................................................. 1112
- ShowForm .......................................................................................... 1112
- Status ............................................................................................... 1112
- URLDelete .......................................................................................... 1113
- URLMakeDirectory ........................................................................... 1113
- Visible .............................................................................................. 1113
- WarningNumber ................................................................................ 1114
- WarningText ...................................................................................... 1114

3.2.2 AuthenticContextMenu ..................................................................... 1114
- CountItems ....................................................................................... 1114
- DeleteItem .......................................................................................... 1114
- GetItemText ....................................................................................... 1115
- InsertItem ........................................................................................... 1115
- SetItemText ......................................................................................... 1115

3.2.3 AuthenticDataTransfer ..................................................................... 1115
- dropEffect .......................................................................................... 1116
- getData .............................................................................................. 1116
- ownDrag ............................................................................................ 1117
- type ................................................................................................... 1117

3.2.4 AuthenticEventContext ................................................................. 1117
- EvaluateXPath ................................................................................... 1117
- GetEventContextType ........................................................................ 1118
- GetNormalizedTextValue .................................................................... 1118
- GetVariableValue..................................................................................................... 1118
- GetXMLNode.......................................................................................................... 1119
- IsAvailable............................................................................................................. 1119
- SetVariableValue.................................................................................................... 1119
  3.2.5
  AuthenticRange...................................................................................................... 1120
- AppendRow.............................................................................................................. 1121
- Application............................................................................................................. 1122
- CanPerformAction.................................................................................................. 1122
- CanPerformActionWith........................................................................................... 1123
- Clone....................................................................................................................... 1123
- CollapsToBegin...................................................................................................... 1123
- CollapsToEnd.......................................................................................................... 1124
- Copy....................................................................................................................... 1124
- Cut......................................................................................................................... 1124
- Delete..................................................................................................................... 1125
- DeleteRow............................................................................................................... 1125
- DuplicateRow......................................................................................................... 1125
- EvaluateXPath........................................................................................................ 1126
- ExpandTo............................................................................................................... 1126
- FirstCursorPosition.............................................................................................. 1127
- FirstXMLData......................................................................................................... 1128
- FirstXMLDataOffset.............................................................................................. 1128
- GetElementAttributeNames.................................................................................... 1129
- GetElementAttributeValue....................................................................................... 1130
- GetElementHierarchy............................................................................................. 1130
- GetEntityNames..................................................................................................... 1131
- GetVariableValue................................................................................................... 1132
- Goto....................................................................................................................... 1132
- GotoNext................................................................................................................. 1132
- GotoNextCursorPosition....................................................................................... 1133
- GotoPrevious......................................................................................................... 1133
- GotoPreviousCursorPosition................................................................................ 1134
- HasElementCursorPosition.................................................................................... 1134
- InsertEntity............................................................................................................ 1135
- InsertRow............................................................................................................... 1135
- IsCopyEnabled....................................................................................................... 1136
- IsCutEnabled.......................................................................................................... 1136
- IsDeleteEnabled..................................................................................................... 1136
- IsEmpty.................................................................................................................. 1137
- IsEqual.................................................................................................................... 1137
- IsFirstRow.............................................................................................................. 1137
- IsInDynamicTable................................................................................................. 1137
- IsLastRow............................................................................................................... 1138
- IsPasteEnabled...................................................................................................... 1138
- IsSelected.............................................................................................................. 1138
- IsTextStateApplied................................................................. 1138
- LastTextPosition................................................................. 1139
- LastXMLData................................................................. 1140
- LastXMLDataOffset....................................................... 1140
- MoveBegin................................................................. 1141
- MoveEnd................................................................. 1142
- MoveRowDown............................................................... 1142
- MoveRowUp................................................................. 1142
- Parent................................................................. 1143
- Parent................................................................. 1143
- Paste................................................................. 1143
- PerformAction............................................................... 1143
- Select................................................................. 1144
- SelectNext............................................................... 1144
- SelectPrevious............................................................. 1145
- SetElementAttribute.................................................. 1146
- SetFromRange............................................................. 1147
- SetVariableValue.......................................................... 1147
- Text................................................................. 1148

3.2.6 AuthenticView............................................................ 1148

- Events................................................................. 1149
  OnBeforeCopy.......................................................... 1149
  OnBeforeCut.......................................................... 1150
  OnBeforeDelete..................................................... 1150
  OnBeforeDrop.......................................................... 1151
  OnBeforePaste....................................................... 1152
  OnBeforeSave.......................................................... 1152
  OnDragOver............................................................. 1153
  OnKeyDown............................................................. 1154
  OnLoad................................................................. 1155
  OnMouseEvent.......................................................... 1155
  OnSelectionChanged................................................ 1156
  OnToolBarButtonClick........................................... 1156
  OnToolBarButtonExecuted....................................... 1158
  OnUserAddedXMLNode................................................ 1158

- Application.............................................................. 1158
- AsXMLString............................................................. 1159
- ContextMenu............................................................ 1159
- CreateXMLNode.......................................................... 1159
- DisableAttributeEntryHelper....................................... 1160
- DisableElementEntryHelper......................................... 1160
- DisableEntityEntryHelper............................................. 1160
- DocumentBegin.......................................................... 1160
- DocumentEnd........................................................... 1160
- DoNotPerformStandardAction....................................... 1161
- EvaluateXPath........................................................... 1161
- Event ............................................................................................................................ 1161
- EventContext .................................................................................................................. 1161
- GetToolBarButtonState .................................................................................................... 1161
- Goto .................................................................................................................................. 1162
- IsRedoEnabled .................................................................................................................. 1163
- IsUndoEnabled .................................................................................................................. 1163
- MarkupVisibility ............................................................................................................... 1164
- Parent ............................................................................................................................... 1164
- Print .................................................................................................................................. 1164
- Redo .................................................................................................................................. 1164
- Selection ........................................................................................................................... 1165
- SetToolBarButtonState ....................................................................................................... 1165
- Undo .................................................................................................................................. 1166
- UpdateXMLInstanceEntities .............................................................................................. 1166
- WholeDocument ............................................................................................................... 1167
- XMLDataRoot .................................................................................................................... 1167

3.2.7 CodeGeneratorDlg ........................................................................................................ 1167
- Application ...................................................................................................................... 1168
- CompatibilityMode (obsolete) ........................................................................................... 1168
- CPPSettings_DOMType ..................................................................................................... 1169
- CPPSettings_GenerateVC6ProjectFile .............................................................................. 1169
- CPPSettings_GenerateGCCMakefile ................................................................................. 1169
- CPPSettings_GenerateVSProjectFile ............................................................................... 1169
- CPPSettings_LibraryType .................................................................................................. 1170
- CPPSettings_UseMFC ......................................................................................................... 1170
- CSharpSettings_ProjectType ............................................................................................ 1170
- OutputPath ....................................................................................................................... 1171
- OutputPathDialogAction ................................................................................................... 1171
- OutputResultDialogAction ............................................................................................... 1171
- Parent ................................................................................................................................ 1171
- ProgrammingLanguage ..................................................................................................... 1172
- PropertySheetDialogAction .............................................................................................. 1172
- TemplateFileName ............................................................................................................ 1172

3.2.8 DatabaseConnection ..................................................................................................... 1173
- ADOConnection ................................................................................................................ 1173
- AsAttributes ..................................................................................................................... 1174
- CommentIncluded ............................................................................................................. 1174
- CreateMissingTables ........................................................................................................ 1174
- CreateNew ....................................................................................................................... 1175
- DatabaseKind .................................................................................................................. 1175
- DatabaseSchema .............................................................................................................. 1175
- ExcludeKeys ..................................................................................................................... 1175
- File .................................................................................................................................... 1176
- ForeignKeys ..................................................................................................................... 1176
- ImportColumnsType ....................................................................................................... 1176
- IncludeEmptyElements ................................................................. 1177
- NullReplacement ................................................................. 1177
- NumberDateTimeFormat ............................................................. 1177
- ODBCConnection ................................................................. 1177
- PrimaryKeys ................................................................. 1178
- SchemaExtensionType ............................................................. 1178
- SchemaFormat ................................................................. 1178
- SQLSelect ................................................................. 1178
- TextFieldLen ................................................................. 1179
- UniqueKeys ................................................................. 1179

3.2.9 Dialogs ............................................................................. 1179
- Application ........................................................................ 1180
- CodeGeneratorDlg ................................................................. 1180
- FileSelectionDlg ................................................................. 1180
- Parent .............................................................................. 1180
- SchemaDocumentationDlg ...................................................... 1181
- GenerateSampleXMLDlg ......................................................... 1181
- DTDSchemaGeneratorDlg ......................................................... 1181
- FindInFilesDlg ...................................................................... 1181

3.2.10 Document ........................................................................ 1182
- Events .................................................................................. 1184
  OnBeforeSaveDocument ....................................................... 1184
  OnBeforeCloseDocument ....................................................... 1184
  OnBeforeValidate ................................................................. 1185
  OnCloseDocument ................................................................. 1186
  OnViewActivation ................................................................. 1186
- Application ........................................................................ 1187
- AssignDTD ........................................................................ 1187
- AssignSchema ..................................................................... 1187
- AssignXSL .......................................................................... 1187
- AssignXSLFO ................................................................. 1188
- AsXMLString ...................................................................... 1188
- AuthenticView ..................................................................... 1188
- Close .............................................................................. 1189
- ConvertDTDOrSchema ........................................................... 1189
- ConvertDTDOrSchemaEx ........................................................ 1190
- ConvertToWSDL20 ................................................................. 1191
- CreateChild ................................................................. 1191
- CreateDBStructureFromXMLSchema ........................................ 1191
- CreateSchemaDiagram ........................................................... 1192
- CurrentViewMode ................................................................. 1192
- DataRoot .......................................................................... 1192
- DocEditView ...................................................................... 1193
- Encoding .......................................................................... 1193
- EndChanges ...................................................................... 1194
- ExecuteXQuery
- ExportToDatabase
- ExportToText
- FlattenDTDOrSchema
- FullName
- GenerateDTDOrSchema
- GenerateDTDOrSchemaEx
- GenerateProgramCode
- GenerateSampleXML
- GenerateSchemaDocumentation
- GetDBStructureList
- GetExportElementList
- GetPathName (obsolete)
- GridView
- IsModified
- IsValid
- IsValidEx
- IsWellFormed
- Name
- Parent
- Path
- RootElement
- Save
- SaveAs
- Saved
- SaveInString
- SaveToURL
- SetActiveDocument
- SetEncoding (obsolete)
- SetExportValid
- SetPathName (obsolete)
- StartChanges
- Suggestions
- SwitchViewMode
- TextView
- Title
- TransformXSL
- TransformXSLEx
- TransformXSLFO
- UpdateViews
- UpdateXMLData
- NewAuthService
- NewFile .......................................................... 1212
- NewFileFromText ............................................. 1212
- OpenAuthenticFile ........................................... 1213
- OpenFile .......................................................... 1213
- OpenURL ............................................................ 1214
- OpenURLDialog .................................................. 1214

3.2.12 DTDSchemaGeneratorDlg .................................. 1215
- Application ......................................................... 1215
- AttributeTypeDefinition ...................................... 1215
- DTDSchemaFormat .............................................. 1216
- FrequentElements .............................................. 1216
- GlobalAttributes .............................................. 1216
- MaxEnumLength .................................................. 1216
- MergeAllEqualNamed .......................................... 1217
- OnlyStringEnums .............................................. 1217
- OutputPath ......................................................... 1217
- OutputPathDialogAction ....................................... 1217
- Parent ............................................................... 1218
- ResolveEntities .................................................. 1218
- TypeDetection .................................................... 1218
- ValueList ........................................................... 1218

3.2.13 ElementList ................................................... 1218
- Count ............................................................... 1219
- Item ................................................................. 1219
- RemoveElement .................................................. 1219

3.2.14 ElementListItem .............................................. 1219
- ElementKind ...................................................... 1220
- FieldCount ....................................................... 1220
- Name ............................................................... 1220
- RecordCount ..................................................... 1220

3.2.15 ExportSettings .................................................. 1221
- CreateKeys ....................................................... 1221
- ElementList ...................................................... 1221
- EntitiesToText ................................................... 1222
- ExportAllElements ............................................ 1222
- ExportCompleteXML ......................................... 1222
- FromAttributes ................................................. 1222
- FromSingleSubElements ..................................... 1222
- FromTextValues .................................................. 1223
- IndependentPrimaryKey ...................................... 1223
- Namespace ........................................................ 1223
- StartFromElement .............................................. 1223
- SubLevelLimit .................................................... 1223

3.2.16 FileSelectionDlg .............................................. 1224
- Application ....................................................... 1224
3.2.17 FindInFilesDlg
- DialogAction ................................................................. 1224
- FullName ................................................................. 1225
- Parent ................................................................. 1225

3.2.17 FindInFilesDlg
- AdvancedXMLSearch ............................................. 1226
- Application ............................................................ 1226
- DoReplace ............................................................ 1226
- FileExtension .......................................................... 1226
- Find ................................................................. 1227
- IncludeSubfolders .................................................. 1227
- MatchCase ............................................................ 1227
- MatchWholeWord ................................................... 1227
- Parent ................................................................. 1227
- RegularExpression ................................................ 1228
- Replace ............................................................... 1228
- ReplaceOnDisk ........................................................ 1228
- SearchInProjectFiles DoExternal ......................... 1228
- SearchLocation ...................................................... 1229
- ShowResult ............................................................. 1229
- StartFolder ............................................................. 1229
- XMLAttributeContents ........................................... 1229
- XMLAttributeNames .............................................. 1229
- XMLCData ............................................................. 1230
- XMLComments ........................................................ 1230
- XMLElementContents ............................................... 1230
- XMLElementNames .................................................. 1230
- XMLPI ............................................................... 1230
- XMLRest .............................................................. 1231

3.2.18 FindInFilesResult
- Application ............................................................. 1231
- Count ................................................................. 1231
- Document ............................................................. 1232
- Item ................................................................. 1232
- Parent ................................................................. 1232
- Path ................................................................. 1232

3.2.19 FindInFilesResultMatch
- Application ............................................................ 1232
- Length ............................................................... 1233
- Line ................................................................. 1233
- LineText ............................................................. 1233
- Parent ................................................................. 1234
- Position ............................................................... 1234
- Replaced .............................................................. 1234

3.2.20 FindInFilesResults
- Application ............................................................. 1234
### 3.2.21 GenerateSampleXMLDlg
- Application
- ChoiceMode
- ConsiderSampleValueHints
- ContentOfNillableElementsIsNonMandatory
- FillAttributesWithSampleData
- FillElementsWithSampleData
- FillWithSampleData - obsolete
- LocalNameOfRootElement
- NamespaceURIOfRootElement
- NonMandatoryAttributes
- NonMandatoryElements
- Optimization - obsolete
- OptionsDialogAction
- Parent
- RepeatCount
- SampleValueHints
- SchemaOrDTDAssignment
- TakeFirstChoice - obsolete
- TryToUseNonAbstractTypes

### 3.2.22 GridView
- Events
  - OnBeforeDrag
  - OnBeforeDrop
  - OnBeforeStartEditing
  - OnEditingFinished
  - OnFocusChanged
- CurrentFocus
- Deselect
- IsVisible
- Select
- SetFocus

### 3.2.23 SchemaDocumentationDlg
- AllDetails
- Application
- CreateDiagramsFolder
- DiagramFormat
- EmbedCSSInHTML
- EmbedDiagrams
- GenerateRelativeLinks
- IncludeAll
- IncludeAttributeGroups
<table>
<thead>
<tr>
<th>SpyProjectItem</th>
<th>SpyProject Item</th>
<th>SpyProject</th>
<th>SpyProject</th>
<th>SpyProject</th>
<th>SpyProject Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseProject</td>
<td>ProjectFile</td>
<td>RootItems</td>
<td>SaveProject</td>
<td>SaveProjectAs</td>
<td>ChildItems</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>FileExtensions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Itemtype</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Open</td>
</tr>
</tbody>
</table>

- IncludeComplexTypes ................................................................. 1248
- IncludeGlobalAttributes ............................................................ 1248
- IncludeGlobalElements ............................................................... 1248
- IncludeGroups ......................................................................... 1248
- IncludeIndex ............................................................................ 1249
- IncludeLocalAttributes .............................................................. 1249
- IncludeLocalElements ................................................................. 1249
- IncludeRedefines ...................................................................... 1250
- IncludeReferencedSchemas .......................................................... 1250
- IncludeSimpleTypes ................................................................... 1250
- MultipleOutputFiles ................................................................... 1251
- OptionsDialogAction ................................................................... 1251
- OutputFile .................................................................................. 1251
- OutputFileDialogAction ............................................................... 1252
- OutputFormat ............................................................................... 1252
- Parent ....................................................................................... 1252
- ShowAnnotations ......................................................................... 1252
- ShowAttributes .......................................................................... 1253
- ShowChildren ............................................................................. 1253
- ShowDiagram .............................................................................. 1253
- ShowEnumerations ....................................................................... 1254
- ShowIdentityConstraints ............................................................. 1254
- ShowNamespace .......................................................................... 1254
- ShowPatterns ............................................................................. 1255
- ShowProgressBar ......................................................................... 1255
- ShowProperties ........................................................................... 1255
- ShowResult .................................................................................. 1255
- ShowSingleFacets ...................................................................... 1256
- ShowSourceCode ......................................................................... 1256
- ShowType .................................................................................... 1256
- ShowUsedBy ............................................................................... 1257
- SPSFile ....................................................................................... 1257
- UseFixedDesign ........................................................................... 1257
- IncludeSimpleTypes ................................................................... 1258
- IncludeGlobalAttributes ............................................................ 1258
- IncludeGlobalElements ............................................................... 1258
- IncludeGroups ......................................................................... 1258
- IncludeIndex ............................................................................ 1259
- IncludeLocalAttributes .............................................................. 1259
- IncludeLocalElements ................................................................. 1259
- IncludeRedefines ...................................................................... 1259
- IncludeReferencedSchemas .......................................................... 1259
- IncludeSimpleTypes ................................................................... 1259
- MultipleOutputFiles ................................................................... 1259
- OptionsDialogAction ................................................................... 1259
- OutputFile .................................................................................. 1259
- OutputFileDialogAction ............................................................... 1259
- OutputFormat ............................................................................... 1259
- Parent ....................................................................................... 1259
- ShowAnnotations ......................................................................... 1259
- ShowAttributes .......................................................................... 1259
- ShowChildren ............................................................................. 1259
- ShowDiagram .............................................................................. 1259
- ShowEnumerations ....................................................................... 1259
- ShowIdentityConstraints ............................................................. 1259
- ShowNamespace .......................................................................... 1259
- ShowPatterns ............................................................................. 1259
- ShowProgressBar ......................................................................... 1259
- ShowProperties ........................................................................... 1259
- ShowResult .................................................................................. 1259
- ShowSingleFacets ...................................................................... 1259
- ShowSourceCode ......................................................................... 1259
- ShowType .................................................................................... 1259
- ShowUsedBy ............................................................................... 1259
- SPSFile ....................................................................................... 1259
- UseFixedDesign ........................................................................... 1259

Altova XMLSpy 2019 Professional Edition
- ParentItem ................................................................. 1260
- Path ........................................................................ 1261
- ValidateWith ........................................................... 1261
- XMLForXSLTransformation .................................... 1261
- XSLForXMLTransformation .................................... 1261
- XSLTransformationFileExtension .......................... 1261
- XSLTransformationFolder ....................................... 1261

3.2.26 SpyProjectItems ................................................. 1262
  - AddFile ................................................................. 1262
  - AddFolder ............................................................. 1262
  - AddURL ................................................................. 1262
  - Count .................................................................... 1263
  - Item ....................................................................... 1263
  - RemoveItem ............................................................ 1263

3.2.27 TextImportExportSettings .................................... 1263
  - CommentIncluded ................................................... 1264
  - DestinationFolder .................................................. 1264
  - EnclosingCharacter ............................................... 1264
  - Encoding ............................................................... 1264
  - EncodingByteOrder ............................................... 1265
  - FieldDelimiter ....................................................... 1265
  - FileExtension ......................................................... 1265
  - HeaderRow ............................................................. 1265
  - ImportFile .............................................................. 1265
  - RemoveDelimiter .................................................. 1266
  - RemoveNewline ...................................................... 1266

3.2.28 TextView ............................................................ 1266
  - Events .................................................................... 1267
    - OnBeforeShowSuggestions ................................ 1267
    - OnChar ................................................................. 1267
    - Application .......................................................... 1268
    - GetRangeText ....................................................... 1268
    - GoToLineChar ....................................................... 1268
    - Length ................................................................ 1268
    - LineCount ................................................................ 1269
    - LineFromPosition ................................................. 1269
    - LineLength .......................................................... 1269
    - MoveCaret ........................................................... 1269
    - Parent .................................................................... 1270
    - PositionFromLine ................................................. 1270
    - ReplaceText .......................................................... 1270
    - SelectionEnd ....................................................... 1270
    - SelectionStart ....................................................... 1270
    - SelectText ............................................................ 1271
    - SetText .................................................................. 1271
3.2.29 XMLData ................................................................. 1271
   - Text ........................................................................... 1271
   - AppendChild .............................................................. 1273
   - CountChildren .......................................................... 1273
   - CountChildrenKind .................................................. 1273
   - EraseAllChildren ..................................................... 1274
   - EraseChild ............................................................... 1274
   - EraseCurrentChild ................................................... 1274
   - GetChild ..................................................................... 1275
   - GetChildAttribute ...................................................... 1276
   - GetChildElement ........................................................ 1276
   - GetChildKind ............................................................. 1276
   - GetCurrentChild ....................................................... 1277
   - GetFirstChild ............................................................ 1277
   - GetNamespacePrefixForURI ........................................ 1277
   - GetNextChild ............................................................ 1278
   - GetTextValueXMLDecoded .......................................... 1279
   - HasChildren .............................................................. 1279
   - HasChildrenKind ...................................................... 1279
   - InsertChild ............................................................... 1280
   - InsertChildAfter ....................................................... 1280
   - InsertChildBefore .................................................... 1280
   - IsSameNode .............................................................. 1281
   - Kind ........................................................................... 1281
   - MayHaveChildren ..................................................... 1281
   - Name ......................................................................... 1282
   - Parent ......................................................................... 1282
   - SetTextValueXMLEncoded .......................................... 1282
   - TextValue .................................................................... 1282

3.3 Interfaces (obsolete) ...................................................... 1284
3.3.1 AuthenticEvent (obsolete) ......................................... 1284
   - altKey (obsolete) ......................................................... 1285
   - altLeft (obsolete) ....................................................... 1286
   - button (obsolete) ....................................................... 1287
   - cancelBubble (obsolete) ............................................. 1288
   - clientX (obsolete) ....................................................... 1289
   - clientY (obsolete) ....................................................... 1290
   - ctrlKey (obsolete) ....................................................... 1291
   - ctrlLeft (obsolete) ..................................................... 1292
   - dataTransfer (obsolete) .............................................. 1293
   - fromElement (obsolete) .............................................. 1294
   - keyCode (obsolete) ..................................................... 1294
   - propertyName (obsolete) ............................................. 1295
   - repeat (obsolete) ....................................................... 1295
   - returnValue (obsolete) .............................................. 1295
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>shiftKey (obsolete)</td>
<td>1296</td>
</tr>
<tr>
<td>shiftLeft (obsolete)</td>
<td>1297</td>
</tr>
<tr>
<td>srcElement (obsolete)</td>
<td>1298</td>
</tr>
<tr>
<td>type (obsolete)</td>
<td>1299</td>
</tr>
<tr>
<td>AuthenticSelection (obsolete)</td>
<td>1300</td>
</tr>
<tr>
<td>End (obsolete)</td>
<td>1301</td>
</tr>
<tr>
<td>EndTextPosition (obsolete)</td>
<td>1301</td>
</tr>
<tr>
<td>Start (obsolete)</td>
<td>1302</td>
</tr>
<tr>
<td>StartTextPosition (obsolete)</td>
<td>1302</td>
</tr>
<tr>
<td>OldAuthentictView (obsolete)</td>
<td>1303</td>
</tr>
<tr>
<td>ApplyTextState (obsolete)</td>
<td>1305</td>
</tr>
<tr>
<td>CurrentSelection (obsolete)</td>
<td>1306</td>
</tr>
<tr>
<td>EditClear (obsolete)</td>
<td>1306</td>
</tr>
<tr>
<td>EditCopy (obsolete)</td>
<td>1307</td>
</tr>
<tr>
<td>EditCut (obsolete)</td>
<td>1307</td>
</tr>
<tr>
<td>EditPaste (obsolete)</td>
<td>1308</td>
</tr>
<tr>
<td>EditRedo (obsolete)</td>
<td>1308</td>
</tr>
<tr>
<td>EditSelectAll (obsolete)</td>
<td>1309</td>
</tr>
<tr>
<td>EditUndo (obsolete)</td>
<td>1309</td>
</tr>
<tr>
<td>event (obsolete)</td>
<td>1310</td>
</tr>
<tr>
<td>GetAllowedElements (obsolete)</td>
<td>1310</td>
</tr>
<tr>
<td>getNextVisible (obsolete)</td>
<td>1313</td>
</tr>
<tr>
<td>GetPreviousVisible (obsolete)</td>
<td>1314</td>
</tr>
<tr>
<td>IsEditClearEnabled (obsolete)</td>
<td>1315</td>
</tr>
<tr>
<td>IsEditCopyEnabled (obsolete)</td>
<td>1315</td>
</tr>
<tr>
<td>IsEditCutEnabled (obsolete)</td>
<td>1317</td>
</tr>
<tr>
<td>IsEditPasteEnabled (obsolete)</td>
<td>1317</td>
</tr>
<tr>
<td>IsEditRedoEnabled (obsolete)</td>
<td>1318</td>
</tr>
<tr>
<td>IsEditUndoEnabled (obsolete)</td>
<td>1318</td>
</tr>
<tr>
<td>IsRowAppendEnabled (obsolete)</td>
<td>1319</td>
</tr>
<tr>
<td>IsRowDeleteEnabled (obsolete)</td>
<td>1319</td>
</tr>
<tr>
<td>IsRowDuplicateEnabled (obsolete)</td>
<td>1320</td>
</tr>
<tr>
<td>IsRowInsertEnabled (obsolete)</td>
<td>1320</td>
</tr>
<tr>
<td>IsRowMoveDownEnabled (obsolete)</td>
<td>1321</td>
</tr>
<tr>
<td>IsRowMoveUpEnabled (obsolete)</td>
<td>1321</td>
</tr>
<tr>
<td>IsTextStateApplied (obsolete)</td>
<td>1322</td>
</tr>
<tr>
<td>IsTextStateEnabled (obsolete)</td>
<td>1322</td>
</tr>
<tr>
<td>LoadXML (obsolete)</td>
<td>1323</td>
</tr>
<tr>
<td>MarkUpView (obsolete)</td>
<td>1323</td>
</tr>
<tr>
<td>RowAppend (obsolete)</td>
<td>1325</td>
</tr>
<tr>
<td>RowDelete (obsolete)</td>
<td>1325</td>
</tr>
<tr>
<td>RowDuplicate (obsolete)</td>
<td>1326</td>
</tr>
<tr>
<td>RowInsert (obsolete)</td>
<td>1326</td>
</tr>
<tr>
<td>RowMoveDown (obsolete)</td>
<td>1327</td>
</tr>
<tr>
<td>RowMoveUp (obsolete)</td>
<td>1327</td>
</tr>
</tbody>
</table>
3.4 Enumerations .......................................................................................................................... 1332
3.4.1 ENUMApplicationStatus ................................................................................................. 1332
3.4.2 SPYAttributeTypeDefinition .......................................................................................... 1332
3.4.3 SPYAuthenticationActions .............................................................................................. 1332
3.4.4 SPYAuthenticationDocumentPosition ............................................................................ 1332
3.4.5 SPYAuthenticationElementActions ................................................................................ 1333
3.4.6 SPYAuthenticationElementKind .................................................................................... 1333
3.4.7 SPYAuthenticationMarkupVisibility ................................................................................ 1333
3.4.8 SPYAuthenticationToolBarButtonState .......................................................................... 1333
3.4.9 SPYDatabaseKind ........................................................................................................... 1334
3.4.10 SPYDialogAction ........................................................................................................ 1334
3.4.11 SPYDOMType ................................................................................................................ 1334
3.4.12 SPYDTDSchemaFormat .................................................................................................. 1335
3.4.13 SPYEncodingByteOrder ............................................................................................... 1335
3.4.14 SPYExportNamespace .................................................................................................... 1335
3.4.15 SPYFindInFilesSearchLocation ..................................................................................... 1335
3.4.16 SPYFrequentElements .................................................................................................. 1335
3.4.17 SPYImageKind ............................................................................................................... 1335
3.4.18 SPYImportColumnsType ............................................................................................... 1335
3.4.19 SPYKeyEvent ................................................................................................................ 1336
3.4.20 SPYKeyStatus ............................................................................................................... 1336
3.4.21 SPYLibType .................................................................................................................. 1336
3.4.22 SPYLoading ................................................................................................................ 1336
3.4.23 SPYMouseEvent ........................................................................................................... 1337
3.4.24 SPYNumberDateTimeFormat ......................................................................................... 1337
3.4.25 SPYProgrammingLanguage ........................................................................................ 1338
3.4.26 SPYProjectItemTypes .................................................................................................... 1338
3.4.27 SPYProjectType ............................................................................................................ 1338
3.4.28 SpySampleXMLGenerationChoiceMode ......................................................................... 1338
3.4.29 SPYSampleXMLGenerationOptimization (Obsolete) .................................................... 1339
3.4.30 SpySampleXMLGenerationSampleValueHints ............................................................. 1339
3.4.31 SPYSampleXMLGenerationSchemaOrDTDAssignment ................................................ 1339
3.4.32 SPYSchemaDefKind ...................................................................................................... 1339
3.4.33 SPYSchemaDocumentationFormat ................................................................................ 1340
3.4.34 SPYSchemaExtensionType ............................................................................................ 1340
3.4.35 SPYSchemaFormat ......................................................................................................... 1340
3.4.36 SPYTextDelimiters ........................................................................................................ 1341
3.4.37 SPYTextEnclosing ......................................................................................................... 1341
3.4.38 SPYTypeDetection ........................................................................................................ 1341
3.4.39 SPYURLTypes ............................................................................................................... 1341
3.4.40 SPYValidateXSDVersion .............................................................................................. 1341
3.5 Application API for Java (obsolete) ............................................................................................................................ 1344
3.5.1 Sample source code (obsolete) ........................................................................................................................................ 1345
3.5.2 SpyApplication (obsolete) ................................................................................................................................................ 1347
3.5.3 SpyCodeGeneratorDlg (obsolete) ...................................................................................................................................... 1348
3.5.4 SpyDatabaseConnection (obsolete) .................................................................................................................................... 1349
3.5.5 SpyDialogs (obsolete) ............................................................................................................................................................ 1350
3.5.6 SpyDoc (obsolete) ................................................................................................................................................................. 1351
3.5.7 SpyDocuments (obsolete) ....................................................................................................................................................... 1353
3.5.8 SpyDTDSchemaGeneratorDlg (obsolete) ................................................................................................................................. 1353
3.5.9 SpyElementList (obsolete) ....................................................................................................................................................... 1354
3.5.10 SpyElementListItem (obsolete) .............................................................................................................................................. 1354
3.5.11 SpyExportSettings (obsolete) ................................................................................................................................................. 1355
3.5.12 SpyFileSelectionDlg (obsolete) ............................................................................................................................................. 1356
3.5.13 SpyFindInFilesDlg (obsolete) .............................................................................................................................................. 1356
3.5.14 SpyFindInFilesMatch (obsolete) ........................................................................................................................................... 1357
3.5.15 SpyFindInFilesResult (obsolete) .......................................................................................................................................... 1358
3.5.16 SpyFindInFilesResults (obsolete) ...................................................................................................................................... 1358
3.5.17 SpyGenerateSampleXMLDlg (obsolete) ................................................................................................................................. 1358
3.5.18 SpyGridView (obsolete) ............................................................................................................................................................ 1359
3.5.19 SpyProject (obsolete) ............................................................................................................................................................... 1360
3.5.20 SpyProjectItem (obsolete) ....................................................................................................................................................... 1360
3.5.21 SpyProjectItems (obsolete) ..................................................................................................................................................... 1361
3.5.22 SpySchemaDocumentationDlg (obsolete) ............................................................................................................................... 1361
3.5.23 SpyTextImportExportSettings (obsolete) ............................................................................................................................. 1363
3.5.24 SpyTextView (obsolete) .......................................................................................................................................................... 1364
3.5.25 SpyXMLData (obsolete) .......................................................................................................................................................... 1364
3.5.26 Authentic (obsolete) ............................................................................................................................................................... 1365
- SpyAuthenticRange (obsolete) .................................................................................................................................................... 1365
- SpyAuthenticView (obsolete) ....................................................................................................................................................... 1367
- SpyDocEditSelection (obsolete) ............................................................................................................................................... 1367
- SpyDocEditView (obsolete) ...................................................................................................................................................... 1368
3.5.27 Predefined constants (obsolete) ........................................................................................................................................... 1369
- SPYApplicationStatus (obsolete) ............................................................................................................................................... 1369
- SPYAttributeTypeDefinition (obsolete) .................................................................................................................................. 1370
- SPYAuthenticActions (obsolete) ................................................................................................................................................. 1370
- SPYAuthenticDocumentPosition (obsolete) ................................................................................................................................. 1371
- SPYAuthenticElementKind (obsolete) ....................................................................................................................................... 1371
- SPYAuthenticMarkupVisibility (obsolete) ................................................................................................................................ 1372
- SPYDatabaseKind (obsolete) ....................................................................................................................................................... 1372
- SPYDialogAction (obsolete) ...................................................................................................................................................... 1373
- SPYDOMType (obsolete) ........................................................................................................................................................... 1373
4 ActiveX Integration

4.1 Prerequisites .................................................................................................................. 1387
4.2 Adding the ActiveX Controls to the Toolbox ................................................................. 1389
4.3 Integration at Application Level ...................................................................................... 1391
4.4 Integration at Document Level ......................................................................................... 1394
4.5 ActiveX Integration Examples ......................................................................................... 1398
4.5.1 C# .......................................................................................................................... 1398
    – Running the Sample C# Solution .................................................................................. 1398
4.5.2 HTML ......................................................................................................................... 1401
    – HTML Integration at Application Level....................................................................... 1401
        Instantiate the Control .............................................................................................. 1401
        Add Button to Open Default Document .................................................................. 1401
        Add Buttons for Code Generation ........................................................................... 1402
        Connect to Custom Events ....................................................................................... 1403
    – HTML Integration at Document Level ....................................................................... 1403

- SPYDTDSchemaFormat (obsolete) .................................................................................. 1374
- SPYEncodingByteOrder (obsolete) ................................................................................ 1374
- SPYExportNamespace (obsolete) .................................................................................. 1374
- SPYFindInFilesSearchLocation (obsolete) .................................................................. 1375
- SPYFrequentElements (obsolete) ................................................................................ 1375
- SPYImageKind (obsolete) ............................................................................................. 1375
- SPYImportColumnsType (obsolete) .............................................................................. 1376
- SPYLibType (obsolete) ................................................................................................. 1376
- SPYLoading (obsolete) ................................................................................................. 1376
- SPYNumberDateTimeFormat (obsolete) ....................................................................... 1377
- SPYProgrammingLanguage (obsolete) ...................................................................... 1377
- SPYProjectItemTypes (obsolete) .................................................................................. 1378
- SPYProjectType (obsolete) .......................................................................................... 1378
- SPYSampleXMLGenerationOptimization (obsolete) ...................................................... 1379
- SPYSampleXMLGenerationSchemaOrDTDAssignment (obsolete) ......................... 1379
- SPYSampleObjectDataKind (obsolete) ......................................................................... 1379
- SPYSchemaDefKind (obsolete) .................................................................................... 1379
- SPYSchemaDocumentationFormat (obsolete) ............................................................ 1380
- SPYSchemaExtensionType (obsolete) .......................................................................... 1381
- SPYSchemaFormat (obsolete) ....................................................................................... 1381
- SPYTextDelimiters (obsolete) ...................................................................................... 1381
- SPYTextEnclosing (obsolete) ...................................................................................... 1382
- SPYTypeDetection (obsolete) ...................................................................................... 1382
- SPYURLTypes (obsolete) .............................................................................................. 1383
- SpYViewModes (obsolete) ............................................................................................ 1383
- SPYWhitespaceComparison (obsolete) ....................................................................... 1384
- SPYXMLDataKind (obsolete) ....................................................................................... 1384
4.7 Object Reference

4.7.1 XMLSpyCommand
- Accelerator
- ID
- IsSeparator
- Label
- Name
- StatusText
- SubCommands
- ToolTip

4.7.2 XMLSpyCommands
- Count
4.7.3 XMLSpyControl

- Properties ................................................................................................................................. 1439
  Appearance ................................................................................................................................. 1440
  Application ................................................................................................................................. 1440
  BorderStyle ................................................................................................................................. 1441
  CommandsList .............................................................................................................................. 1441
  EnableUserPrompts ...................................................................................................................... 1441
  IntegrationLevel .......................................................................................................................... 1442
  MainMenu ...................................................................................................................................... 1442
  Toolbars ...................................................................................................................................... 1443
- Methods .................................................................................................................................. 1443
  Exec ........................................................................................................................................... 1444
  Open .......................................................................................................................................... 1444
  QueryStatus ................................................................................................................................. 1444
- Events .................................................................................................................................... 1444
  OnCloseEditingWindow ................................................................................................................. 1445
  OnContextChanged .................................................................................................................... 1445
  OnDocumentOpened .................................................................................................................. 1445
  OnFileChangedAlert ................................................................................................................... 1445
  OnLicenseProblem ..................................................................................................................... 1446
  OnOpenedOrFocused .................................................................................................................. 1446
  OnToolWindowUpdated ............................................................................................................. 1446
  OnUpdateCmdUI .......................................................................................................................... 1446
  OnValidationWindowUpdated ...................................................................................................... 1447

4.7.4 XMLSpyControlDocument

- Properties ................................................................................................................................. 1447
  Appearance ................................................................................................................................. 1448
  BorderStyle ................................................................................................................................ 1448
  Document ................................................................................................................................... 1448
  IsModified .................................................................................................................................. 1448
  Path ............................................................................................................................................ 1449
  ReadOnly .................................................................................................................................... 1449
- Methods .................................................................................................................................. 1449
  Exec ........................................................................................................................................... 1449
  New ........................................................................................................................................... 1450
  Open .......................................................................................................................................... 1450
  QueryStatus ................................................................................................................................. 1450
  Reload ....................................................................................................................................... 1450
  Save .......................................................................................................................................... 1451
  SaveAs ...................................................................................................................................... 1451
- Events .................................................................................................................................... 1451
  OnActivate .................................................................................................................................. 1451
  OnContextChanged .................................................................................................................... 1451
  OnDocumentClosed ................................................................................................................... 1452
Appendices

1 XSLT and XQuery Engine Information
1.1 XSLT 1.0 ................................................................. 1460
1.2 XSLT 2.0 ................................................................. 1461
1.3 XSLT 3.0 ................................................................. 1463
1.4 XQuery 1.0 ............................................................... 1464
1.5 XQuery 3.1 ............................................................... 1467

2 XSLT and XPath/XQuery Functions
2.1 Altova Extension Functions .............................................. 1470
   2.1.1 XSLT Functions .................................................. 1471
   2.1.2 XPath/XQuery Functions: Date and Time ............... 1473
   2.1.3 XPath/XQuery Functions: Geolocation ................. 1489
   2.1.4 XPath/XQuery Functions: Image-Related ............. 1499
   2.1.5 XPath/XQuery Functions: Numeric ..................... 1504
   2.1.6 XPath/XQuery Functions: Sequence ................... 1506
   2.1.7 XPath/XQuery Functions: String ....................... 1515
## 2.1.8 XPath/XQuery Functions: Miscellaneous

**Miscellaneous Extension Functions**

### 2.2 Java Extension Functions

- **User-Defined Class Files**
- **User-Defined Jar Files**
- **Java: Constructors**
- **Java: Static Methods and Static Fields**
- **Java: Instance Methods and Instance Fields**
- **Datatypes: XPath/XQuery to Java**
- **Datatypes: Java to XPath/XQuery**

### 2.2.2 .NET Extension Functions

- **.NET: Constructors**
- **.NET: Static Methods and Static Fields**
- **.NET: Instance Methods and Instance Fields**
- **Datatypes: XPath/XQuery to .NET**
- **Datatypes: .NET to XPath/XQuery**

### 2.2.3 MSXSL Scripts for XSLT

---

## 3 Datatypes in DB-Generated XML Schemas

### 3.1 ADO

### 3.2 MS Access

### 3.3 MS SQL Server

### 3.4 MySQL

### 3.5 ODBC

### 3.6 Oracle

### 3.7 Sybase

---

## 4 Datatypes in DBs Generated from XML Schemas

### 4.1 MS Access

### 4.2 MS SQL Server

### 4.3 MySQL

### 4.4 Oracle

---

## 5 Technical Data

### 5.1 OS and Memory Requirements

### 5.2 Altova Engines

### 5.3 Unicode Support

### 5.4 Internet Usage
6 License Information ........................................1564
6.1 Electronic Software Distribution .................................................................1565
6.2 Software Activation and License Metering ..................................................1566
6.3 Altova End-User License Agreement ............................................................1567

Index ..........................................................1568
Altova XMLSpy 2019 Professional Edition

Altova XMLSpy 2019 Professional Edition is the industry standard XML Development Environment for designing, editing and debugging enterprise-class applications involving XML, XML Schema, XSLT, and XQuery. It is the ultimate productivity enhancer for J2EE, .NET and database developers. XMLSpy® runs on Windows 7 SP1 with Platform Update, Windows 8, Windows 10, and Windows Server 2008 R2 SP1 with Platform Update or newer. XMLSpy is available in 64-bit and 32-bit versions. XMLSpy and Altova MissionKit can be integrated with applications of the Microsoft Office suite (MS Access, MS Excel, MS Word), version 2007 or newer.

This documentation is organized into the following main sections:

- XMLSpy Tutorial
- User Guide and Reference
- Programmers’ Reference
- Appendices

It is available in the following formats:

- As the built-in Help system of XMLSpy (Help menu or F1)
- In HTML and PDF formats, and for purchase as a book. These formats are available via the Altova website

Last updated: 28 March 2019
XMLSpy Tutorial

This tutorial provides an overview of XML and takes you through a number of key XML tasks. In the process you will learn how to use some of XMLSpy’s most powerful features.

The tutorial is divided into the following parts:

- **XMLSpy Interface**, which helps you to familiarize yourself with the application’s graphical user interface (GUI).
- **Creating an XML Schema**. You will learn how to create an XML Schema in XMLSpy’s intuitive Schema View, how to create complex content models using drag-and-drop mechanisms, and how to configure Schema View.
- **Using Schema View features** to create complex and simple types, global element references, and attribute enumerations.
- Learning how to navigate schemas in Schema View, and how to generate documentation of schemas.
- **Creating an XML document**. You will learn how to assign a schema for an XML document, edit an XML document in Grid View and Text View, and validate XML documents using XMLSpy’s built-in validator.
- **Transforming an XML file using an XSLT stylesheet**. This involves assigning an XSLT file and carrying out the transformation using XMLSpy’s built-in XSLT engines.
- **Working with XMLSpy projects**, which enable you to easily organize your XML documents.

Installation and configuration

This tutorial assumes that you have successfully installed XMLSpy on your computer and received a free evaluation key-code, or are a registered user. The evaluation version of XMLSpy is fully functional but limited to a 30-day period. You can request a regular license from our secure web server or through any one of our resellers.

Tutorial example files

The tutorial files are available in the application folder:

C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2019\Examples\Tutorial

The Examples folder contains various XML files for you to experiment with, while the Tutorial folder contains all the files used in this tutorial.

The Template folder in the application folder (typically in c:\Program Files\Altova) contains all the XML template files that are used whenever you select the menu option File | New. These files supply the necessary data (namespaces and XML declarations) for you to start working with the respective XML document immediately.
1 XMLSpy Interface

In this section of the tutorial, you will start XMLSpy and get to know the interface.

Starting XMLSpy

To start XMLSpy, double-click the XMLSpy icon on your desktop or use the Start | All Programs menu to access the XMLSpy program. XMLSpy is started with no documents open in the interface. Open XMLSpy now.

Overview of the interface

The default view of the XMLSpy interface is structured into three vertical areas (figure below). These three areas contain, from left to right: (i) the Project and Info windows; (ii) the Main and Output windows; and (iii) the Entry Helper windows. Look at the Project window. It will contain the Examples project, which is opened by default when you start XMLSpy for the first time.

Given below are key points that will help you to understand the layout of the interface and the functions of its various components. The sub-sections of this first part of the tutorial will help you get familiar with the interface.

Document bar in the Main window: When multiple documents are open, each document is displayed in a tab in the document bar of the Main window (see figure). Clicking a tab makes that document the active document. You can scroll document tabs by clicking the arrows on the right hand side of the document bar. Open two or more files (for example, from the Examples project), and check how the tabs work.
**Document editing views:** The active document can be viewed in one of multiple applicable editing views. For example:

- An XML (.xml) document can be viewed in Text View, Grid View, Authentic View, and Browser View, but cannot be viewed in other views, such as Schema View.
- An XML Schema (.xsd) document, on the other hand can be viewed in Text View, Grid View, Schema View, and Browser View, but not in Authentic View.

The following views are available: Text View, Grid View, Schema View, Authentic View, and Browser View.

**Entry helpers:** The entry helper windows change according to the kind of the active document (for example, XML or XSD or CSS or WSDL) and according to the currently active document view (for example, Text View or Schema View). The entry helpers enable you to quickly and correctly edit the active document by providing context-sensitive editing support.
1.1 The Views

In this part of the tutorial you will learn: (i) to switch between document editing views, and (ii) to change the default editing view of a particular document type.

Switching between document views

When you open a document it will open in the view that has been set as the default view for that type of document. Open a document as follows:

1. Click the command File | Open.
2. Browse for the file AddressFirst.xsd, which is located in the C:\Documents and Settings\<username>C:My Documents\Altova\XMLSpy2019\Examples\Tutorial folder, select it, and click Open. The file opens in Schema View.
3. Switch among the various views by clicking the view tabs at the bottom of the Main window (Text View, Grid View, etc). You will be able to view the XML Schema document in Text View, Grid View, Schema View, and Browser View.
4. You can also change views by selecting the view you want from the options in the View menu. Try switching the view of the AddressFirst.xsd document using the View menu commands.
5. Close the document (via File | Close).

Changing the default view of a document type

All documents with the .xsd extension will open by default in Schema View. You can change the default opening view of any type of document in the Options dialog. Let us do this for .xsd documents now.

1. Click the command Tools | Options and go to the File Types section (screenshot below; note that some options, such as those related to XBRL and Avro support, are available in Enterprise Edition only).
2. In the File Types pane, scroll down to .xsd and select it (highlighted in screenshot).
3. In the Default View pane, select Text View.
4. Click **OK**.
5. Click the **File | Open** command, and open the file `AddressFirst.xsd`. The file opens in **Text View**.
6. Switch to **Schema View** to see the file in this view, then close the file (**File | Close**).
7. Go back to the Options dialog (**Tools | Options**), and, in the **File Types** section, change the default view of `.xsd` files back to **Schema View**.

**Note:** In the **File Types** section of the Options dialog (**screenshot above**), you can change the default view of any of the listed file extensions. A new file extension can be added to the list via the **Add New File Extension** button.
1.2 **The Windows**

By default, the various windows are located around the Main window (see screenshot below) and are organized into the following window groups:

- Project window
- Info window
- Entry helpers (various, depending on the type of document currently active)
- Output windows: Messages, XPath, XSL Outline, Find in Files, Find in Schemas

In this section, you will learn how to turn on and off the display of window groups and how to move windows around the screen. Being able to manage the display of windows well will be useful when you need more space within the interface.

### Switching the display of window groups on and off

Window groups (Project Window, Info Window, Entry Helpers, Output Windows) can be displayed or hidden by toggling them on and off via the commands in the **Window** menu. A displayed window group can also be hidden by right-clicking its title bar and selecting the command **Hide**. A hidden window can only be displayed via the **Window** menu.

Open any XML file in the `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2019\Examples\Tutorial` folder, and practise using these basic commands till you are familiar with the way the commands work. For more information about displaying and hiding window groups, see the section, *[XMLSpy Interface]*. 
Moving windows around the screen

An individual window can either float free of the interface or be docked within it. A window can also be docked as a tab within a window group (window groups are explained above). For example, the screenshot below shows the Components entry helper in Schema View, which has three tabbed windows: the Globals window, Namespaces window, and Identity Constraints window.

A window can be made to float or dock using one of the following methods in any view:

- Double-click the title bar of the window. If docked, the window will now float. If floating, the window will now dock in the last position in which it was docked.
- Right-click the title bar of a window and choose the required command (Floating or Docking).
- Drag the window (using its title bar as a handle) out of its docked position so that it floats. Drag a floating window (by its title bar) to the location where it is to be docked. Two sets of blue arrows appear. The outer set of four arrows enables docking relative to the application window (along the top, right, bottom, or left edge of the GUI). The inner set of arrows enables docking relative to the window over which the cursor is currently placed. Dropping a dragged window on the button in the center of the inner set of arrows (or on the title bar of a window) docks the dragged window as a tabbed window within the window in which it is dropped.

To float a tabbed window, double-click its tab. To drag a tabbed window out of a group of tabbed windows, drag its tab.

To practise moving windows around open any XML Schema file from the C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2019\Examples\Tutorial folder, and, while in Schema View, try the methods described above till you are able to move windows around the interface comfortably.
1.3 Menus and Toolbars

In this section of the tutorial, you will quickly learn about the main features of the menus and toolbars of XMLSpy.

Menus

There are two menu bars: (i) a default menu that is displayed when no document is open, and (ii) the full XMLSpy application menu, which is displayed as soon as a document is open. Do the following:

1. Close all open documents with the menu command File | Close All. You will see the default menu.
2. Open the AddressFirst.xsd file by clicking its name from the list of most recently opened files located at the bottom of the File menu. When the file opens in Schema View, the menu will change to the full XMLSpy application menu.

The menus are organized primarily according to function, and a command in a menu is enabled only when it can be executed at the cursor point or for a selection in the current view of the active document. Do the following to understand the factors that determine whether a menu command is enabled or disabled:

1. Click the Schema Design menu. Notice that the Save Diagram, Configure View, and Zoom commands are disabled (screenshot below).

![Schema Design Menu Screenshot]

2. Click in a blank space outside the menu to make the menu disappear. Then click the Display Diagram icon located to the left of the element component. This takes you to the Content Model View of Schema View (the second of Schema View’s two views; the first is Schema Overview). If you check the Schema Design menu now, you will see that the Save Diagram, Configure View, and Zoom commands have been enabled. They are enabled only in the Content Model View of Schema View, not in the Schema Overview of Schema View, nor in any other view. Note also that only XML Schema files can be opened in Schema View.
3. An XML Schema file is also an XML file, so it is displayed as an XML file in Text View and Grid View, and all menu commands that apply to XML files will be enabled in these views. Compare commands in the Edit menu (whether they are enabled or not) in Schema View and Text View.
4. Next compare commands in the XML | Insert menu (enabled or disabled) in Text View and Grid View. The commands in this menu are enabled only in Grid View.

For descriptions of all the menu commands, see the User Reference section of the user
Toolbars

The display of toolbars varies according to the current view. The application’s default settings provide the correct toolbars for each view and will be different for each view. However, you can customize toolbars in the Toolbars tab of the Customize dialog (Tools | Customize | Toolbars, screenshot below).

Now, practise moving toolbars around the GUI. Click the handle of a toolbar and drag the toolbar to any desired location in the GUI. (The toolbar handle is indicated by the dotted vertical line at the left of each toolbar; see screenshot below.)

Try dragging a toolbar to the following locations: (i) another line in the toolbar area; (ii) left or right of another toolbar; (iii) the middle of the Main window; (iv) docked to the left or right side of the application window (for this to happen, the grab handle must be placed above the left or right border of the application window).

After you have finished, close the file AddressFirst.xsd.
1.4 Text View Settings

In this section, you will learn how to set up a "pretty-printed" document and how to use bookmarks while editing. When a document is pretty-printed it is displayed in Text View so that each lower level in the XML hierarchy is indented deeper than the previous level (see screenshot below). Bookmarks enable you to mark document positions that you wish to return to quickly.

Pretty-printing

Pretty-printing involves two steps: (i) Setting pretty-printing on and specifying the amount of indentation, and (ii) applying pretty-printing.

1. Open the file CompanyFirst.xml, which is in the C:\Documents and Settings \<username>\My Documents\Altova\XMLSpy2019\Examples\Tutorial folder (and switch to Text View if Text View is not the default starting view of XML documents).
2. In the View section of the Options dialog (Tools | Options | View, screenshot below), check the Use Indentation check box. This switches on pretty-printing with indentation (the default setting). Click OK when done. Note that this setting will apply to all files opened in Text View.
3. Open the Text View Settings dialog (with the **View | Text View Settings** command, *screenshot below*) and in the Tabs pane, decrease the Tab size to 3. Leave the default selection of the Insert Tabs radio button as it is. This will cause the pretty-printing indent to be a tab (rather than spaces) and each tab will have a width of three spaces. Click **OK** when done.

4. Click the menu command **Edit | Pretty-Print**. The document will be pretty-printed with the new tab values.

5. Open the Text View Settings dialog (**View | Text View Settings**) and, in the Visual Aid pane, switch on the end-of-line markers.

6. In Text View, go to the end of any line and delete the end-of-line marker so that the next line jumps up a line.

7. Switch to Grid View and back again to Text View. The document will be pretty-printed, and the the end-of-line marker you deleted will be reinstated.

**Note:** If, in the View section of the Options dialog (**Tools | Options | View**, *screenshot above*), you uncheck the Use Indentations check box and then pretty-print all lines will begin without any indentation.

**Bookmarking**

Bookmarks are placed in a bookmark margin on the left of lines you wish to mark. You can then quickly move up and down through the bookmarks in your document.

1. In the Text View Settings dialog (**View | Text View Settings**, *screenshot below*) ensure that the Bookmarks Margin option in the **Margins** pane is selected. Click **OK** when done.
2. In Text View of the file CompanyFirst.xml, place the cursor anywhere in a line you wish to bookmark, then select the menu command Edit | Insert/Remove Bookmark. The line will be bookmarked and is indicated with a blue bookmark in the bookmark margin (see screenshot below).

3. Create a bookmark on another line in the same way as in Step 2.

4. Press F2 (or the command Edit | Go to Next Bookmark) to go down the document to the next bookmark. Press Shift+F2 (or the command Edit | Go to Previous Bookmark) to go up the document to the previous bookmark. Repeat either or both commands as many times as you like.

5. Place the cursor in one of the bookmarked lines and select the menu command Edit | Insert/Remove Bookmark. The bookmark is removed.

6. Save and close the file. No bookmark information is saved with the file. Reopen the file to
check this.
2  XML Schemas: Basics

An XML Schema describes the structure of an XML document. An XML document can be validated against an XML Schema to check whether it conforms to the requirements specified in the schema. If it does, it is said to be valid; otherwise it is invalid. XML Schemas enable document designers to specify the allowed structure and content of an XML document and to check whether an XML document is valid.

The structure and syntax of an XML Schema document is complex, and being an XML document itself, an XML Schema must be valid according to the rules of the XML Schema specification. In XMLSpy, Schema View enables you to easily build valid XML Schemas by using graphical drag-and-drop techniques. The XML Schema document you construct is also editable in Text View and Grid View, but is much easier to create and modify in Schema View.

Objective

In this section of the tutorial, you will learn how to edit XML Schemas in Schema View. Specifically, you will learn how to do the following:
- Create a new schema file
- Define namespaces for the schema
- Define a basic content model
- Add elements to the content model using context menus and drag-and-drop
- Configure the Content Model View

After you have completed creating the basic schema, you can go to the next section of the tutorial, which teaches you how to work with the more advanced features of XML Schema in XMLSpy. This advanced section is followed by a section about schema navigation and documentation in XMLSpy.

Commands used in this section

In this section of the tutorial, you will use Schema View exclusively. The following commands are used:

- Display Diagram (or Display Content Model View). This icon is located to the left of all global components in Schema Overview. Clicking the icon causes the content model of the associated global component to be displayed.
2.1 Creating a New XML Schema File

To create a new XML Schema file:

1. Select the menu option **File | New**. The Create new document dialog opens.

![Create new document dialog](image)

2. In the dialog, select the xsd (XML Schema v1.0) entry (the document description and the list in the window might vary from that in the screenshot) and confirm with **OK**. An empty schema file appears in the Main Window in Schema View.

3. In the Schema Design toolbar click the **XSD 1.0** mode button (see screenshot below) so that Schema View is in XSD 1.0 editing mode.

![Schema Design toolbar](image)

4. You are prompted to enter the name of the root element.

![Enter name of root element](image)

5. Double-click in the highlighted field and enter **Company**. Confirm with **Enter**. Company is now the root element of this schema and is created as a global element. The view you see in the Main Window (screenshot below) is called the Schema Overview. It provides an overview of the schema by displaying a list of all the global components in the top pane of the Main Window; the bottom pane displays the attributes and identity constraints of the selected global component. (You can view and edit the content model of individual global components by clicking the Display Diagram icon to the left of that global component.)
6. In the Annotations field (ann) of the Company element, enter the description of the element, in this case, Root element.

7. Click the menu option File | Save, and save your XML Schema with any name you like (AddressFirst.xsd, for example).
2.2 Defining Namespaces

XML namespaces are an important issue in XML Schemas and XML documents. An XML Schema document must reference the XML Schema namespace and, optionally, it can define a target namespace for the XML document instance. As the schema designer, you must decide how to define both these namespaces (essentially, with what prefixes.)

In the XML Schema you are creating, you will define a target namespace for XML document instances. (The required reference to the XML Schema namespace is created automatically by XMLSpy when you create a new XML Schema document.)

To create a target namespace:

1. Select the menu option Schema Design | Schema settings. This opens the Schema Settings dialog.

2. Click the Target Namespace radio button, and enter http://my-company.com/namespace. In XMLSpy, the namespace you enter as the target namespace is created as the default namespace of the XML Schema document and displayed in the list of namespaces in the bottom pane of the dialog.

3. Confirm with the OK button.

Note the following:
- The XML Schema namespace is automatically created by XMLSpy and given a prefix of `xs:`.
- When the XML document instance is created, it must have the target namespace defined in the XML Schema for the XML document to be valid.
2.3 **Defining a Content Model**

In the Schema Overview, you have already created a global element called `Company`. This element is to contain one `Address` element and an unlimited number of `Person` elements—its content model. Global components that can have content models are elements, complexTypes, and element groups.

In XMLSpy, the content model of a global component is displayed in the Content Model View of Schema View *(screenshot below)*. To view and edit the content model of a global component, click the Display Diagram icon located to the left of the global component.

In this section, you will create the content model of the `Company` element.

**Creating a basic content model**

To create the content model of the `Company` element:

1. In the Schema Overview, click the Display Diagram icon of the `Company` element. This displays the content model of the `Company` element *(screenshot below)*, which is currently empty. Alternatively, you can double-click the `Company` entry in the Components entry helper to display its content model.

2. A content model consists of **compositors** and **components**. The compositors specify the relationship between two components. At this point of the `Company` content model, you must add a child compositor to the `Company` element in order to add a child element. To add a compositor, right-click the `Company` element. From the context menu that appears, select **Add Child | Sequence**. (Sequence, Choice, and All are the three compositors that can be used in a content model.)
This inserts the Sequence compositor, which defines that the components that follow must appear in the specified sequence.

3. Right-click the Sequence compositor and select **Add Child | Element**. An unnamed element component is added.

4. Enter **Address** as the name of the element, and confirm with **Enter**.

5. Right-click the Sequence compositor again, select **Add Child | Element**. Name the newly created element component **Person**.
You have so far defined a schema which allows for one address and one person per company. We need to increase the number of Person elements.

6. Right-click the Person element, and select Unbounded from the context menu. The Person element in the diagram now shows the number of allowed occurrences: 1 to infinity.

Alternatively, in the Details Entry Helper, you can edit the minOcc and maxOcc fields to specify the allowed number of occurrences, in this case 1 and unbounded, respectively.

Adding additional levels to the content model structure

The basic content model you have created so far contains one level: a child level for the company element which contains the Address and Person elements. Now we will define the content of the Address element so it contains Name, Street, and City elements. This is a second level. Again we need to add a child compositor to the Address element, and then the element components themselves.

Do this as follows:

1. Right-click the Address element to open the context menu, and select Add Child | Sequence. This adds the Sequence compositor.
2. Right-click the Sequence compositor, and select Add Child | Element. Name the newly created element component Name.
Complex types, simple types, and XML Schema data types

Till this point, we have not explicitly defined any element type. Click the Text tab to display the Text View of your schema (listing below). You will notice that whenever a Sequence compositor was inserted, the xs:sequence element was inserted within the xs:complexType element. In short, the Company and Address elements, because they contain child elements, are complex types. A complex type element is one which contains attributes or elements.

```xml
<xs:element name="Company">
  <xs:annotation>
    <xs:documentation>Root element</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Address">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Name"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Person"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Simple type elements, on the other hand, contain only text and have no attributes. Text can be strings, dates, numbers, etc. We want the Name child of Address to contain only text. It is a simple type, the text content of which we want to restrict to a string. We can do this using the XML Schema data type xs:string.

To define the Name element to be of this datatype:

1. Click the Schema tab to return to Schema View.
2. Click the Name element to select it.
3. In the Details Entry Helper, from the dropdown menu of the type combo box, select the xs:string entry.

Note that both minOcc and maxOcc have a value of 1, showing that this element occurs
only once.

The text representation of the Name element is as follows:

```xml
<xsl:element name="Name" type="xs:string"/>
```

**Note:** A simple type element can have any one of several XML Schema data types. In all these cases, the icon indicating text-content appears in the element box.
2.4 Adding Elements with Drag-and-Drop

You have added elements using the context menu that appears when you right-click an element or compositor. You can also create elements using drag-and-drop, which is quicker than using menu commands. In this section, you will add more elements to the definition of the Address element using drag-and-drop, thus completing this definition.

To complete the definition of the Address element using drag-and-drop:

1. Click the Name element of the Address element, hold down the Ctrl key, and drag the element box with the mouse. A small “plus” icon appears in the element box, indicating that you are about to copy the element. A copy of the element together with a connector line also appears, showing where the element will be created.

2. Release the mouse button to create the new element in the Address sequence. If the new element appears at an incorrect location, drag it to a location below the Name element.

3. Double-click in the element box, and type in Street to change the element name.

4. Use the same method to create a third element called City. The content model should now look like this:
The Address element has a sequence of a Name, a Street, and a City element, in that order.
2.5 Configuring the Content Model View

This is a good time to configure the Content Model View. We will configure the Content Model View such that the type of the element is displayed for each element. Do this as follows:

1. Select the Content Model View (click the Content Model View icon) of a component in order to enable the Configure view command.
2. Select the menu option **Schema Design | Configure view**. The Schema Display Configuration dialog appears.

![Schema display configuration dialog](image)

3. Click the **Append** icon (in the **Element** tab) to add a property descriptor line for each element box.
4. From the dropdown menu, select **type** (or double-click in the line and enter "*type*"). This will cause the data type of each element to be displayed in the Content Model View.
5. In the Single Line Settings pane, select **Hide Line If No Value**. This hides the description of the datatype in the element box if the element does not have a datatype (for example, if the element is a complex type).
Notice that the type descriptor line appears for the Name, Street, and City elements, which are simple types of type xs:string, but not for the complex type elements. This is because the Hide Line If No Value toggle is selected.

6. In the Single Line Settings group, select the Always Show Line radio button.
7. Click **OK** to confirm the changes.

Notice that the descriptor line for the data type is always shown—even in element boxes of complex types, where they appear without any value.

Note the following:

- The property descriptor lines are editable, so values you enter in them become part of the element definition.
- The settings you define in the Schema display configuration dialog apply to the schema documentation output as well as the printer output.
Chapter 2.6 Completing the Basic Schema

You have defined the content of the Address element. Now you need to define the content of the Person element. The Person element is to contain the following child elements, all of which are simple types: First, Last, Title, PhoneExt, and Email. All these elements are mandatory except Title (which is optional), and they must occur in the order just specified. All should be of datatype xs:string except PhoneExt, which must be of datatype xs:integer and limited to 2 digits.

To create the content model for Person:

1. Right-click the Person element to open the context menu, and select Add Child | Sequence. This inserts the Sequence compositor.
2. Right-click the Sequence compositor, and select Add Child | Element.
3. Enter First as the name of the element, and press the Tab key. This automatically places the cursor in the type field.
4. Select the xs:string entry from the dropdown list or enter it into the type value field.
5. Use the drag-and-drop method to create four more elements. Name them Last, Title, PhoneExt, and Email, respectively.
Note: You can select multiple elements by holding down the Ctrl key and clicking each of the required elements. This makes it possible to, e.g., copy several elements at once.

Making an element optional

Right-click the Title element and select Optional from the context menu. The frame of the element box changes from solid to dashed; this is a visual indication that an element is optional.

In the Details Entry Helper, you will see that minOcc=0 and maxOcc=1, indicating that the element is optional. Alternatively to using the context menu to make an element optional, you can set minOcc=0 in order to make the element optional.

Limiting the content of an element

To define the PhoneExt element to be of type xs:integer and have a maximum of two digits:

1. Double-click in the type field of the PhoneExt element, and select (or enter) the
The items in the Facets Entry Helper change at this point.

2. In the Facets Entry Helper, double-click in the `maxIncl` field and enter 99. Confirm with Enter.

This defines that all phone extensions up to, and including 99, are valid.

3. Select the menu option `File | Save` to save the changes to the schema.

Note the following:

- Selecting an XML Schema datatype that is a simple type (for example, `xs:string` or `xs:date`), automatically changes the content model to simple in the Details Entry Helper (`content = simple`).
- Adding a compositor to an element (sequence, choice, or all), automatically changes the content model to complex in the Details Entry Helper (`content = complex`).
- The schema described above is available as `AddressFirst.xsd` in the `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2019\Examples\Tutorial` folder of your XMLSpy application folder.
3 XML Schemas: Advanced

Now that you have created a basic schema, we can move forward to a few advanced aspects of schema development.

Objective

In this section, you will learn how to:

- Work with complex types and simple types, which can then be used as the types of schema elements.
- Create global elements and reference them from other elements.
- Create attributes and their properties, including enumerated values.

You will start this section with the basic AddressFirst.xsd schema you created in the first part of this tutorial.

Commands used in this section

In this section of the tutorial, you will use Schema View exclusively. The following commands are used:

- Display Diagram (or Display Content Model View). This icon is located to the left of all global components in Schema Overview. Clicking the icon causes the content model of the associated global component to be displayed.
- Display All Globals. This icon is located at the top left-hand corner of the Content Model View. Clicking the icon switches the view to Schema Overview, which displays all global components.
- Append. The Append icon is located at the top left-hand corner of the Schema Overview. Clicking the icon enables you to add a global component.
3.1 Working with Complex Types and Simple Types

Having defined the content model of an element, you may decide you want to reuse it elsewhere in your schema. The way to do this is by creating that element definition as a global complex type or as a global element. In this section, you will work with global complex types. You will first create a complex type at the global level and then extend it for use in a content model. You will learn about global elements later in this tutorial.

Creating a global complex type

The basic Address element that we defined (containing Name, Street, and City elements) can be reused in various address formats. So let us create this element definition as a complex type, which can be reused.

To create a global complex type:

1. In the Content Model View, right-click the Address element.
2. In the context menu that now appears, select Make Global | Complex type. A global complex type called AddressType is created, and the Address element in the Company content model is assigned this type. The content of the Address element is the content model of AddressType, which is displayed in a yellow box. Notice that the datatype of the Address element is now AddressType.
3. Click the Display All Globals icon. This takes you to the Schema Overview, in which you can view all the global components of the schema.
4. Click the expand icons for the element and complexType entries in the Components entry helper, to see the respective schema constructs. The Schema Overview now displays two global components: the Company element and the complex type AddressType. The Components Entry Helper also displays the AddressType complex type.
5. Click on the Content Model View icon of AddressType to see its content model (screenshot below). Notice the shape of the complex type container.

6. Click the Display All Globals icon to return to the Schema Overview.

**Extending a complex type definition**

We now want to use the global AddressType component to create two kinds of country-specific addresses. For this purpose we will define a new complex type based on the basic AddressType component, and then extend that definition.

Do this as follows:

1. Switch to Schema Overview. (If you are in Content Model View, click the Display All Globals icon.)

2. Click the Append icon at the top left of the component window. The following menu opens:
3. Select **ComplexType** from the menu. A new line appears in the component list, and the cursor is set for you to enter the component name.

4. Enter **US-Address** and confirm with **Enter**. (If you forget to enter the hyphen character "-" and enter a space, the element name will appear in red, signalling an invalid character.)

5. Click the Content Model View icon of **US-Address** to see the content model of the new complex type. The content model is empty (see screenshot below).

6. In the Details entry helper, click the **base** combo box and select the **AddressType** entry.
The Content Model View now displays the `AddressType` content model as the content model of US-Address (screenshot below).

7. Now we can extend the content model of the `US-Address` complex type to take a ZIP Code element. To do this, right-click the `US-Address` component, and, from the context menu that appears, select `Add Child | Sequence`. A new sequence compositor is displayed outside the `AddressType` box (screenshot below). This is a visual indication that this is an extension to the element.
8. Right-click the new sequence compositor and select **Add Child | Element**.
9. Name the newly created element **zip**, and then press the **Tab** key. This places the cursor in the value field of the type descriptor line.
10. Select **xs:positiveInteger** from the dropdown menu that appears, and confirm with **Enter**.

You now have a complex type called **US-Address**, which is based on the complex type **AddressType** and extends it to contain a ZIP code.

**Global simple types**

Just as the complex type **US-Address** is based on the complex type **AddressType**, an element can also be based on a simple type. The advantage is the same as for global complex types: the simple type can be reused. In order to reuse a simple type, the simple type must be defined globally. In this tutorial, you will define a content model for US states as a simple type. This simple type will be used as the basis for another element.
Creating a global simple type

Creating a global simple type consists of appending a new simple type to the list of global components, naming it, and defining its datatype.

To create a global simple type:

1. Switch to Schema Overview. (If you are in Content Model View, click the Display All Globals icon.)
2. Click the Append icon, and in the context menu that appears, select SimpleType.
3. Enter US-State as the name of the newly created simpleType.
4. Press Enter to confirm. The simple type US-State is created and appears in the list of simple types in the Components Entry Helper (Click the expand icon of the simpleType entry to see it).

5. In the Details Entry Helper (screenshot below), place the cursor in the value field of restr and enter xs:string, or select xs:string from the dropdown menu in the restr value field.

This creates a simple type called US-State, which is of datatype xs:string. This global
Using a global simple type in a content model

A global simple type can be used in a content model to define the type of a component. We will use `US-State` to define an element called `State` in the content model of `US-Address`.

Do the following:

1. In Schema Overview, click the Component Model View icon of `US-Address`.
2. Right-click the lower sequence compositor and select `Add Child | Element`.
3. Enter `State` for the element name.
4. Press the Tab key to place the cursor in the value field of the type descriptor line.
5. From the drop-down menu of this combo box, select `US-State`.

The `State` element is now based on the `US-State` simple type.

Creating a second complex type based on `AddressType`

We will now create a global complex type to hold UK addresses. The complex type is based on `AddressType`, and is extended to match the UK address format.

Do the following:

1. In Schema Overview, create a global complex type called `UK-Address`, and base it on `AddressType (base=AddressType)`.
2. In the Content Model View of `UK-Address`, add a `Postcode` element and give it a type
of \texttt{xs:string}. Your \texttt{UK-Address} content model should look like this:

\begin{figure}
\centering
\includegraphics[width=0.7\textwidth]{uk-address-content-model.png}
\end{figure}

\textbf{Note:} In this section you created global simple and complex types, which you then used in content model definitions. The advantage of global types is that they can be reused in multiple definitions.
3.2 Referencing Global Elements

In this section, we will convert the locally defined `Person` element to a global element and reference that global element from within the `Company` element.

1. Click (Display All Globals) to switch to Schema Overview.
2. Click the Display Diagram icon of the `Company` element.
3. Right-click the `Person` element, and select Make Global | Element. A small link arrow icon appears in the `Person` element, showing that this element now references the globally declared `Person` element. In the Details Entry Helper, the isRef check box is now activated.

4. Click the Display All Globals icon to return to Schema Overview. The `Person` element is now listed as a global element. It is also listed in the Components Entry Helper.
5. In the Components Entry Helper, double-click the **Person** element to see the content model of the global **Person** element.

Notice that the global element box does **not** have a link arrow icon. This is because it is the referenced element, not the referencing element. It is the referencing element that has the link arrow icon.

Note the following:

- An element that references a global element must have the same name as the global element it references.
- A global declaration does not describe where a component is to be used in an XML document. It only describes a content model. It is only when a global declaration is referenced from within another component that its location in the XML document is specified.
- A globally declared element can be reused at multiple locations. It differs from a globally declared complex type in that its content model cannot be modified without also modifying the global element itself. If you change the content model of an element that references a global element, then the content model of the global element will also be changed, and, with it, the content model of all other elements that reference that global element.
element.
3.3 Attributes and Attribute Enumerations

In this section, you will learn how to create attributes and enumerations for attributes.

Defining element attributes

1. In the Schema Overview, click the Person element to make it active.
2. Click the Append icon, in the top left of the Attributes/Identity Constraints tab group (in the lower part of the Schema Overview window), and select the Attribute entry.
3. Enter Manager as the attribute name in the Name field.
4. Use the Type combo box to select xs:boolean.
5. Use the Use combo box to select required.
6. Use the same procedure to create a **Programmer** attribute with **Type=xs:boolean** and **Use=optional**.

**Defining enumerations for attributes**

Enumerations are values allowed for a given attribute. If the value of the attribute in the XML instance document is not one of the enumerations specified in the XML Schema, then the document is invalid. We will create enumerations for the **Degree** attribute of the **Person** element.

Do the following:

1. In the Schema Overview, click the **Person** element to make it active.
2. Click the Append icon in the top left of the Attributes window, and select the **Attribute** entry.
3. Enter **Degree** as the attribute name, and select **xs:string** as its type.
4. With the **Degree** attribute selected, in the Facets Entry Helper, click the **Enumerations** tab (see screenshot).
5. In the **Enumerations** tab, click the Append icon.
6. Enter **BA**, and confirm with **Enter**.
7. Use the same procedure to add two more enumerations: **MA** and **PhD**.
8. Click on the Content Model View icon of **Person**.

The previously defined attributes are visible in the Content Model View. Clicking the expand icon displays all the attributes defined for that element. This display mode and the Attributes tab can be toggled by selecting the menu option **Schema Design | Configure view**, and checking and unchecking the **Attributes** check box in the **Show in diagram** pane.

9. Click the Display all Globals icon to return to the Schema Overview.
Saving the completed XML Schema
Before saving your schema file, rename the AddressLast.xsd file that is delivered with XMLSpy to something else (such as AddressLast_original.xsd), so as not to overwrite it. Save the completed schema with any name you like (File | Save as). We recommend that you save it with the name AddressLast.xsd. This is because the XML file you will create in the next part of the tutorial will be based on the AddressLast.xsd schema.
4 XML Schemas: XMLSpy Features

After having completed the XML Schema, we suggest you become familiar with a few navigation shortcuts and learn about the schema documentation that you can generate from within XMLSpy. These are described in the subsections of this section.

Commands used in this section

In this section of the tutorial, you will use Schema View exclusively. The following commands are used:

- Display Diagram (or Display Content Model View). This icon is located to the left of all global components in Schema Overview. Clicking the icon causes the content model of the associated global component to be displayed.
4.1 Schema Navigation

This section shows you how to navigate Schema View efficiently. We suggest that you try out these navigation mechanisms to become familiar with them.

Displaying the content model of a global component

Global components that can have content models are complex types, elements, and element groups. The Content Model View of these components can be opened in the following ways:

- In Schema Overview, click the Display Diagram icon to the left of the component name.
- In either Schema Overview or Content Model View, double-click the element, complex type, or element group in the Components Entry Helper (screenshot below). This displays the content model of that component.

If you double-click any of the other global components (simple type, attribute, attribute group) in the Components Entry Helper, that component will be highlighted in Schema Overview (since such a component would not have a content model).

In the Components Entry Helper, the double-clicking mechanism works in both the Globals and Namespaces tabs.

Going to the definition of a global element from a referencing element

If a content model contains an element that references a global element, you can go directly to the content model of that global element or to any of its contained components by holding down Ctrl and double-clicking the required element.
When the `Last` element is highlighted, all its properties are immediately displayed in the relevant entry helpers and information window.

**Going to the definition of a complex type**

Complex types are often used as the type of some element within a content model. To go directly to the definition of a complex type from within a content model, double-click the name of the complex type in the yellow box (see mouse pointer in screenshot below).

This takes you to the Content Model View of the complex type.
Note: Just as with referenced global elements, you can go directly to an element within the complex type definition by holding down Ctrl and double-clicking the required element in the content model that contains the complex type.
4.2 Schema Documentation

XMLSpy provides detailed documentation of XML Schemas in HTML and Microsoft Word (MS Word) formats. You can select the components and the level of detail you want documented. Related components are hyperlinked in both HTML and MS Word documents. In order to generate MS Word documentation, you must have MS Word installed on your computer (or network).

In this section, we will generate documentation for the AddressLast.xsd XML Schema.

Do the following:

1. Select the menu option Schema design | Generate documentation. This opens the Schema Documentation dialog.

2. For the Output Format option, select HTML, and click OK.

3. In the Save As dialog, select the location where the file is to be saved and give the file a suitable name (say AddressLast.html). Then click the Save button.

The HTML document appears in the Browser View of XMLSpy. Click on a link to go to the corresponding linked component.
### Schema AddressLast.xsd

- **schema location:** `C:\Users\[User]\Documents\Altova\XMLSpy2013\Examples\Tutorial\AddressLast.xsd`
- **attributeFormDefault:** unqualified
- **elementFormDefault:** qualified
- **targetNamespace:** `http://my-company.com/namespace`

#### Elements
- **Complex types:**
  - Company
  - Address
- **Simple types:**
  - AddressType
  - US-State
  - Person
  - UK-Address
  - US-Address

#### element Company

```xml
<xs:element name="Company">
  <xs:annotation>
    <xs:documentation>Root element</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Address" type="AddressType"/>
      <xs:element ref="Person" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The diagram above shows the first page of the schema documentation in HTML form. If components from other schemas have been included, then those schemas are also documented.
The diagram above shows how complex types are documented.
The diagram above shows how elements and simple types are documented.

You can now try out the MS Word output option. The Word document will open in MS Word. To use hyperlinks in the MS Word document, hold down Ctrl while clicking the link.
5 XML Documents

In this section you will learn how to create and work with XML documents in XMLSpy. You will also learn how to use the various intelligent editing features of XMLSpy.

Objective

The objectives of this section are to learn how to do the following:

- Create a new XML document based on the AddressLast.xsd schema.
- Specify the type of an element so as to make an extended content model for that element available to the element during validation.
- Insert elements and attributes and enter content for them in Grid View and Text View using intelligent entry helpers.
- Copy XML data from XMLSpy to Microsoft Excel; add new data in MS Excel; and copy the modified data from MS Excel back to XMLSpy. This functionality is available in the Database/Table View of Grid View.
- Sort XML elements using the sort functionality of Database/Table View.
- Validate the XML document.
- Modify the schema to allow for three-digit phone extensions.

Commands used in this section

In this section of the tutorial, you will mostly use the Grid View and Text View, and in one section Schema View. The following commands are used:

- **File | New.** Creates a new type of XML file.
- **View | Text View.** Switches to Text View.
- **View | Grid View.** Switches to Enhanced Grid View.
- **XML | Table | Display as Table.** Displays multiple occurrences of a single element type at a single hierarchic level as a table. This view of the element is called the Database/Table View (or simply Table View). The icon is used to switch between the Table View and regular Grid View.
- **F7.** Checks for well-formedness.
- **F8.** Validates the XML document against the associated DTD or Schema.
- **Opens the associated DTD or XML Schema file.**
5.1 Creating a New XML File

When you create a new XML file in XMLSpy, you are given the option of basing it on a schema (DTD or XML Schema) or not. In this section you will create a new file that is based on the AddressLast.xsd schema you created earlier in the tutorial.

To create the new XML file:


2. Select the Extensible Markup Language entry (or generic XML document entry) from the dialog, and confirm with OK. A prompt appears, asking if you want to base the XML document on a DTD or Schema.

3. Click the Schema radio button, and confirm with OK. A further dialog appears, asking you to select the schema file your XML document is to be based on.

4. Use the Browse or Window buttons to find the schema file. The Window button lists all files open in XMLSpy and projects. Select AddressLast.xsd (see Tutorial introduction for location), and confirm with OK. An XML document containing the main elements defined by the schema opens in the main window.
5. Click the **Grid** tab to select Grid View.
6. In Grid View, notice the structure of the document. Click on any element to reduce selection to that element. Your document should look something like this:

```
    XML
    └── version 1.0
    └── encoding UTF-8

    Company
    ├── xmlns http://my-company.com/namespace
    ├── xmlns:xsi http://www.w3.org/2001/XMLSchema
    ├── xsi:schemaLocation http://my-company.com/namespace AddressLast.xsd

    Address
    └── Person
```

7. Click on the **icon** next to **Address**, to view the child elements of **Address**. Your document should look like this:

```
    Company
    └── xmlns http://my-company.com/namespace
    ├── xmlns:xsi http://www.w3.org/2001/XMLSchema
    └── xsi:schemaLocation http://my-company.com/namespace AddressLast.xsd

    Address
    └── Name
    └── Street
    └── City

    Person
    └── Manager
```
5.2 Specifying the Type of an Element

The child elements of `Address` are those defined for the global complex type `AddressType` *(the content model of which is defined in the XML Schema `AddressLast.xsd` shown in the Schema View screenshot below)*.

![Diagram of AddressType structure]

We would, however, like to use a specific US or UK address type rather than the generic address type. You will recall that, in the `AddressLast.xsd` schema, we created global complex types for US-Address and UK-Address by extending the `AddressType` complex type. The content model of US-Address is shown below.

![Diagram of US-Address structure]

In the XML document, in order to specify that the `Address` element must conform to one of the extended `Address` types (US-Address or UK-Address) rather than the generic `AddressType`, we must specify the required extended complex type as an attribute of the `Address` element.

Do the following:

1. In the XML document, right-click the `Name` element, and select **Insert | Attribute** from the context menu.
An attribute field is added to the Address element.

2. Ensure that xsi:type is entered as the name of the attribute (screenshot below).

3. Press Tab to move into the next (value) field.

4. Enter US-Address as the value of the attribute.

**Note:** The xsi prefix allows you to use special XML Schema related commands in your XML document instance. Notice that the the namespace for the xsi prefix was automatically added to the document element when you assigned a schema to your XML file. In the above case, you have specified a type for the Address element. See the XML Schema specification for more information.
5.3 Entering Data in Grid View

You can now enter data into your XML document. Do the following:

1. Double-click in the Name value field (or use the arrow keys) and enter **US dependency**. Confirm with Enter.

2. Use the same method to enter a Street and City name (for example, Noble Ave and Dallas).

3. Click the Person element and press Delete to delete the Person element. (We will add it back in the next section of the tutorial.) After you do this, the entire Address element is highlighted.

4. Click on any child element of the Address element to deselect all the child elements of Address except the selected element. Your XML document should look like this:
5.4 Entering Data in Text View

Text View presents the actual data and markup of XML files in an easy-to-follow structural layout, and provides schema-related intelligent editing features. Individual Text View features can be switched on and off in the Text View Settings dialog (View | Text View Settings, screenshot below).

The screenshot below shows the current XML file in Text View with features switched on according to the settings in the dialog above.

On the left are the three margins: (i) the line number margin, (ii) the bookmark margin (containing two blue bookmarks), and (iii) the source folding margin (which allows you to expand and collapse the display of XML elements).

Additionally, visual aids such as indentation guides, end-of-line markers, and whitespace markers can be switched on and off, by checking and unchecking, respectively, their check boxes in the Visual Aid pane of the Text View Settings dialog (see screenshot above). The screenshot above has indentation guides switched on, and shows one indentation guide, at the Address element.

**Note:** The Text View-related pretty-printing and bookmark features were covered in the earlier Text View Settings section of this tutorial.

Editing in Text View

In this section, you will enter and edit data in Text View in order to become familiar with the features of Text View.
Do the following:

1. Select the menu item **View | Text view**, or click on the **Text** tab. You now see the XML document in its text form, with syntax coloring.
2. Place the text cursor after the end tag of the **Address** element, and press **Enter** to add a new line.
3. Enter the less-than angular bracket `<` at this position. A dropdown list of all elements allowed at that point (according to the schema) is displayed. Since only the **Person** element is allowed at this point, it will be the only element displayed in the list.
4. Select the **Person** entry. The **Person** element, as well as its attribute **Manager**, are inserted, with the cursor inside the value-field of the **Manager** attribute.
5. From the dropdown list that pops up for the **Manager** attribute, select **true**.
6. Move the cursor to the end of the line (using the **End** key if you like), and press the space bar. This opens a dropdown list containing a list of attributes allowed at that point. Also, in the Attributes Entry Helper, the available attributes are listed in red. The **Manager** attribute is grayed out because it has already been used.
7. Select **Degree** with the Down arrow key, and press **Enter**. This opens another list box, from which you can select one of the predefined enumerations (**BA**, **MA**, or **PhD**). (Enumerations are values that are allowed by the XML Schema.)
8. Select **BA** with the Down arrow key and confirm with **Enter**. Then move the cursor to the
end of the line (with the End key), and press the space bar. Manager and Degree are now grayed out in the Attributes Entry Helper.

```
</Address>
<Person Manager="true" Degree="BA"
</Company>
```

<table>
<thead>
<tr>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmer</td>
</tr>
<tr>
<td>xsi:type</td>
</tr>
</tbody>
</table>

9. Select Programmer with the Down arrow key and press Enter.

```
</Address>
<Person Manager="true" Degree="BA" Programmer="true"
</Company>
```

10. Enter the letter "t" and press Enter.

11. Move the cursor to the end of the line (with the End key), and enter the greater-than angular bracket >. XMLSpy automatically inserts all the required child elements of Person. (Note that the optional Title element is not inserted.) Each element has start and end tags but no content.

```
  <Address xsi:type="US-Address">
    <Name>US dependency</Name>
    <Street>Noble Ave.</Street>
    <City>Dallas</City>
  </Address>
  <Person Manager="true" Degree="BA" Programmer="false">
    <First></First>
    <Last></Last>
    <PhoneExt></PhoneExt>
    <Email></Email>
  </Person>
</Company>
```

You could now enter the Person data in Text View, but let’s move to Grid View to see the flexibility of moving between views when editing a document.

**Switching to Grid View**

To switch to Grid View, select the menu item View | Grid View, or click the Grid tab. The newly added child elements of Person are highlighted.
Now let us validate the document and correct any errors that the validation finds.
5.5 Validating the Document

XMLSpy provides two evaluations of the XML document:
- A well-formedness check
- A validation check

If either of these checks fails, we will have to modify the document appropriately.

Checking well-formedness

An XML document is well-formed if starting tags match closing tags, elements are nested correctly, there are no misplaced or missing characters (such as an entity without its semi-colon delimiter), etc.

You can do a well-formedness check in any editing view. Let us select Text View. To do a well-formedness check, select the menu option XML | Check well-formedness, or press the F7 key, or click . A message appears in the Messages window at the bottom of the Main Window saying the document is well-formed.

Notice that the output of the Messages window has 9 tabs. The validation output is always displayed in the active tab. Therefore, you can check well-formedness in Tab1 for one schema file and keep the result by switching to Tab2 before validating the next schema document (otherwise Tab1 is overwritten with the validation result ).

Note: This check does not check the structure of the XML file for conformance with the schema. Schema conformance is evaluated in the validity check.

Checking validity

An XML document is valid according to a schema if it conforms to the structure and content specified in that schema.

To check the validity of your XML document, first select Grid View, then select the menu option XML | Validate, or press the F8 key, or click . An error message appears in the Messages window saying the file is not valid. Mandatory elements are expected after the City element in Address. If you check your schema, you will see that the US-Address Complex type (which you have set this Address element to be with its xsi:type attribute) has a content model in which the City element must be followed by a Zip element and a State element.

Fixing the invalid document

The point at which the document becomes invalid is highlighted, in this case the City element.
Now look at the Elements Entry Helper (at top right). Notice that the Zip element is prefixed with an exclamation mark, which indicates that the element is mandatory in the current context.

To fix the validation error:

1. In the Elements Entry Helper, double-click the Zip element. This inserts the Zip element after the City element (we were in the Append tab of the Elements Entry Helper).
2. Press the Tab key, and enter the Zip Code of the State (04812), then confirm with Enter. The Elements Entry Helper now shows that the State element is mandatory (it is prefixed with an exclamation mark). See screenshot below.

3. In the Elements Entry Helper, double-click the State element. Then press Tab and enter the name of the state (Texas). Confirm with Enter. The Elements Entry Helper now contains only grayed-out elements. This shows that there are no more required child elements of Address.
Let us now complete the document (enter data for the Person element) before revalidating.

Do the following:

1. Click the value field of the element First, and enter a first name (say Fred). Then press Enter.

2. In the same way enter data for all the child elements of Person, that is, for Last, PhoneExt, and Email. Note that the value of PhoneExt must be an integer with a maximum value of 99 (since this is the range of allowed PhoneExt values you defined in your schema). Your XML document should then look something like this in Grid View:
3. Click again to check if the document is valid. A message appears in the Messages window stating that the file is valid. The XML document is now valid against its schema.

4. Select the menu option File | Save and give your XML document a suitable name (for example CompanyFirst.xml). Note that the finished tutorial file CompanyFirst.xml is in the Tutorial folder, so you may need to rename it before you give that name to the file you have created.

Note: An XML document does not have to be valid in order to save it. Saving an invalid document causes a prompt to appear warning you that you are about to save an invalid document. You can select Save anyway, if you wish to save the document in its current invalid state.
5.6 Adding Elements and Attributes

At this point, there is only one Person element in the document.

To add a new Person element:

1. Click the gray sidebar to the left of the Address element to collapse the Address element. This clears up some space in the view.
2. Select the entire Person element by clicking on or below the Person element text in Grid View. Notice that the Person element is now available in the Append tab of the Elements Entry Helper.

3. Double-click the Person element in the Elements Entry Helper. A new Person element with all mandatory child elements is appended (screenshot below). Notice that the optional Title child element of Person is not inserted.

4. Switch to Grid View and then press F12 to switch the new Person element from Table View to normal Grid View.
5. Click on the Manager attribute of the new Person element. Take a look at the Attributes Entry Helper. The Manager entry is grayed out because it has already been entered. Also look at the Info Window, which now displays information about the Manager attribute. It is
a required attribute and has therefore been added. The Programmer attribute has not been added.

6. In the Append tab of the Attributes Entry Helper, double-click the Programmer entry. This inserts an empty Programmer attribute after the Manager attribute.

The Programmer attribute is now grayed out in the Attributes Entry Helper.

You could enter content for the Person element in this view, but let's switch to the Database/Table View of Grid View since it is more suited to editing a structure with multiple occurrences, such as Person.
5.7 Editing in Database/Table View

Grid View contains a special view called Database/Table View (hereafter called Table View), which is convenient for editing elements with multiple occurrences. Individual element types can be displayed as a table. When an element type is displayed as a table, its children (attributes and elements) are displayed as columns, and the occurrences themselves are displayed as rows.

To display an element type as a table, you select any one of the element type occurrences and click the Display as Table icon in the toolbar (XML | Table | Display as table). This causes that element type to be displayed as a table. Descendant element types that have multiple occurrences are also displayed as tables. Table View is available in Enhanced Grid View, and can be used to edit any type of XML file (XML, XSD, XSL, etc.).

Advantages of Table View
Table View provides the following advantages:

- You can drag-and-drop column headers to reposition the columns relative to each other. This means that, in the actual XML document, the relative position of child elements or attributes is modified for all the element occurrences that correspond to the rows of the table.

- Tables can be sorted (in ascending or descending order) according to the contents of any column using XML | Table | Ascending Sort or Descending Sort.

- Additional rows (i.e., element occurrences) can be appended or inserted using XML | Table | Insert Row.

- You can copy-and-paste structured data to and from third party products.

- The familiar intelligent editing feature is active in Table View also.

Displaying an element type as a Table
To display the Person element type as a table:

1. In Grid View, select either of the Person elements by clicking on or near the Person text.

2. Select the menu option XML | Table | Display as table, or click the Display as Table icon. Both Person elements are combined into a single table. The element and attribute names are now the column headers, and the element occurrences are the rows of the

```
<table>
<thead>
<tr>
<th>Address</th>
<th>xsd:type=US-Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Manager: true</td>
</tr>
<tr>
<td></td>
<td>Degree: BA</td>
</tr>
<tr>
<td></td>
<td>Programmer: false</td>
</tr>
<tr>
<td></td>
<td>First: Fred</td>
</tr>
<tr>
<td></td>
<td>Last: Smith</td>
</tr>
<tr>
<td></td>
<td>PhoneExt: 22</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:Smith@work.com">Smith@work.com</a></td>
</tr>
</tbody>
</table>
```

Both Person elements are combined into a single table. The element and attribute names are now the column headers, and the element occurrences are the rows of the
3. Select the menu option **View | Optimal widths**, or click the Optimal Widths icon, to optimize the column widths of the table.

**Note:** Table View can be toggled off for individual element types in the document by selecting that table (click the element name in the table) and clicking the Display As Table icon. Note however that child elements which were displayed as tables will continue to be displayed as tables.

### Entering content in Table View

To enter content for the second **Person** element, double-click in each of the table cells in the second row, and enter some data. Note, however, that **PhoneExt** must be an integer up to 99 in order for the file to be valid. The intelligent editing features are active also within cells of a table, so you can select options from dropdown lists where options are available (Boolean content and the enumerations for the **Degree** attribute).

**Note:** The Entry Helpers are active also for the elements and attributes displayed as a table. Double-clicking the **Person** entry in the Elements Entry Helper, for example, would add a new row to the table (i.e., a new occurrence of the **Person** element).

### Copying XML data to and from third party products

You can copy spreadsheet-type data between third party products and XML documents in XMLSpy. This data can be used as XML data in XMLSpy and as data in the native format of the application copied to/from. In this section you will learn how to copy data to and from an Excel data sheet.

Do the following:

1. Click on the row label 1, hold down the **Ctrl** key, and click on row label 2. This selects both rows of the table.
2. Select the menu option **Edit | Copy as Structured text**. This command copies elements to the clipboard as they appear on screen.

3. Switch to Excel and paste (**Ctrl+V**) the XML data in an Excel worksheet.

4. Enter a new row of data in Excel. Make sure that you enter a three digit number for the **PhoneExt** element (say, 444).

5. Mark the table data in Excel, and press **Ctrl+C** to copy the data to the clipboard.

6. Switch back to XMLSpy.

7. Click in the top left **data** cell of the table in XMLSpy, and select **Edit | Paste**.

8. The updated table data is now visible in the table.

9. Change the uppercase boolean values **TRUE** and **FALSE** to lowercase **true** and **false**, respectively, using the menu option **Edit | Replace (Ctrl+H)**.

**Sorting the table by the contents of a column**

A table in Table View can be sorted in ascending or descending order by any of its columns. In this case, we want to sort the **Person** table by last names.

To sort a table by the contents of a column:

1. Select the **Last** column by clicking in its header.
2. Select the menu option **XML | Table | Ascending sort** or click on the Ascending Sort icon. The column, and the whole table with it, are now sorted alphabetically. The column remains highlighted.

The table is sorted not just in the display but also in the underlying XML document. That is, the order of the Person elements is changed so that they are now ordered alphabetically on the content of Last. (Click the Text tab if you wish to see the changes in Text View.)

3. Select the menu option **XML | Validate** or press **F8**. An error message appears indicating that the value ‘444’ is not allowed for a PhoneExt element (see screenshot). The invalid PhoneExt element is highlighted.

Expand "Details" to see that PhoneExt is not valid because it is not less than or equal to the maximum value of 99. You can click on the links in the error message to jump to the spot in the XML file where the error was found. Since the value range we set for phone extension numbers does not cover this extension number, we have to modify the XML Schema so that this number is valid. You will do this in the next section.
5.8 Modifying the Schema

Since two-digit phone extension numbers do not cover all likely numbers, let's extend the range of valid values to cover three-digit numbers. We therefore need to modify the XML Schema. You can open and modify the XML Schema without having to close your XML document.

Do the following:

1. Select the menu option **DTD/Schema | Go to definition** or click the Go To Definition icon. The associated schema, in this case *AddressLast.xsd* is opened. Switch to Schema View (screenshot below) if you need to. (By default an XSD schema file will open in Schema View. The default view for every filetype, however, can be changed in the File Types section of the Options dialog (Tools | Options).)

![Schema Overview Diagram](image)

2. In Schema Overview, click the Display Diagram icon of the global **Person** element. In the Content Model View of the **Person** element, select the **PhoneExt** element. The facet data in the Facets tab is displayed.

![Content Model View](image)

3. In the Facets tab, double-click the **maxIncl** value field, change the value *99* to *999*, and confirm with **Enter**.
4. Save the schema document.
5. Press **Ctrl+Tab** to switch back to the XML document.
6. Click to revalidate the XML document.

A message that the file is valid appears in the Validation window. The XML document now conforms to the modified schema.

7. Select the menu option **File | Save As...** and save the file as **CompanyLast.xml**. (Remember to rename the original CompanyLast.xml file that is delivered with XMLSpy to something else, like **CompanyLast_orig.xml**).

**Note:** The CompanyLast.xml file delivered with XMLSpy is in the in the Tutorial folder.
6  XSLT Transformations

Objective
To generate an HTML file from the XML file using an XSL stylesheets to transform the XML file. You should note that a "transformation" does not change the XML file into anything else; instead, a new output file is generated. The word "transformation" is a convention.

Method
The method used to carry out the transformation is as follows:

- Assign a predefined XSL file, Company.xsl, to the XML document.
- Execute the transformation within the XMLSpy interface using one of the two built-in Altova XSLT engines. (See note below.)

Commands used in this section
The following XMLSpy commands are used in this section:

- XSL/XQuery | Assign XSL, which assigns an XSL file to the active XML document.
- XSL/XQuery | Go to XSL, opens the XSL file referenced by the active XML document.
- XSL/XQuery | XSL Transformation (F10), or the toolbar icon, transforms the active XML document using the XSL stylesheet assigned to the XML file. If an XSL file has not been assigned then you will be prompted for one when you select this command.

Note: XMLSpy has built-in XSLT engines for XSLT 1.0, 2.0, and 3.0. The correct engine is automatically selected by XMLSpy on the basis of the version attribute in the xsl:stylesheet or xsl:transform element. In this tutorial transformation, we use XSLT 1.0 stylesheets. The XSLT 1.0 Engine will automatically be selected for transformations with these stylesheets when the XSL Transformation command is invoked.
6.1 Assigning an XSLT File

To assign an XSL file to the CompanyLast.xml file:

1. Click the CompanyLast.xml tab in the main window so that CompanyLast.xml becomes the active document, and switch to Text View.
2. Select the menu command XSL/XQuery | Assign XSL.
3. Click the Browse button, and select the Company.xsl file from the Tutorial folder. In the dialog, you can activate the option Make Path Relative to CompanyLast.xml if you wish to make the path to the XSL file (in the XML document) relative.
4. Click OK to assign the XSL file to the XML document.
5. Switch to Grid View to see the assignment (screenshot below).

An XML-stylesheet processing instruction is inserted in the XML document that references the XSL file. If you activated the Make Path Relative to CompanyLast.xml check box, then the path is relative; otherwise absolute (as in the screenshot above).
6.2 Transforming the XML File

To transform the XML document using the XSL file you have assigned to it:

1. Ensure that the XML file is the active document.
2. Select the menu option XSL/XQuery | XSL Transformation (F10) or click the icon. This starts the transformation using the XSL stylesheet referenced in the XML document. (Since the Company.xsl file is an XSLT 1.0 document, the built-in Altova XSLT 1.0 Engine is automatically selected for the transformation.) The output document is displayed in Browser View; it has the name XSL Output.html. (If the HTML output file is not generated, ensure that, in the XSL section of the Options dialog (Tools | Options), the default file extension of the output file has been set to .html.) The HTML document shows the Company data in one block down the left, and the Person data in tabular form below.

![Your Company](image)

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>Ext,</th>
<th>E-Mail</th>
<th>Manager</th>
<th>Degree</th>
<th>Programmer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alfred</td>
<td>Aldrich</td>
<td>33</td>
<td>Aldrich@work</td>
<td>false</td>
<td>MA</td>
<td>true</td>
</tr>
<tr>
<td>Colin</td>
<td>Coletti</td>
<td>444</td>
<td><a href="mailto:Coletti@work.com">Coletti@work.com</a></td>
<td>true</td>
<td>Ph.D</td>
<td>false</td>
</tr>
<tr>
<td>Fred</td>
<td>Smith</td>
<td>22</td>
<td><a href="mailto:Smith@work.com">Smith@work.com</a></td>
<td>true</td>
<td>BA</td>
<td>false</td>
</tr>
</tbody>
</table>

Note: Should you only see a table header and no table data in the output file, make sure that you have defined the target namespace for your schema as detailed in Defining your own namespace at the beginning of the tutorial. The namespace must be identical in all three files (Schema, XML, and XSL).
6.3 Modifying the XSL File

You can change the output by modifying the XSL document. For example, let's change the background-color of the table in the HTML output from lime to yellow.

Do the following:

1. Click the CompanyLast.xml tab to make it the active document, and make sure you are in Grid View.
2. Select the menu option XSL/XQuery | Go to XSL. The command opens the Company.xsl file referenced in the XML document.

3. Find the line `<table border="1" bgcolor="lime">`, and change the entry `bgcolor="lime"` to `bgcolor="yellow"`.
4. Select the menu option File | Save to save the changes made to the XSL file.
5. Click the CompanyLast.xml tab to make the XML file active, and select XSL/XQuery | XSL Transformation, or press F10. A new XSL Output.html file appears in the XMLSpy GUI in Browser View. The background color of the table is yellow.
# Your Company

Name: US dependency  
Street: Noble Ave  
City: Dallas  
State: Texas  
Zip: 04812

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>Ext.</th>
<th>E-Mail</th>
<th>Manager</th>
<th>Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alfred</td>
<td>Aldrich</td>
<td>33</td>
<td><a href="mailto:Aldrich@work.com">Aldrich@work.com</a></td>
<td>false</td>
<td>MA</td>
</tr>
<tr>
<td>Colin</td>
<td>Coletti</td>
<td>444</td>
<td><a href="mailto:Coletti@work.com">Coletti@work.com</a></td>
<td>true</td>
<td>Ph.D</td>
</tr>
<tr>
<td>Fred</td>
<td>Smith</td>
<td>22</td>
<td><a href="mailto:Smith@work.com">Smith@work.com</a></td>
<td>true</td>
<td>BA</td>
</tr>
</tbody>
</table>

6. Select the menu option **File | Save**, and save the document as **Company.html**.
7 Project Management

This section introduces you to the project management features of XMLSpy. After learning about the benefits of organizing your XML files into projects, you will organize the files you have just created into a simple project.
7.1 Benefits of Projects

The benefits of organizing your XML files into projects are listed below.

- Files and URLs can be grouped into folders by common extension or any other criteria.
- Batch processing can be applied to specific folders or the project as a whole.
- A DTD or XML Schema can be assigned to specific folders, allowing validation of the files in that folder.
- XSLT files can be assigned to specific folders, allowing transformations of the XML files in that folder using the assigned XSLT.
- The destination folders of XSL transformation files can be specified for the folder as a whole.

All the above project settings can be defined using the menu option Project | Project Properties.

In the next section, you will create a project using the Project menu. Additionally, the following advanced project features are available:

- XML files can be placed under source control using the menu option Project | Source control | Add to source control. (Please see the Source Control section in the online help for more information.)
- Personal, network and web folders can be added to projects, allowing batch validation.
### 7.2 Building a Project

Having come to this point in the tutorial, you will have a number of tutorial-related files open in the Main Window. You can group these files into a tutorial project. First you create a new project and then you add the tutorial files into the appropriate sub-folders of the project.

#### Creating a basic project

To create a new project:

1. Select the menu option **Project | New Project**. A new project folder called New Project is created in the Project Window. The new project contains empty folders for typical categories of XML files in a project (screenshot below).

![Project Window Screenshot](image)

2. Click the CompanyLast.xml tab to make the CompanyLast.xml file the active file in the Main Window.
3. Select the menu option **Project | Add active and related files to project**. Two files are added to the project: CompanyLast.xml and AddressLast.xsd. Note that files referenced with Processing instructions (such as XSLT files) do not qualify as related files.
4. Select the menu option **Project | Save Project** and save the project under the name Tutorial.

#### Adding files to the project

You can add other files to the project as well. Do this as follows:

1. Click on any open XML file (with the .xml file extension) other than CompanyLast.xml to make that XML file the active file. (If no other XML file is open, open one or create a new XML file.)
2. Select the menu option **Project | Add active file to project**. The XML file is added to the XML Files folder on the basis of its .xml file type.
3. In the same way, add an HTML file and XSD file (say, the Company.html and AddressFirst.xsd files) to the project. These files will be added to the HTML Files folder and DTD/Schemas folder, respectively.
4. Save the project, either by selecting the menu option **Project | Save Project** or by selecting any file or folder in the Project Window and clicking the Save icon in the toolbar (or **File | Save**).

**Note:** Alternatively, you can right-click a project folder and select **Add Active File** to add the active file to that specific folder.
Other useful commands

Here are some other commonly used project commands:

- To add a new folder to a project, select **Project | Add Project folder to Project**, and insert the name of the project folder.
- To delete a folder from a project, right-click the folder and select **Delete** from the context menu.
  To delete a file from a project, select the file and press the **Delete** key.
8 That's It

If you have come this far congratulations, and thank you!

We hope that this tutorial has been helpful in introducing you to the basics of XMLSpy. If you need more information please use the context-sensitive online help system, or print out the PDF version of the tutorial, which is available as tutorial.pdf in your XMLSpy application folder.
User Guide and Reference

Following introductory sections that list new features in recent releases and general information, the User Guide and Reference starts with a description of XMLSpy’s multiple editing views. The sections that follow describe the functionality that XMLSpy offers for working with various XML and XML-related technologies. Each section provides an overview of the functionality available for the technology being described. The User Guide part concludes with descriptions of application-wide features, such as Altova Global Resources and XMLSpy projects. The final section is a reference section that describes XMLSpy menu commands.

The User Guide and Reference is organized into the following sections:

- New Features
- General Information
- Interface and Environment
- Editing Views
- XML
- DTDs and XML Schemas
- XSLT
- XQuery
- XSLT/XQuery Debugger and Profiler
- Authentic
- HTML and CSS
- JSON and JSON Schema
- HTTP
- Office Open XML, ZIP and EPUB
- Databases
- Altova Global Resources
- Projects
- RaptorXML Server
- File/Directory Comparisons
- Source Control
- XMLSpy in Visual Studio
- XMLSpy in Eclipse
- Menu Commands
1 New Features 2019

Version 2019 Release 3
Features that are new in XMLSpy Version 2019r3 are listed below.

- Text View has been enhanced with Split Views. This provides you with two views of a document that you are editing, which enables you to see different parts of a long document side-by-side—instead of having to scroll a single view to refer to another part of the document.
- Support for the newer JSON schema versions: draft-06 and draft-07. See the JSON section of the documentation.
- The Options dialog (Tools | Options) provides an option to set the Java Virtual Machine path, which overrides the automatically detected path.
- Support for Eclipse has been extended to Eclipse 4.9 and 4.10.

Version 2019
Features that are new in XMLSpy Version 2019 are listed below.

- Integration with Eclipse 4.8.
1.1 Version 2018

Version 2018 Release 2

Features that are new in XMLSpy Version 2018r2 are listed below.

- In the HTTP output window, you can import an HTTP request from a WSDL 1.1 or WSDL 2.0 file. This is in addition to being able to import a request from a WADL file.
- XPath/XQuery 3.1 support for JSON has been implemented in JSON transformations and JSON querying: (i) the XPath/XQuery output window has been enhanced to run queries on JSON documents, (ii) JSON documents can be transformed with XSLT 3.0 and XQuery 3.0 documents; (iii) the XSLT and XQuery Debugger has been enhanced to debug JSON files.
- XPath intelligent editing support has been added for XSLT 3.0 value templates in Text View.
- Text View editing has been enhanced (i) with more syntax-related text formatting options (visible whitespace, etc), and (ii) wider applicability (to additional languages).
- Editing support for the following file types has been added: CSS, Python, C-style languages. Conformance of file types to languages can be set in the File Types section of the Options dialog.
- The Options dialog (Tools | Options) has been restructured and redesigned for a better overview of the application’s options and for more user-friendliness.
- Network proxy settings can be customized via any one of multiple available options.

Version 2018

Features that are new in XMLSpy Version 2018 are listed below.

- A new HTTP output window, in which you can create an HTTP request, send it to the target URL, and receive and check the response.
- Significant speed improvements in the XSLT engine.
- The Options dialog has been reorganized so that different groups of options are organized into a tree structure. This provides a better overview of the different groups.
- Support for XBRL Versioning 1.0.
- Support for Sybase 16, PostgreSQL 9.6, MySQL 5.7
- Integration with Eclipse 4.7
- Support for FOP 2.2
1.2 Version 2017

Version 2017 Release 3

Features that are new in XMLSpy Version 2017r3 are listed below.

- **JSON5 support has been added.** The functionality includes auto-completion, validation against JSON schema, and conversion to/from XML.
- **Support for JSON schema definitions has been extended** to allow the inclusion of definitions from a container named `resourceDefinitions`, which is in addition to the standard JSON container named `definitions`.
- Text View: The **validation errors** of different document types (XML, JSON, etc) are indicated in the line-numbering margin and scroll bars; quick fixes for these errors, when available, are shown in the document itself.
- Text View: The **navigation** and **Find and Replace** functions have been enhanced.

Version 2017

Features that are new in XMLSpy Version 2017 are listed below.

- Text View: **Text highlighting** shows all matches of the currently selected text, together with meta information about the matches.
- Text View: The **scroll bar** contains information about the document size, current position, size of a text selection relative to document size.
- Text View: Easier usage of, and additional functionality in, the **Find and Replace functions**.
1.3 Version 2016

Version 2016 Release 2

Features that are new in XMLSpy Version 2016r2 are listed below.

- XPath/XQuery debugging functionality has been added to the XPath/XQuery Window.
- JSON schema documentation can be generated automatically in HTML, RTF, and MS Word formats.
- JSON instance files containing JSON data that is structured on a JSON schema can be generated from that JSON schema.
- Conversion of XML instances to/from JSON instances has been enhanced so that users can select conversion options.
- Conversion of XML schemas to/from JSON schemas has been added. Users can select conversion options.
- When a JSON instance is active you can go to the JSON schema and JSON definition by clicking the respective commands in the DTD/Schema menu.
- When editing the names of elements in XML documents in Text View, the names are highlighted in different colors according to whether the names in the start and end tags match or not. This serves as a visual editing aid.
- Source folding in XML, XQuery, JSON, and CSS documents in Text View, as well as of nodes in the XPath/XQuery Window, has been enhanced to display ellipses for collapsed nodes. Placing the mouse cursor over an ellipsis displays the content of the collapsed node in a popup.
- Eclipse support has been extended to Eclipse 4.5.

Version 2016

Features that are new in XMLSpy Version 2016 are listed below.

- The new JSON Schema View is introduced. This enables the creation and editing of JSON schemas graphically.
- Generation of JSON schemas from JSON instances.
- JSON document validation: Provides validation of JSON instances and JSON schemas.
- Editing of JSON documents in Text View and Grid View has been enhanced. A new auto-completion feature (based on information obtained from the associated JSON schema) enables the quick and accurate creation of instance and schema documents.
- XSLT transformation using Altova's RaptorXML Server from within XMLSpy. This enables you to customize XSLT high-speed transformations with the help of Raptor's wide range of options.
- XQuery execution and updates using Altova's RaptorXML Server from within XMLSpy. This enables you to customize XQuery executions with Raptor's many options.
1.4 Version 2015

Version 2015 Release 4
Features that are new in XMLSpy Version 2015r4 are listed below.

- Validation of XML and XBRL documents using Altova's RaptorXML Server from within XMLSpy. This enables you to customize validations with the help of Raptor's validation options.

Version 2015 Release 3
Features that are new in XMLSpy Version 2015r3 are listed below.

- Support for XPath and XQuery Functions and Operators 3.1 Candidate Recommendation of 18 December 2014 in the XSLT and XQuery engines of XMLSpy, and in the XPath/XQuery Builder and Evaluator.
- Enhancements to the XPath/XQuery Builder and Evaluator include improved window layout, and interactive functions-and-operators information in popups.
- New Altova XPath/XQuery extension functions for geolocation information.

Version 2015
Features that are new in XMLSpy Version 2015 are listed below.

- Support for XQuery Update Facility: XMLSpy recognizes the XQuery Update document type, and processes such documents so that target XML documents are updated by expressions in the XQuery Update document. This enables only specific parts of documents to be updated, obviating the need to regenerate entire target documents.
- XQuery Update preview: XQuery Update expressions can be entered in the XPath/XQuery output window, and updates can be previewed. These "pending updates" can then be applied or rejected.
2 General Information

This section contains the following general information:

- **Windows File Paths** for the My Document and XMLSpy application folders on different versions of Windows.
- **About RaptorXML Server**, which is Altova’s standalone server-based XML processor; it provides a wide range of processing options as well as customizable processing for larger and faster job execution.
2.1 Windows File Paths

File paths in Windows 7, Windows 8, and Windows 10

File paths given in this documentation will not be the same for all operating systems. You should note the following correspondences:

- **(My) Documents folder**: Located by default at the following locations. Example files are located in a sub-folder of this folder.

<table>
<thead>
<tr>
<th>Windows 7/8/10</th>
<th>C:\Users&lt;username&gt;\Documents</th>
</tr>
</thead>
</table>

- **Application folder**: The Application folder is the folder where your Altova application is located. The path to the Application folder is, by default, the following.

<table>
<thead>
<tr>
<th>Windows 7/8/10</th>
<th>C:\Program Files\Altova\</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit version on 64-bit OS</td>
<td>C:\Program Files (x86)\Altova\</td>
</tr>
</tbody>
</table>

**Note**: XMLSpy is also supported on Windows Server 2008 R2 SP1 with Platform Update or newer.
2.2 About RaptorXML Server

Altova RaptorXML(+)XBRL Server (also called Raptor or RaptorXML for short) is Altova’s third-generation, hyper-fast XML (and XBRL) processor. It has been built to optimally utilize the latest standards and parallel computing environments. It can be used on multiple platforms, and 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:

- RaptorXML Server, which can be accessed over a network and can transform multiple files at a time.
- RaptorXML+XBRL Server edition, which can be accessed over a network, can transform multiple files at a time, and additionally supports XBRL validation.

RaptorXML can be run from the command line and has interfaces for COM, Java, .NET, and Python. A Raptor server can also be run from within the XMLSpy interface.

Altova website: XML validation server, XML validator
3 Interface and Environment

This section describes:

- The application GUI, and
- The application environment.

The GUI section starts off by presenting an overview of the GUI and then goes on to describe each of the various GUI windows in detail. It also shows you how to re-size, move, and otherwise work with the windows and the GUI.

The Application Environment section points out the various settings that control how files are displayed and can be edited. It also explains how and where you can customize your application. In this section, you will learn where important example and tutorial files have been installed on your machine, and, later in the section, you are linked to the Altova website, where you can explore the feature matrix of your application, learn about the multiple formats of your user manual, find out about the various support options available to you, and discover other products in the Altova range.
3.1 The Graphical User Interface (GUI)

The Graphical User Interface (GUI) consists of a Main Window and several sidebars (see illustration below). By default, the sidebars are located around the Main Window and are organized into the following groups:

- Project Window
- Info Window
- Entry Helpers: Elements, Attributes, Entities, etc (depending upon the type of document currently active)
- Output Windows: Messages, XPath, XSL Outline, HTTP, Find in Files, Find in Schemas

The main window and sidebars are described in the sub-sections of this section.

The GUI also contains a menu bar, status bar, and toolbars, all of which are described in a subsection of this section.

Switching on and off the display of sidebars

Sidebar groups (Project Window, Info Window, Entry Helpers, Output Windows) can be displayed or hidden by toggling them on and off via the commands in the Window menu. A displayed sidebar (or a group of tabbed sidebars) can also be hidden by right-clicking the title bar of the displayed sidebar (or tabbed-sidebar group) and selecting the command Hide.

Floating and docking the sidebars

An individual sidebar window can either float free of the GUI or be docked within the GUI. When a floating window is docked, it docks into its last docked position. A window can also be docked as
a tab within another window.

A window can be made to float or dock using one of the following methods:

- Right-click the title bar of a window and choose the required command (Floating or Docking).
- Double-click the title bar of the window. If docked, the window will now float. If floating, the window will now dock in the last position in which it was docked.
- Drag the window (using its title bar as a handle) out of its docked position so that it floats. Drag a floating window (by its title bar) to the location where it is to be docked. Two sets of blue arrows appear. The outer set of four arrows enables docking relative to the application window (along the top, right, bottom, or left edge of the GUI). The inner set of arrows enables docking relative to the window over which the cursor is currently placed. Dropping a dragged window on the button in the center of the inner set of arrows (or on the title bar of a window) docks the dragged window as a tabbed window within the window in which it is dropped.

To float a tabbed window, double-click its tab. To drag a tabbed window out of a group of tabbed windows, drag its tab.

**Auto-hiding sidebars**

The Auto-hide feature enables you to minimize docked sidebars to buttons along the edges of the application window. This gives you more screen space for the Main Window and other sidebars. Scrolling over a minimized sidebar rolls out that sidebar.

To auto-hide and restore sidebars click the drawing pin icon in the title bar of the sidebar window (or right-click the title bar and select Auto-Hide).

### 3.1.1 Main Window

The Main Window (screenshot below) is where you view and edit documents.

Files in the Main Window
Any number of files can be opened and edited at once.

Each open document has its own window and a tab (containing the document's file name) at the bottom of the Main Window. To make an open document active, click its tab.

If several files are open, some document tabs might not be visible for lack of space in the document tabs bar. Document tabs can be brought into view by: (i) using the scroll buttons at the right of the document tab bar, or (ii) selecting the required document from the list at the bottom of the Window menu.

When the active document is maximized, its Minimize, Restore, and Close buttons are located at the right side of the Menu Bar. When a document is cascaded, tiled, or minimized, the Maximize, Restore, and Close buttons are located in the title bar of the document window.

When you maximize one file, all open files are maximized.

Open files can be cascaded or tiled using commands in the Window menu.

You can also activate open files in the sequence in which they were opened by using Ctrl+Tab or Ctrl+F6.

Right-clicking a document tab opens a context-menu with a selection of File commands, such as Print and Close.

Views in the Main Window

The active document can be displayed and edited in multiple views. The available views are displayed in a bar above the document tabs (see illustration above), and the active view is highlighted. A view is selected by clicking the required view button or by using the commands in the View menu.

The available views are either editing or browser views:

- **Text View**: An editing view with syntax-coloring for working directly with document code.
- **Grid View**: For structured editing. The document is displayed as a structured grid that can be manipulated graphically. This view also contains an embedded Table view which shows repeating elements in a tabular format.
- **Schema View**: For viewing and editing XML Schemas.
- **Authentic View**: For editing XML documents that are based on StyleVision Power Stylesheets in a graphical interface.
- **Browser View**: An integrated browser view that supports both CSS and XSL stylesheets.

**Note**: The default view for individual file extensions can be customized in the Tools | Options dialog: in the Default View pane of the File Types tab.

### 3.1.2 Project Window

A project is a collection of files that are related to each other in some way you determine. For example, in the screenshot below, a project named Examples collects the files for various examples in separate example folders, each of which can be further organized into sub-folders.

Within the Examples project, for instance, the OrgChart example folder is further organized into sub-folders for XML, XSL, and Schema files.
Projects thus enable you to gather together files that are used together and to access them quicker. Additionally, you can define schemas and XSLT files for individual folders, thus enabling the batch processing of files in a folder.

**Project operations**

Commands for folder operations are available in the **Project** menu, and some commands are available in the context menus of the project and its folders (right-click to access).

- One project is open at a time in the Project Window. When a new project is created or an existing project opened, it replaces the project currently open in the Project Window.
- After changes have been made to a project, the project must be saved (by clicking the **Project | Save Project** command). A project with unsaved changes is indicated with an asterisk next to its name (*see screenshot above*).
- The project has a tree structure composed of folders, files, and other resources. Such resources can be added at any level and to an unlimited depth.
- Project folders are *semantic* folders that represent a logical grouping of files. They do not need to correspond to any hierarchical organization of files on your hard disk.
- Folders can correspond to, and have a direct relationship to, physical directories on your file system. We call such folders *external folders*, and they are indicated in the Project Window by a yellow folder icon (as opposed to normal project folders, which are green). External project folders must be explicitly synchronized by using the **Refresh** command.
- A folder can contain an arbitrary mix of file-types. Alternatively, you can define file-type extensions for each folder (in the Properties dialog of that folder) to keep common files in one convenient place. When a file is added to the parent folder, it is automatically added to the sub-folder that has been defined to contain files of that file extension.
- In the Project Window, a folder can be dragged to another folder or to another location within the same folder, while a file can be dragged to another folder but cannot be moved within the same folder (within which files are arranged alphabetically). Additionally, files
and folders can be dragged from Windows File Explorer to the Project Window.

- Each folder has a set of properties that are defined in the Properties dialog of that folder. These properties include file extensions for the folder, the schema by which to validate XML files, the XSLT file with which to transform XML files, etc.
- Batch processing of files in a folder is done by right-clicking the folder and selecting the relevant command from the context menu (for example, Validate XML or Check Well-Formedness).

For a more detailed description of projects, see the section Projects.

Note: The display of the Project Window can be turned on and off in the Window menu.

3.1.3 Info Window

The Info Window (screenshot below) shows information about the element or attribute in which the cursor is currently positioned. Information is available in the Info Window in Text View, Grid View, and Authentic View.

![Info Window screenshot]

The display of the Info Window can be turned on and off in the Window menu.

Note the following points:

- When an XSLT document is active, additional XSLT-specific information and commands are available in the XSLT tab of the Info window. How to read the information and use the commands in the XSLT tab is explained in the section XSLT and XQ | XSLT | XSL Outline.

3.1.4 Entry Helpers

Entry helpers are an intelligent editing feature that helps you to create valid XML documents quickly. When you are editing a document, the entry helpers display structural editing options according to the current location of the cursor. The entry helpers get the required information from the underlying DTD, XML Schema, and/or StyleVision Power Stylesheet, etc. If, for example, you are editing an XML data document, then the elements, attributes, and entities that can be inserted at the current cursor position are displayed in the relevant entry helpers windows.

The entry helpers that are available depend upon:

1. The kind of document being edited. For example, XML documents will have different entry helpers than XQuery documents: elements, attributes, and entities entry helpers in the former case, but XQuery keywords, variables, and functions entry helpers in the latter case. The available entry helpers for each document type are described in the description
of that document type in the User Manual section of this documentation.

2. The current view. Since the editing mechanisms in the different views are different, the entry helpers are designed so as to be compatible with the editing mechanism of the current view. For example: In Text View, an element can only be inserted at the cursor location point, so the entry helper is designed to insert an element when the element is double-clicked. But in Grid View, an element can be inserted before the selected node, appended after it, or added as a child node, so the Elements entry helper in Grid View has three tabs for Insert, Append, and Add as Child, with each tab containing the elements available for that particular operation.

A general description of entry helpers in each type of view is given in Editing Views. Further document-type-related differences within a view are noted in the description of the individual document types, for example XML entry helpers and XQuery entry helpers.

Note the following:

- You can turn the display of entry helpers on or off with the menu option Window | Entry Helpers.
- In Visual Studio .NET, entry helper windows have a prefix that is the application name.

3.1.5 Output Window: Messages

The Messages Window displays messages about actions carried out in XMLSpy as well as errors and other output. For example, if an XML, XML Schema, DTD, or XQuery document is validated and is valid, a successful validation message (screenshot below) is displayed in the Messages Window:

![Messages Window](image)

Otherwise, a message that describes the error (screenshot below) is displayed. Notice in the screenshot below that there are links (black link text) to nodes and node content in the XML document, as well as links (blue link text) to the sections in the relevant specification on the Internet that describe the rule in question. Clicking the purple Def buttons opens the relevant schema definition in Schema View.
The Messages Window is enabled in all views, but clicking a link to content in an XML document highlights that node in the XML document in Text View. However, when an XML Schema has been validated in Schema View, clicking a Def button does not change the view.

XML Validation smart fixes
Based on information in the schema, options for a smart fix are also suggested if validation was carried out in Text View or Grid View. To view a list of smart fix options, click the Show Smart Fix button (see screenshot above). A pane with suggested smart fix options appears in the Messages window (screenshot below).

Note that errors in the Messages window are displayed one at a time. Also, errors of well-formedness (such as mismatched start and end tags), if such exist, are displayed prior to validation errors being displayed. So the Show Smart Fix button will be enabled only when a validation error is reached (after all well-formedness errors have been corrected).

In the Smart Fix pane, select one of the suggested smart fixes and click either the Fix + Validate button or the Fix button (see screenshot above). The invalid text in the XML document will be replaced with the selected smart fix. Alternatively, you can double-click the smart fix you want. This action either fixes, or fixes and validates, according to the option selected in the dropdown Options list (see screenshot above). The Fix + Validate command is useful because when another validation is carried out after the fix it will pick up further validation errors if there are any.

To hide the Smart Fix pane, click the Hide Smart Fix button (see screenshot above). For more details, see the section Editing Views | Schema View | Validation and Smart Fixes.

Validating folders and files in the Project window
The Validate command (in the XML menu) is normally applied to the active document. But you
can also apply the command to a file, folder, or group of files in the active project. Select the
required file or folder in the Project Window (by clicking on it), and click XML | Validate XML or
F8. Invalid files in a project will be opened and made active in the Main Window, and the File is
not valid error message will be displayed.

Note: You can also carry out the well-formedness check (Check Well-Formedness or F7) in
the Project window.

3.1.6 Output Window: XPath/XQuery

The XPath/XQuery Window (screenshots below) enables you to evaluate and debug XPath and
XQuery expressions with respect to XML or JSON* documents. The window can be used in two
modes:

- Evaluate Mode, in which an XPath or XQuery expression is evaluated with respect to one
  or more XML documents
- Debug Mode, in which you can debug an XPath/XQuery expression as it applies to the
currently active XML document. You can set breakpoints and tracepoints, and go step-by-
step through the evaluation

To switch between the two modes, select/deselect the Debug Mode button (located in the left-
hand corner of the window's toolbar; see screenshots below). How to use the two modes is
described in the sub-sections of this section. For information about the syntax and support of
XPath/XQuery expressions in the XPath/XQuery Window, see XPath and XQuery Specification
Information.

Note: Switching to Debug Mode button is enabled only when the Current File option is selected
in the Where options list (see screenshot below)

* Features that enable JSON queries were introduced in XPath/XQuery 3.1. See JSON
Transformations with XSLT/XQuery.

Buttons in this section

- Debug Mode
- Horizontal/Vertical Layout

Horizontal and vertical layouts

In the right-hand corner of the toolbar (of both modes, Evaluate and Debug) is a button (highlighted
in the screenshot below) that enables you to switch between a horizontal and a vertical layout.
You can switch layouts at any time. The screenshot below shows the horizontal layout, which is
useful in cases where the result has lines that have a large horizontal extent.
The vertical layout *(screenshot below)* is useful when the XPath/XQuery expression (in the left-hand pane) spans multiple lines.

**Multiple tabs**

The XPath/XQuery Window has nine tabs, enabling you to work with nine different expressions at a time. After you enter an expression in a tab, the mode of the tab is saved. If an expression is entered in a tab that is in *Evaluate Mode*, then the expression and evaluation settings of the tab are saved. As a result, the settings of each tab are retained. When switching to a new tab, the settings of the previously selected tab are passed to the new tab.

You can switch from one tab to another at any time, as long as the debugger is not running in the current tab. Typically, you would enter different XPath or XQuery expressions in different tabs, evaluating and/or debugging each expression separately and switching between tabs to compare results.

**Evaluate Mode**

The XPath/XQuery Window enables you to build and evaluate up to nine XPath or XQuery expressions during one session, each within its own evaluation scope (which is specified in the *Where* option; *see screenshot below*). The expression can be evaluated on the current file, as well as on the following sets of multiple files: (i) all currently open documents; (ii) files of the currently active XMLSpy project; or (iii) files of a selected folder.
**Buttons in this section**

- **Debug Mode**
  - Switches between Debug and Evaluate Modes

- **Horizontal/Vertical Layout**
  - Switches between horizontal and vertical layouts

- **XML/JSON Evaluation Mode**
  - The highlighted part indicates which evaluation mode is active. When evaluation scope is multiple files, they are both enabled and one can be selected; otherwise, they are disabled. When disabled, evaluation mode is auto-detected according to file type

### About Evaluate Mode

- To switch to Evaluate Mode, deselect the **Debug Mode** button, which is located at the far left of the window's toolbar (see screenshot above).
- You can switch between a horizontal and vertical layout by clicking the **Horizontal/Vertical Layout** button, which is located at the far right of the window's toolbar (see screenshot above).
- Select the evaluation scope: either the current file, or a set of multiple files (open documents, files of a project, files of a folder). If the evaluation scope is the current file, evaluation mode (XML or JSON) is automatically determined by the **conformance type of the document's file type** (JSON mode for JSON-conformant files, XML mode for non-JSON files); the auto-detected mode cannot be changed and the buttons are disabled. If the evaluation scope is a multiple-file option, then the default evaluation mode is XML, but both buttons are enabled and you can select the evaluation mode you want; see **Evaluating expressions** below.
- Evaluate Mode has: (i) a **Builder tab**, which has entry helpers to aid in building the expression, and (ii) an **Evaluator tab**, which shows the results of evaluating the expression (see screenshot above).
- If the bottom pane is too short to display the names of the two tabs, the names are hidden and only the tabs' icons are displayed. Hovering over an icon displays the name of the tab.
- The **XPath/XQuery** Window in Evaluate Mode has nine tabs. After an expression is
entered in one of these tabs, the tab's settings are saved. So, if you return to a tab after switching to other tabs, the settings for that tab will be unchanged. If you switch to a new tab, the settings of the previous tab are passed to the new tab.

- If the expression is evaluated in the context of multiple XML documents (see the description of the Where option below), clicking on the filename in the results list opens the file in XMLSpy and makes it the active file.

XPath evaluations are described below. For a description of XQuery evaluations, see the section, XQuery Evaluation. (The XQ icons are for XQuery evaluation; the XQU icons are for XQuery Update executions.) For information about writing XPath/XQuery expressions in the XPath/XQuery Window, see XPath and XQuery Specification Information.

**XPath/XQuery Expression Builder**

The Builder tab (screenshot below) has two entry-helper panes: (i) for operators and expressions; and (ii) for functions. Note that, if more than one signature exists for a single function name, each signature is listed as a separate function. (These variants are known as overloads of that function name.) The items in both panes can be shown either grouped hierarchically or as a flat list. Select the option you want in the dropdown list at the top right of each pane (circled in red in the screenshot below). In the screenshot, both panes show their items in hierarchical groups.

Features of the Builder:

- To view a text description of an item in either entry-helper pane, hover over the item.
- Each function is listed with its signature (that is, with its arguments, the datatypes of the arguments, and the datatype of the function's output).
- Arguments are listed by their names (if any) or by their datatypes. Select the option you want from the dropdown list in the title bar of the Functions pane (circled in green in the
Double-clicking an item in any of the panes (operator, expression, or function), inserts that item at the cursor location in the expression. Functions are inserted with their arguments indicated by placeholders (the # symbol).

If (i) text is selected in the expression's edit field, and (ii) an operator, expression or function that contains a placeholder is double-clicked to insert it, then the text that was selected is inserted instead of the placeholder. This is a quick way to insert long text (such as a path expression) into an operator, expression, or function.

After you have entered a function in the expression, hovering over the function name displays the function's signature and a text description of the function. If more than one signature exists for a function, these are indicated with an overload factor at the bottom of the display. If you place the cursor within the parentheses of the function and press Ctrl+Shift+Spacebar, you can view the signatures of the various overloads of that function name.

### Evaluating expressions

Given below are the main steps for evaluating an XPath expression. The steps for XQuery expression evaluation are the same. For a dedicated description of XQuery evaluations, see the section, XQuery Evaluation. (The XQ icons are for XQuery evaluation; the XQU icons are for XQuery Update executions.) For a description of querying JSON documents, see JSON Transformations with XSLT/XQuery.

1. Depending on where the XPath expression is to be evaluated, select from one of the options in the Where options list (screenshot below): Current file; Open files; (XMLSpy) Project; or Folder.

   ![Screenshot](altova.png)

   If Current file is selected, the file that is currently active is used. Selecting Open files causes the XPath expression to be evaluated against all the files currently open in XMLSpy. Project refers to the currently active XMLSpy project. The external folders in an XMLSpy project can be excluded by checking the Skip external folders check box. The Folder option enables you to browse for the required folder; the XPath expression will be evaluated against XML files in this folder.

2. Select the XPath version you wish to use (1.0, 2.0, or 3.1) by clicking the appropriate icon in the toolbar of the output window (see screenshot below).

3. Select the type of XPath expression from the dropdown list. Allow Complete XPath is the usually required option. The other options of this combo box are XML Schema Selector and XML Schema Field, which can be used for a narrow subset of specific XPath 1.0 cases and are useful when unique identity constraints have been defined in the XML Schema. When either of these options is selected, only name tests (and the wildcard *) are allowed in the XPath expression, and predicates and XPath functions may not be used. Furthermore, for the XML Schema Selector option, only expressions on the child axis are allowed; for the XML Schema Field option, expressions on the child axis and attribute axis are allowed. For more information, see the W3C's XML Schema: Structures Recommendation. When the XPath version you select is not XPath 1.0, then the combo.
box is disabled: Allow Complete XPath is selected by default and cannot be changed.

4. Toggle the Evaluate XPath Expression On Typing icon on if you want the XPath expression to be evaluated while you are typing it in. If this icon is toggled off, the expression will be evaluated only when you click the Evaluate XPath Expression icon.

5. Toggle the Show Header In Output icon on if, in the output, you wish to show the XPath expression and the location of the XML file (as in the screenshot below).

6. If the XPath expression returns nodes—such as elements or attributes—you can select whether the entire contents of the selected nodes should be shown. This is done by switching the Show Complete Result icon on.

7. To set an XPath expression relative to a selection in the XML document, toggle the Set Current Selection As Origin icon on.

8. Click the Validate XML files icon to validate the XML files being evaluated for the XPath/XQuery expression. By default, the option is selected. Errors are treated as warnings and evaluation continues.

9. Enter the XPath expression. If you wish to create the expression over multiple lines (for easier readability), use the Return key.

10. You can increase/decrease the size of text in the expression field. Do this by clicking in the expression field, then pressing Ctrl and turning the scroll wheel.

11. Instead of manually entering the locator path expression of a node, you can let XMLSpy enter it for you. Do this as follows: (i) Place the cursor at the point in the XPath expression where you want to enter the locator path; (ii) Place the cursor in the start tag of the node you want to target; (iii) In the XPath/XQuery Window, click the Copies the XPath of the Current Selection to the Edit Field icon to enter the locator path of the selected node into the expression. The locator path will be entered as an absolute path starting at the root node of the document.

12. If the Show XPath Auto-completion icon has been toggled on, a popup will show a list of XPath functions and axes, and document elements and attributes that can be validly inserted at this point, and from which you can choose.

13. To evaluate the expression (if Evaluate on Typing is toggled off or when a new XML document is made active), click the Evaluate XPath Expression icon.

Results pane of the Evaluator tab
The Evaluator tab has a Results Pane (see screenshot above). The pane has the following functionality:
• If the evaluation is carried out on multiple files (specified in the Where option), then the results of each file are listed separately under the path of that file (see screenshot above).
• If the evaluation mode is XML, then XML conformant files are evaluated, other types are skipped (see screenshot above). If the evaluation mode is JSON, then JSON conformant files are evaluated, other types are skipped. See JSON Transformations with XSLT/ XQuery for information about querying JSON documents.
• When the result contains a node (including a text node)—as opposed to expression-generated literals—clicking on that node in the Result Pane highlights the corresponding node in the XML document in the Main Window.
• The result list consists of two columns: (i) a node name or a datatype; (ii) the value of the node. You can copy both columns of a result sub-line, or only the value column. Right-click a sub-line and toggle on/off Copying Includes All Columns. (Alternatively you can toggle the command on/off via its icon in the toolbar.) Then right-click the sub-line you want to copy and select either Copy Subline or Copy All.

The toolbar of the Evaluator tab contains icons that provide navigation, search, and copy functionality. These icons, starting from the left, are described in the table below. The corresponding commands are also available in the context menu of result list items.

<table>
<thead>
<tr>
<th>Icon</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next, Previous</td>
<td>Selects, respectively, the next and previous item in the result list</td>
</tr>
<tr>
<td>Copy the selected text line to the clipboard</td>
<td>Copies the value column of the selected result item to the clipboard. To copy all columns, toggle on the Copying includes all columns command (see below)</td>
</tr>
<tr>
<td>Copy all messages to the clipboard</td>
<td>Copies the value column of all result items to the clipboard, including empty values. Each item is copied as a separate line</td>
</tr>
<tr>
<td>Copying includes all columns</td>
<td>Switches between copying (i) all columns, or (ii) only the value column. The column separator is a single space</td>
</tr>
<tr>
<td>Find</td>
<td>Opens a Find dialog to search for any string, including special characters, in the result list</td>
</tr>
<tr>
<td>Find previous</td>
<td>Finds the previous occurrence of the term that was last entered in the Find dialog</td>
</tr>
<tr>
<td>Find next</td>
<td>Finds the next occurrence of the term that was last entered in the Find dialog</td>
</tr>
<tr>
<td>Clear</td>
<td>Clears the result list</td>
</tr>
<tr>
<td>Collapse multi-line results to a single line</td>
<td>If the value column of a result item contains multi-line text (text that includes newline character/s), you can toggle between a multi-line and single-line display</td>
</tr>
</tbody>
</table>

**Debug Mode**

The Debug Mode of the XPath/XQuery Window enables you to debug an XPath/XQuery expression in the context of the active file.
Switch to Debug Mode by clicking the **Debug Mode** button. Note that, in **Evaluate Mode**, the **Debug Mode** button is enabled only when the **Where** option is set to **Current File**. This is because debugging is carried out only in the context of the active file.

Debug Mode has two panes: (i) the pane in which the expression is entered, and (ii) the results pane. These panes can be divided from each other horizontally or vertically. To switch between these layouts, use the **Horizontal/Vertical Layout** button.

The **Results** pane has three tabs: **Result**, **Variables**, and **Call Stack**. When the height of the Results pane is not sufficient to display the names of the three tabs, the names are hidden and only the tabs’ icons are displayed. Hovering over an icon displays the name of the tab.

### Buttons in this section

- **Debug Mode**: Switches between Debug and Evaluate Modes
- **Horizontal/Vertical Layout**: Switches between horizontal and vertical layouts
- **Result**: Switches to the **Result** tab of the **Results** pane
- **Variables**: Switches to the **Variables** tab of the **Results** pane
- **Call Stack**: Switches to the **Call Stack** tab of the **Results** pane

### Debugging steps

The broad steps for debugging an **XPath/XQuery** expression are as follows:

1. Make the XML file on which you wish to run the expression the active file.
2. Enter the **XPath/XQuery** expression in the expression pane.
3. Set any breakpoints or tracepoints you want. A breakpoint is a point at which the evaluation is paused. A tracepoint is a breakpoint at which the evaluation result of that node is recorded, thus providing a traceable path through the evaluation.
4. Start the Debugger, or use the Step Into/Out/Over functionality to go step-by-step through the evaluation.

These steps are described in more detail below.
Starting, stopping, and restarting the debugger

When the debugger is stopped, only the Start Debugger button is enabled. After the Debugger has been started and before it stops, the Stop Debugger and Restart Debugger buttons are enabled.

- **Buttons in this section**
  - **Start Debugger (Alt +F11)**: Starts the debugger. The debugger evaluates the expression through to the end. It pauses only at breakpoints.
  - **Stop Debugger**: Exits the evaluation, and stops the debugger.
  - **Restart Debugger**: When the evaluation is paused (for example, at a breakpoint), restarts the evaluation from the beginning.

Stepping in, out, and over evaluation steps

The Step functionality enables you to go step-by-step through the evaluation. Each click takes you through the corresponding step of the evaluation.

- **Buttons in this section**
  - **Step Into (F11)**: Proceeds through the evaluation, one step at a time. This is a detailed stepping through the evaluation.
  - **Step Out (Shift+F11)**: Steps out of the current evaluation step, and goes to the "parent" step.
  - **Step Over (Ctrl+F11)**: Steps over "descendant" steps.

As the evaluation progresses, the expression step that is currently being processed is highlighted in the expression and the corresponding result is displayed in the Result tab (see screenshot above). While Step Into provides the most detailed debugging by stopping at every step, Step Out and Step Over provide a quicker way to track the expression's evaluation.

- **XQuery expression for trying the Step Into, Step Out, and Step Over functionality**
  
  ```xquery
  declare function local:plus($a, $b) {
      $a + $b
  }
  ```
declare function local:loop() {
    for $i in 1 to 10
    return (local:plus($i, $i+1), local:plus($i+1, $i+2))
};

local:loop()

Breakpoints and tracepoints

Breakpoints are points where you want the Debugger to stop after it has been started with Start Debugger. They are useful if you wish to analyze a specific part of the expression. When the Debugger stops at he breakpoint, you can check the result and could then use the Step Into functionality to display the results of the next steps of the evaluation. To set a breakpoint, place the cursor in the expression at the point where you want the breakpoint, and click the Insert/Remove Breakpoint (F9) toolbar button. The breakpoint will be marked with a dashed red overline (see screenshot below). To remove a breakpoint, select it and click Insert/Remove Breakpoint (F9).

Tracepoints are breakpoints at which the results are recorded and displayed in the Traces tree of the Result tab (see screenshot above). This enables you to see all the evaluation results of particular parts of the expression. For example, in the screenshot above, tracepoints have been set on $a, $b, local:plus($i, $i+1), and local:plus($i+1, $i+2); tracepoints are indicated by blue overlines. When the Debugger is at the expression part that is highlighted in the screenshot and when the value of $i is 1, then, in the Result tab, the values of the expression nodes at the tracepoints $a, $b, local:plus($i, $i+1) are displayed for this value of $i. To set a tracepoint, place the cursor in the expression at the point where you want the tracepoint, and click the Insert/Remove Tracepoint (Shift+F9) toolbar button. The tracepoint will be marked with a dashed blue overline (see screenshot below). To remove a tracepoint, select it and click
Insert/Remove Tracepoint (F9).

If both a breakpoint and a tracepoint are set on a part of the expression, then the overline is composed of alternating red and blue dashes (see the `local:plus($i, $i+1)` and `local:plus($i+1, $i+2)` function-calls in the screenshot above).

Result tab of the Results pane

The Result tab (see screenshot above) contains icons that provide navigation, search, and copy functionality. These icons, starting from the left, are described in the table below. The corresponding commands are also available in the context menu of result list items.

<table>
<thead>
<tr>
<th>Icon</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next, Previous</td>
<td>Selects, respectively, the next and previous item in the result list</td>
</tr>
<tr>
<td>Copy the selected text line to the clipboard</td>
<td>Copies the value column of the selected result item to the clipboard. To copy all columns, toggle on the Copying includes all columns command (see below)</td>
</tr>
<tr>
<td>Copy all messages to the clipboard</td>
<td>Copies the value column of all result items to the clipboard, including empty values. Each item is copied as a separate line</td>
</tr>
<tr>
<td>Copying includes all columns</td>
<td>Switches between copying (i) all columns, or (ii) only the value column. The column separator is a single space</td>
</tr>
<tr>
<td>Find</td>
<td>Opens a Find dialog to search for any string, including special characters, in the result list</td>
</tr>
<tr>
<td>Find previous</td>
<td>Finds the previous occurrence of the term that was last entered in the Find dialog</td>
</tr>
<tr>
<td>Find next</td>
<td>Finds the next occurrence of the term that was last entered in the Find dialog</td>
</tr>
<tr>
<td>Clear</td>
<td>Clears the result list</td>
</tr>
<tr>
<td>Collapse multi-line results to a single line</td>
<td>If the value column of a result item contains multi-line text (text that includes newline character/s), you can toggle between a multi-line and single-line display</td>
</tr>
</tbody>
</table>

Variables and Call Stack

The Results pane contains, in addition to the main Result tab, a Variables tab and Call Stack tab (screenshots below).

During debugging, the Variables tab displays the variables of the current step and their values (see screenshot below).
The **Call Stack** tab displays the various processor calls up to that point in the debugging. The most recent calls are on top, and the current call is highlighted in both panes (see screenshot below).

**XPath and XQuery Specification Information**

**XPath 1.0 expressions**

- XPath 1.0 functions must be entered without any namespace prefix.
- The four node tests by type are supported: `node()`, `text()`, `comment()`, and `processing-instruction()`.

**XPath 2.0 and 3.1 expressions**

- String (e.g. 'Hello') and numeric literals (e.g. 256) are supported. To create other literals based on XML Schema types, you use a namespace-prefixed constructor (e.g. `xs:date('2004-09-02')`). The namespace prefix that you use for XML Schema types must be bound to the XML Schema namespace: `http://www.w3.org/2001/XMLSchema`, and this namespace must be declared in your XML file.
- XPath 2.0 and 3.1 functions used by the XPath Evaluator belong to the namespace `http://www.w3.org/2005/xpath-functions`. Conventionally, the prefix `fn:` is bound to this namespace. However, since this namespace is the default functions namespace used by the XPath Evaluator, you do not need to specify a prefix on functions. If you do use a prefix, make sure that the prefix is bound to the XPath Functions namespace, which you must declare in the XML document. Examples of function usage: `current-date()` (with Functions namespace not declared in XML document); `fn:current-date()` (with Functions namespace not declared in XML document, or declared in XML document...
and bound to prefix \textit{fn:}). You can omit the namespace prefix even if the Functions namespace has been declared in the XML document with or without a prefix; this is because a function so used in an XPath expression is in the default namespace—which is the default namespace for functions.

- Altova’s XPath extensions are in the namespace \texttt{http://www.altova.com/xslt-extensions}.

\textbf{Note:} To summarize the namespace issue: If you use constructors or types from the XML Schema namespace, you must declare the XML Schema namespace in the XML document and use the correct namespace prefixes in the XPath expression. You do not need to use a prefix for XPath functions.

\section*{Datatypes in XPath 2.0 and 3.1}

If you are evaluating an XPath 2.0 or 3.1 expression for an XML document that references an XML Schema and is valid according to this schema, you must explicitly construct or cast datatypes that are not implicitly converted to the required datatype by an operation. In the XPath 2.0 and 3.1 Data Models used by the built-in XPath engine, all \textit{atomized} node values from the XML document are assigned the \texttt{xs:untypedAtomic} datatype. The \texttt{xs:untypedAtomic} type works well with implicit type conversions. For example, the expression \texttt{xs:untypedAtomic("1") + 1} results in a value of \texttt{2} because the \texttt{xs:untypedAtomic} value is implicitly promoted to \texttt{xs:double} by the addition operator. Arithmetic operators implicitly promote operands to \texttt{xs:double}. Comparison operators promote operands to \texttt{xs:string} before comparing.

In some cases, however, it is necessary to explicitly convert to the required datatype. For example, if you have two elements, \texttt{startDate} and \texttt{endDate}, that are defined as being of type \texttt{xs:date} in the XML Schema, then, for example, using the XPath 2.0 expression \texttt{endDate - startDate} will show an error. On the other hand, if you use \texttt{xs:date(endDate) - xs:date(startDate)} or \texttt{(endDate cast as xs:date) - (startDate cast as xs:date)}, the expression will correctly evaluate to a singleton sequence of type \texttt{xs:dayTimeDuration}.

\textbf{Note:} The XPath Engines used by the XPath Evaluator are also used by the Altova XSLT Engine, so XPath 2.0 or 3.1 expressions in XSLT stylesheets that are not implicitly converted to the required datatype must be explicitly constructed as or cast to the required datatype.

\section*{String length of character and entity references}

When character and entity references are used as the input string for the \texttt{string-length()} function, the references cannot be resolved, and the length of the unresolved text string is returned. Within an XSLT environment, however, these references would have meaning, and the length of the resolved string is returned.

\section*{XPath 2.0 and 3.1 Functions Support}

See the \texttt{appendices}.

\subsection*{3.1.7 Output Window: XSL Outline}

The XSL Outline Window (\textit{screenshot below}) lists all the templates and functions in an XSLT stylesheet, and, optionally, in all included and imported XSLT stylesheets as well. The XSL Outline Window is located by default docked with the Output Windows at the bottom of the
The XSL Outline Window provides information about templates and functions in the stylesheet. This information can be sorted and searched, and the window's toolbar contains commands that enable you to easily insert calls to named templates and to set named templates as the starting point of transformations. How to work with the XSL Outline Window is described in the section \textit{XSLT and XQuery | XSLT | XSL Outline | XSL Outline Window}.

\textbf{Note:} File-related information about the stylesheet and file-related commands are available in the XSLT tab of the Info Window. How to use these commands is described in the section \textit{XSLT and XQuery | XSLT | XSL Outline | Info Window}.

\subsection*{3.1.8 Output Window: HTTP}

In the HTTP output window (\textit{screenshot below}), you can test HTTP commands: you can create and send an HTTP request to a web server, and receive and check the response.

The HTTP output window has nine tabs (\textit{see screenshot below}). You can store a separate request in each tab, and switch between tabs. After creating a request in the window, you can send the request by clicking the \textbf{Send} button. The response is displayed directly in the window.
The window consists of the following parts:

- At the top: (i) a combo box in which to select the HTTP method you want to use; (ii) an entry field for the URL of the web server; (iii) buttons related to the execution of HTTP requests (Send, Import, and Reset).
- A left-hand pane for creating the request.
- A right-hand pane for displaying the response.

For details about how to use the HTTP output window, see the section HTTP.

3.1.9 Output Window: Find in Files

The Find in Files Window (screenshot below) enables you to carry out find-and-replace operations quickly within several documents at a time, and provides mechanisms that help you to quickly navigate among the found instances. The results of each find-and-replace action are presented in one of the tabs numbered 1 to 9. Clicking on a found item in the results takes you to that item in the Text View of that document.
Find criteria
There are two broad find criteria: (i) what to find, and (ii) where to look?

**What to find:** The string to find is entered in the Find What text box. If that string must match a whole word, then the Match Whole Word check box must be clicked. For example, for the find string *fit*, with Match Whole Word checked, only the word *fit* will match the find string; the *fit* in *fitness*, for example, would not. You can specify whether casing is significant using the Match Case check box. If the text entered in the Find What text box is a regular expression, then the Regular Expression check box must be checked. An entry helper for regular expression characters can be accessed by clicking the button. The use of regular expressions for searching is explained in the section, **Find**. The More button opens the Find in Files dialog, where you can set advanced search conditions and actions. For more information, see **Edit | Find in Files**.

**Where to look:** The search can be conducted in: (i) all the files that are open in the GUI; (ii) the files of the current project; and (iii) the files of a selected folder. You can set additional conditions in the Find in Files dialog (accessed by clicking More).

Replace with
The string with which the found string is to be replaced is entered in the Replace With text box. Note that if the Replace With text box is empty and you click the Replace button, the found text will be replaced by an empty string.

The results
After you click the Find or Replace buttons, the results of the find or replace are displayed in the Find in Files output window. The results are divided into four parts:

- A summary of the search parameters, which lists the search string and what files were searched.
- A listing of the found or replaced strings (according to whether the Find or Replace button was pressed). The items in this listing are links to the found/replaced text in the Text View of the document. If the document is not open, it will be opened in Text View and the found/replaced text will be highlighted.
- A list of the files which were searched but in which no matches were found.
- A summary of statics for the search action, including the number of matches and number of files checked.

**Note:** Note that the Find in Files feature executes the Find and the Replace commands on multiple files at once and displays the results in the Find in Files output window. To do a find so that you go from one found item to the next, use the Find command.

### 3.1.10 Output Window: Find in Schemas
When an XML Schema is active in Schema View, it can be searched intelligently using XMLSpy's Find and Replace in Schema View feature. The Find and Replace in Schema View feature is accessed via: (i) the Find and Replace commands in the Edit menu; and (ii) the Find and Replace buttons in the Find in Schemas Window (screenshot below).
The results of the Find and Replace in Schema View feature (i.e. each time a Find or Replace command is executed) are displayed in the Find in Schemas window. The term that was searched for is displayed in green; (in the screenshot above, it can be seen that email was the search term, with no case restriction specified). Notice that the location of the schema file is also given.

Results are displayed in nine separate tabs (numbered 1 to 9). So you can keep the results of one search in one tab, do a new search in a new tab, and compare results. To show the results of a new search in a new tab, select the new tab before starting the search. Clicking on a result in the Find In Schemas window pops up and highlights the relevant component in the Main Window of Schema View. In this way you can search and navigate quickly to the desired component, as well as copy messages to the clipboard. For more details, see the Results and Information section in the description of the Find in Schemas feature.

3.1.11 Menu Bar, Toolbars, Status Bar

Menu Bar

The menu bar (see illustration) contains the various application menus. The following conventions apply:

- If commands in a menu are not applicable in a view or at a particular location in the document, they are unavailable.
- Some menu commands pop up a submenu with a list of additional options. Menu commands with submenus are indicated with a right-pointing arrowhead to the right of the command name.
- Some menu commands pop up a dialog that prompts you for further information required to carry out the selected command. Such commands are indicated with an ellipsis (...) after the name of the command.
- To access a menu command, click the menu name and then the command. If a submenu is indicated for a menu item, the submenu opens when you mouseover the menu item. Click the required sub-menu item.
- A menu can be opened from the keyboard by pressing the appropriate key combination. The key combination for each menu is Alt+KEY, where KEY is the underlined letter in the menu name. For example, the key combination for the File menu is Alt+F.
- A menu command (that is, a command in a menu) can be selected by sequentially selecting (i) the menu with its key combination (see previous point), and then (ii) the key combination for the specific command (Alt+KEY, where KEY is the underlined letter in
the command name). For example, to create a new file (**File | New**), press **Alt+F** and then **Alt+N**.

- Some menu commands can be selected **directly** by pressing a special **shortcut** key or key combination (**Ctrl+KEY**). Commands which have shortcuts associated with them are indicated with the shortcut key or key combination listed to the right of the command. For example, you can use the shortcut key combination **Ctrl+N** to create a new file; the shortcut key **F8** to validate an XML file. You can **create your own shortcuts** in the Keyboard tab of the Customize dialog (**Tools | Customize**).

**Toolbars**

The toolbars (**see illustration**) contain icons that are shortcuts for selecting menu commands. The name of the command appears when you place your mouse pointer over the icon. To execute the command, click the icon.

Toolbar buttons are arranged in groups. In the **Tools | Customize | Toolbars** dialog, you can specify which toolbar groups are to be displayed. These settings apply to the current view. To make a setting for another view, change to that view and then make the setting in the **Tools | Customize | Toolbars**. In the GUI, you can also drag toolbar groups by their handles (or title bars) to alternative locations on the screen. Double-clicking the handle causes the toolbar to undock and to float; double-clicking its title bar causes the toolbar to dock at its previous location.

**Status Bar**

The Status Bar is located at the bottom of the application window (**see illustration**) and displays (i) status information about the loading of files, and (ii) information about menu commands and command shortcuts in the toolbars when the mouse cursor is placed over these. If you are using the 64-bit version of XMLSpy, this is indicated in the status bar with the suffix (x64) after the application name. There is no suffix for the 32-bit version.
3.2 The Application Environment

In this section we describe various aspects of the application that are important for getting started. Reading through this section will help you familiarize yourself with XMLSpy and get you off to a confident start. It contains important information about settings and customization, which you should read for a general idea of the range of settings and customization options available to you and how these can be changed.

This section is organized as follows:

- **Settings and Customization**: Describes how and where important settings and customization options can be defined.
- **Tutorials, Projects, Examples**: Notes the location of the various non-program files included in the application package.
- **Product features and documentation, and Altova products**: Provides links to the Altova website, where you can find information about product features, additional Help formats, and other Altova products.

3.2.1 Settings and Customization

In XMLSpy, there are several settings and customization options that you can select. In this section, we point you to these options and also briefly discuss some aspects of XMLSpy menus. This section is organized into the following parts.

- **Settings**
- **Customization**
- **Menus**

**Settings**

Several important XMLSpy settings are defined in different tabs in the Options dialog (screenshot below, accessed via the menu command [Tools | Options](#)). You should look through the various options to familiarize yourself with what's available.
Given below is a summary of the most important settings. For details, see the description of the Options dialog in the User Reference section.

- **File types and default views**: In the File Types tab, you can add file types that XMLSpy will recognize. A file type is specified by a file extension. For each file type, you can then specify conformance to a particular standard (for example, conformance to the DTD, XQuery, or JSON standard). This setting will switch on editing aids relevant to the standard selected for a particular file type. You can also specify in what XMLSpy view files of each file type should open (the default view for this file type).

- **File validation**: In the File tab (screenshot above), you can specify whether files should be validated automatically on opening and/or saving. In the File Types tab (see previous bullet point), file validation can then be disabled for specific file types.

- **Editing features**: In the Editing tab, you can specify how entry helpers should be organized, how new elements are generated, and whether auto-completion is enabled. Additional options are available for individual views in the View tab. In the Fonts tabs for various views, you can specify the font characteristics of individual node types in each of these views.

- **XSLT and FO Engines**: In the XSL tab, you can specify that an external XSL engine be used for transformations made from within the GUI. You must also specify the location of the FO processor executable to be used for FO processing within XMLSpy. For more information, see the XSLT Processing section.

- **Encoding**: Default encodings for XML and non-XML files are specified in the Encoding tab.

**Customization**

You can also customize various aspects of XMLSpy, including the appearance of the GUI. These customization options are available in the Customize dialog (screenshot below, accessed via the menu command **Tools | Customize**).
The various customization options are described in the User Reference section.

**Menus**

Menu commands are enabled/disabled depending upon three factors: (i) file type, (ii) active view, and (iii) current cursor location or current document status. For example:

- **File type**: The command DTD/Schema | Include Another DTD is enabled only when the active file is a DTD. Similarly, commands in the WSDL menu will be enabled only when a WSDL file is active.
- **Active view**: Most commands in the Schema Design menu will be active only when the active view is Schema View.
- **Current cursor location, document status**: In Grid View, whether the command to add an attribute as a child node (XML | Add Child | Attribute) is enabled will depend on whether the selected item in Grid View is an element or not (current cursor location). When an XSLT document is active the Stop Debugger command will not be active till after a debugger session has been started (current document status).

Note also that you can customize menus (Tools | Options) as well as drag and reorganize them within the GUI (see Menu Bar, Toolbars, Status Bar).

### 3.2.2 Tutorials, Projects, Examples

The XMLSpy installation package contains tutorials, projects, and example files.

**Location of tutorials, projects, and example files**

The XMLSpy tutorials, projects, and example files are installed in the folder.
The My Documents\Altova\XMLSpy2019\Examples\ folder will be installed for each user registered on a PC within that user's <username> folder. Under this installation system, therefore, each user will have his or her own Examples folder in a separate working area.

Note about the master XMLSpy folder

When XMLSpy is installed on a machine, a master Altova\XMLSpy2019 folder is created at the following folder location:

C:\Documents and Settings\All Users\Application Data\%

When a user on that machine starts XMLSpy for the first time, XMLSpy creates a copy of this master folder in the user's <username>\My Documents\ folder. It is therefore important not to use the master folder when working with tutorial or example files, otherwise these edited files will be copied to the user folder of a user who subsequently uses XMLSpy for the first time.

Location of tutorial, project, and examples files

All tutorial, project, and example files are located in the Examples folder. Specific locations are as follows:

- XMLSpy tutorial: Tutorial folder.
- Authentic View tutorial: Examples folder.
- WSDL tutorial: Examples folder.
- Project file: The Examples project with which XMLSpy opens is defined in the file Examples.spp, which is located in the Examples folder.
- Example files: are in the Examples folder and in sub-folders of the Examples folder.

3.2.3 XMLSpy Features and Help, and Altova Products

The Altova website, www.altova.com, has a wealth of XMLSpy-related information and resources. Among these are the following.

XMLSpy feature listing

The Altova website carries an up-to-date list of XMLSpy features, which also compares the support of various features across XMLSpy editions (Enterprise and Professional). On the website, you can also obtain a listing of features that are new since any previous release.

XMLSpy Help

This documentation is the Altova-supplied Help for XMLSpy. It is available as the built-in Help system of XMLSpy, which is accessible via the Help menu or by pressing F1. Additionally, the user manuals for all Altova products are available in the following formats:

- Online HTML manuals, accessed via the Support page at the Altova website
- Printable PDFs, which you can download from the Altova website and print locally
- Printed books that you can buy via a link at the Altova website
Support options
If you require additional information to what is available in the user manual (this documentation) or have a query about Altova products, visit our Support Center at the Altova website. Here you will find:

- Links to our FAQ pages
- Discussion forums on Altova products and general XML subjects
- Online Support Forms that enable you to make support requests, should you have a support package. Your support request will be processed by our support team.

Altova products
For a list of all Altova products, see the Altova website.
4 Editing Views

XMLSpy contains powerful editing views. In addition to a Text View with intelligent editing features, there are graphical views that greatly simplify the editing of documents. Depending on what type of document is currently active in XMLSpy, the Main Window will have one or more of XMLSpy's Editing Views. For example, when an HTML document is active, the Main Window will contain two editing views: Text View and Browser View. When an XML document is active, there will be five editing views: Text View, Grid View, Schema View, Authentic View, and Browser View. Of these views, Schema View will be enabled only for XML Schema documents.

In this section, we describe the various editing views available in XMLSpy:

- Text View
- Grid View
- Schema View
- Authentic View
- Browser View
4.1 Text View

In Text View (screenshot below), you can type in the text of your document—both, markup and content—directly. Any text file, including non-XML documents (such as XQuery and HTML documents) can be edited in Text View. A number of features help you to quickly and accurately type in your document.

```
<Office>
  <Name>NaNonull, Inc.</Name>
  <Desc/>
  <Location>US</Location>
  <Address>
    <postal:street>119 Oakstreet, Suite 4076</postal:street>
    <postal:city>Vernon</postal:city>
    <postal:state>DC</postal:state>
    <postal:zip>22211</postal:zip>
  </Address>
  <Phone>+1 (321) 555 5155 0</Phone>
  <Fax>+1 (321) 555 5155 4</Fax>
  <Email>office@nanonull.com</Email>
  <Department>
    <Name>Administration</Name>
    <Person>
      <First>Vernon</First>
      <Last>Callaby</Last>
      <Title>Office Manager</Title>
      <PhoneExt>582</PhoneExt>
      <Email>V.Callaby@nanonull.com</Email>
      <Shares>1500</Shares>
      <LeaveTotal>25</LeaveTotal>
      <LeaveUsed>4</LeaveUsed>
      <LeaveLeft>21</LeaveLeft>
    </Person>
  </Department>
</Office>
```

In this section, we describe general Text View features that are available for all kinds of documents. Specific document types, such as XML, XQuery, and CSS have certain type-specific features, which are described in the respective sections for those document types. For example, additional XML-specific features of Text View are described in the section XML | Editing XML in Text View.

The general Text View features have been organized as follows:

- **Formatting in Text View** describes how the font properties, indentation, and word-wrapping of the document can be specified.
- **Displaying the Document** contains information about the line-numbering, bookmarking, expanding/collapsing of nodes, and other display-related features.
- **Editing in Text View** describes the features that are available while you edit, particularly the intelligent editing features.
- **Navigating the Document** explains the various ways in which you can navigate a document in Text View.
- **Find and Replace** describes the Find and Replace features of Text View.
- **Entry helpers** are the windows that provide context-sensitive data-entry options. For example, the elements or attributes that can be validly added at a given document location are displayed in an entry helper and any one of these options can be inserted by double-clicking it.
- **Split View** divides the main window of Text View in two and displays the active document in both views. This enables you to see different parts of a long document side-by-side.
- **Text View Shortcuts** lists the default shortcuts of commonly used Text View commands.
Switching to Text View

To open the Text View of a document, click the Text button at the bottom of the Document Window or select View | Text View.

4.1.1 Formatting in Text View

Text View offers a number of text formatting options. These are listed below.

Fonts

The font-family, font-size, font-style, and text background-color can be customized separately for the following groups of documents: (i) generic XML documents (including HTML); (ii) XQuery documents; and (iii) CSS documents.

Text items in a document that have different semantics, can be colored differently. For example, you can color element names, attribute names, and element content differently. When you set different colors for different text items, the syntax-coloring feature is enabled. Text fonts are customized in the Fonts and Colors section of the Options dialog.

Indentation

To display the document in this way, you need to do the following:

1. In the View section of the Options dialog, check the Use Indentation option for pretty printing. This will cause the document to be pretty-printed with indents to indicate the hierarchical structure. Each deeper level will be displayed with a deeper indent than its parent element. If the Use Indentation option is not checked, every line in the document will start with a zero indent.
2. In the Text View Settings dialog (View | Text View Settings; screenshot below), select either Insert Tabs or Insert Spaces. This determines whether tabs or spaces will be used for indentation when the document is pretty-printed. If spaces are specified, each deeper level of the hierarchy is indented with an additional number of spaces as specified in the Tab Size setting of the Text View Settings dialog.

3. Click the Edit | Pretty-Print XML Text command or the Pretty Print icon in the Text toolbar. This will cause the document text to be displayed (i) with or without indentation as specified in the View section of the Options dialog; and (ii) if indentation is specified in the View section of the Options dialog, then the the indentation is determined by the settings in the Tabs pane of the Text View Settings dialog. Clicking the Pretty Print command removes unnecessary leading or trailing whitespace.

Note: Pretty-printing is also used in the background when you save the document or switch views. If the document is not well-formed, you will get an error message to that effect. Correct the error and then pretty-print. The extent of indentation of a line is indicated by indentation guides, which are vertical dotted lines (see screenshot at the start of this section) that are toggled on and off with the Indentation Guides check box in the Visual Aid pane of the Text View Settings dialog (see screenshot above).

Using tabs and spaces for formatting
You can use tabs and spaces for formatting text, especially for non-XML documents, where the pretty-printing option is not available. When you press Return or Shift+Return, the cursor will jump to a position on the next line that corresponds to the starting position of the previous line.

Word-wrapping
Lines of text that are longer than the breadth of the Main Window can be made to wrap by toggling the View | Word Wrap command on; the corresponding icon is in the Text toolbar.

4.1.2 Displaying the Document
Text View has visual features to make the display and editing of large sections of text easier. Some very useful features are: (i) Line Numbers, (ii) Bookmarks, (iii) Source Folding (expanding and collapsing the display of nodes), (iv) Indentation Guides, and (v) End-of-Line and Whitespace Markers. These commands are available in the Text View Settings dialog (first screenshot below) and the Text toolbar (second screenshot below).
The **Text View Settings dialog** is accessed via the **View | Text View Settings** command, the **Text View Settings** button in the Text toolbar, or the Text View context menu. Settings in the Text View Settings dialog apply to the entire application—not only to the active document.

Other useful features are the **Zooming** and **navigation and search** features.

### Line numbers

Line numbers are displayed in the line numbers margin (*screenshot below*), which can be toggled on and off in the Text View Settings dialog (see screenshot above). When a section of text is collapsed, the line numbers of the collapsed text are also hidden. A related command is the **Go-to-Line/Character** command.

### Bookmarks

Lines in the document can be separately bookmarked for quick reference and access. If the bookmarks margin is toggled on, bookmarks are displayed in the bookmarks margin; otherwise, bookmarked lines are highlighted in cyan.

The bookmarks margin can be toggled on or off in the Text View Settings dialog (*screenshot above*).

You can edit and navigate bookmarks using commands in the **Edit** menu and Text toolbar. Bookmarks can be inserted with the **Edit | Insert/Remove Bookmark** command, enabling you to mark a line in the document for reference. A bookmark can be removed by selecting the bookmarked line and then selecting the **Edit | Insert/Remove Bookmark** command. To navigate through the bookmarks in a document, use the **Edit | Next Bookmark** and **Edit | Previous Bookmark** commands. These bookmark commands are also available as icons in the Text toolbar (*screenshot above*).
Source folding

Source folding refers to the ability to expand and collapse nodes in XML, XQuery, JSON, and CSS documents. Nodes that can be expanded/collapsed are indicated in the source folding margin by a +/- sign (see screenshot below). The margin can be toggled on and off in the Text View Settings dialog (see screenshot above). In the screenshot below, notice that three nodes have been collapsed: the `shipTo` element and two `item` elements. When a node is collapsed, this is visually indicated by an ellipsis (marked in green in the screenshot below). If the mouse cursor is placed over an ellipse, the content of the collapsed node is displayed in a popup (marked in blue in the screenshot below). If the content is too large for a popup, this is indicated by an ellipsis at the bottom of the popup.

The **Toggle All Folds** icon in the Text toolbar toggles all nodes to their expanded forms or collapses all nodes to the top-level document element.

The following options are available when clicking on the node's +/- icon:

<table>
<thead>
<tr>
<th>Action</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Click [-]</strong></td>
<td>Collapses the node.</td>
</tr>
<tr>
<td><strong>Click [+]</strong></td>
<td>Expands the node so that descendant nodes are shown expanded or collapsed according to how they were before the node was collapsed.</td>
</tr>
<tr>
<td><strong>Shift+Click [-]</strong></td>
<td>Collapses all descendant nodes, but leaves the node that was clicked in its current state.</td>
</tr>
<tr>
<td><strong>Ctrl+Click [+]</strong></td>
<td>Expand the clicked node as well as all its descendant nodes.</td>
</tr>
</tbody>
</table>

**Indentation guides**

Indentation guides are vertical dotted lines that indicate the extent of a line's indentation (see
End-of-line (EOL) markers and whitespace markers can be toggled on in the Text View Settings dialog. The screenshot below shows these markers in the document display; each dot represents a whitespace.

```
5     .<CompanyLogo href="nanonull.gif"/>
6     .<Name>Organization.Chart</Name>
7     .<Office>
8     .<Name>Nanonull, Inc.</Name>
9     .<Desc>
```

Zooming in and out
You can zoom in and out of Text View by turning the scroll-wheel of the mouse while keeping the Ctrl key pressed. This enables you to magnify and reduce the size of text in Text View. If you wish to increase the size of fonts, do this in the Options dialog.

4.1.3 Editing in Text View
The following text editing features are available in Text View generally for all document types. These features are in addition to common features of editing applications, such as Cut, Copy, Paste, Delete, and Select All (which are available as commands in the Edit menu).

- Syntax coloring
- Start-tag and end-tag matching
- Intelligent editing
- Auto-completion
- Moving siblings relative to each other
- Selecting an entire element and going to parent
- Drag-and-drop and context menus
- Unlimited undo
- Spelling check

For some document types (such as XML and XQuery) additional specialized features are available, and these are described, respectively, in the sections that deal with those document types.

**Note:** For large files, Auto-completion and entry helpers can be disabled, thus enabling faster loading and editing. The threshold file size is specified by the user. For more details, see the reference section Options | Editing.

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. A number of document type are distinguished, such as: (i) generic XML (which includes HTML); (ii) XQuery; (iii) CSS; and (iv) JSON. The text properties (including color) of each group can be set in the Text Fonts section of the Options dialog (Tools | Options).
Intelligent Editing

If you are working with an XML document based on a schema, XMLSpy provides you with various intelligent editing capabilities in Text View. These allow you to quickly insert the correct element, attribute, or attribute value according to the content model defined for the element you are currently editing. For example, when you start typing the start tag of an element, the names of all the elements allowed by the schema at this point are shown in a pop-up (screenshot below). Selecting the required element name and pressing **Enter** inserts that element name in the start tag. Also, after the start tag is created, the end tag is automatically added (see Auto-completion below).

Popup windows also appear in the following cases:

- When the cursor is inside the start tag of an element that has an attribute defined for it and the space bar is pressed. The popup will contain all available attributes.
- When the cursor is within the double-quotes delimiting an attribute value that has enumerated values. The popup will contain the enumerated values.
- When you type `<<!--` (which signifies the start of a closing tag), the name of the element to be closed appears in the popup.
- When you wish to write an empty element as a single tag or convert an empty element of two tags to an empty element of one tag, type in the closing slash after the element name: `<element/>. An empty element with a single tag is created; if a close tag exists, it is removed: `<element/>.`

Auto-completion

Editing in Text View can easily result in XML and other marked-up documents (such as HTML) that are not well-formed. For example, closing tags may be missing, mis-spelled, or structurally mismatched. XMLSpy automatically completes the start and end tags of elements, as well as inserts all required attributes as soon as you finish entering the element name on your keyboard. The cursor is also automatically positioned between the start and end tags of the element, so that you can immediately continue to add child elements or contents: `<img src="" alt=""/>`</img>

XMLSpy makes use of the XML rules for well-formedness and validity to support auto-completion. The information about the structure of the document is obtained from the schema on which the document is based. (In the case of well-used schemas, such as HTML and XSLT, the schema information is built into XMLSpy.) Auto-completion uses not only information about the structure of the document, but also the values stored in the document. For example, enumerations and
schema annotations in an XML Schema are actively used by the Auto-Completion feature. If, in the schema, values are enumerated for a particular node, then those enumerations will be displayed as auto-completion options when that node comes to be edited. Similarly, if, for a node, annotations exist in the schema, then these annotations are displayed when the node name is being typed in the document (screenshot below). (First (given) name of person is the schema annotation of the First element.)

Auto-completion can be switched on and off in the Editing section of the Options dialog (Tools | Options | Editing).

Start-tag and end-tag matching
When you place the cursor inside a start or end tag of a markup element, pressing Ctrl+E moves the selection to the other member of the pair. Pressing Ctrl+E repeatedly enables you to switch repeatedly between the start and end tags. This feature is an excellent aid for quickly locating the start and end tags of an XML element. Additionally, the names of elements are highlighted in two different colors according to whether the names in the start and end tags match or not. This serves as a visual editing aid. The highlight colors can be set in the Options dialog. When you edit the name of an element in a start tag, then the end tag will be automatically edited as well.

Moving sibling elements relative to each other
When the cursor is within an element, pressing Alt+ArrowUp or Alt+ArrowDown moves the selected element up or down relative to its siblings.

Selecting an entire element and going to parent
When the cursor is within an element, pressing Ctrl+Shift+E selects the entire element. If you click Ctrl+Alt+E, the start tag of the parent element is highlighted. Both these shortcuts help you to quickly locate your current cursor position relative to the document structure.

Drag-and-Drop and Context Menus
You can also use drag-and-drop to move a text block to a new location, as well as right-click to directly access frequently used editing commands (such as Cut, Copy, Paste, Delete, Send by Mail, and Go to line/char) in a context menu.

Unlimited Undo
XMLSpy offers unlimited levels of Undo and Redo for all editing operations.

Spelling check
In Text View, documents can be spellchecked with any of the built-in language dictionaries. A
user dictionary can also be created and edited to allow words not contained in the language dictionary. For details, see the descriptions of the Spelling and Spelling Options commands.

4.1.4 Navigating the Document

You can use the following features to navigate a document in Text View:

- **Text highlighting** enables you to find all matches of a text string or word that you select. Each match is indicated in the scroll bar, so you can navigate easily through all the matches.
- **Document overview in the scroll bar** shows you the relative location of the cursor and text selection within the document.
- **Go to line/character** takes you straightaway to the line and character you specify.

Text highlighting

If text highlighting is enabled in the Text View Settings dialog (View | Text View Settings), then all matches in the document of a text selection that the user makes are highlighted. The selection will be highlighted in pale blue, and matches will be highlighted in pale orange (see screenshot below). The selection and its matches are indicated in the scroll bar by gray marker-squares. Note also that the current cursor position is given by the blue cursor-marker in the scroll bar.

To switch the text highlighting feature on, select Enable auto-highlighting in the Text View Settings dialog (View | Text View Settings, screenshot below). 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.
Note the following points:

- 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 `t`, then no matches will be shown; selecting `ty` will show all `ty` matches; selecting `typ` will show all `typ` 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.

Note: Element start-tag and end-tag matching is a separate feature that is not affected by the Enable auto-highlighting setting.

Document overview in the scroll bar

The scroll bar provides the following features:

- It relates the sizes of the following to each other (see screenshot below): (i) the entire document (scroll bar); (ii) the document segment that is currently in the window (thumb); (iii) the current text selection (blue bar), if any; and (iv) the current cursor location (cursor-marker).
- It enables you to navigate the document by either: (i) dragging the scroll bar's thumb up and down, or (ii) clicking the Page Up and Page Down arrows (circled in green in the screenshot below).
Note the following points:

- The length of the scroll bar corresponds to the length of the entire document.
- If only a part of the document fits in the window, then this windowed part corresponds to the scroll bar's thumb (see screenshot above). You can drag the thumb up and down to bring other parts of the document into the window. It is as if the thumb represents the window and you are moving the window up and down the document in order to view the document.
- The current text selection is indicated in the scroll bar by the blue bar. The size of the blue bar relative to the size of the scroll bar is proportional to the size of the text selection relative to the size of the entire document. If the text selection does not exceed one line, the blue bar will not be visible.
- The cursor position is indicated by a dark blue cursor-marker. The cursor-marker's relative position in the scroll bar corresponds to the cursor's relative position in the document.

Go to line/character

This command in the View menu and Text toolbar enables you to go to a specific line and character in the document text.

4.1.5 Find and Replace

In this section:

- Searching for text in a document
- Find options
- Replacing text
- Using regular expressions
- Regular expression examples
Searching for text in a document

Press Ctrl+F (or select the menu command Edit | Find) to display the Find dialog (screenshot below, top right). 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 (see screenshot).
- 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 (see screenshot below). 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 (Shift+F3) and Next (F3) buttons at bottom right.

![Find dialog screenshot]

- To switch between the Find and Replace modes, click the down-arrow button at top left.
- To close the dialog click the Close button at top right.

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 (aka Find 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
Options to determine Find criteria can be specified via buttons located below the search term field (see screenshot below). When an option is toggled on, its button color changes to blue (see the first (casing) option in the screenshot below).

You can select from the following options:

- **Match case**: Case-sensitive search when toggled on (Address is not the same as address).
- **Match whole word**: 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 be matched.
- **Regular expression**: If toggled on, the search term will be read as a regular expression. See Regular expressions below for a description of how regular expressions are used.
- **Filter results**: Select one or more document components where the search is to be carried out.
- **Find anchor**: 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**: 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 currently selection by toggling off the Find in Selection option.

Replacing text
To replace a text string, do the following:

1. Press Ctrl+H (or select the menu command Edit | Replace) to open the Replace dialog (screenshot below). (Alternatively, you can switch to Replace mode of the Find/Replace dialog by clicking the down-arrow button at the top left of the dialog.)

2. Enter the string to be replaced in the Find field, and enter the new string in the Replace field. The number of text matches to replace and the index of the currently selected match is displayed below the Replace field. Also, the locations of matches are indicated in the scroll bar by beige markers (see Searching for text in a document above for more information). For example, the screenshot above shows that there are five text matches for the string type, and that the third of these matches is currently selected.
3. The Replace Next and Replace All buttons are located to the right of the Replace field.
   If you click Replace Next, one of the following happens: (i) If the cursor is located
   adjacent to a match or inside a match, then the match is replaced; (ii) if the cursor is
   located outside a match, it jumps to the next match; click Replace Next to replace this
   match. Click Replace All to replace all matches.

Note the following points:

- To replace text within a selection—rather than the entire document—do the following: (i)
  Mark the selection; (ii) Toggle on the Find in Selection option to lock the selection; (iii)
  Enter the Find and Replace text strings; (iii) Click Replace Next or Replace All as
  required. To replace text 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 to lock the new selection.

- To undo a replace action, press Ctrl+Z or select Edit | Undo.

Using regular expressions

You can use regular expressions (regex) to find a text string. To do this, first, switch the Regular
expression option on (see Find options above). 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 (see screenshot below). 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. For a brief description of
metacharacters, see the section Regular expression metacharacters below.
Regular expression metacharacters
Given below is a list of regular expression metacharacters.

| .   | Matches any character. This is a placeholder for a single character. |
| (   | Marks the start of a tagged expression.                           |
| )   | Marks the end of a tagged expression.                            |
| \{ | 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 expressions can be tagged (and then back-referenced later, either in the Find or Replace field).

For example, \(\text{the}\) \(\backslash 1\) matches the string \text{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 a variable that can take integer values from 1 through 9. The expression refers to the first through ninth tagged region when replacing. For example, if the find string is \text{Fred([1-9])XXX} and the replace string is \text{Sam\{1\}YYY}, this means that in the find string there is one tagged expression that is (implicitly) indexed with the number 1; in the replace string, the tagged expression is referenced with \(1\). If the find-replace command is applied to \text{Fred2XXX}, it would generate \text{Sam2YYY}.

\(<\) Matches the start of a word.

\(>\) Matches the end of a word.

\(x\) Allows you to use a character \(x\), which would otherwise have a special meaning. For example, \(\[\) would be interpreted as \[ and not as the start of a character set.

[..] Indicates a set of characters. For example, \[abc\] means any of the characters \(a\), \(b\) or \(c\). You can also use ranges: for example \[a-z\] for any lower case character.

[^..] The complement of the characters in the set. For example, \[^A-Za-z\] means 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. Example: \(a+S\) to find one or more \(a\)'s at end of line.

* Matches 0 or more times. For example, \(Sa*m\) matches \(Sm\), \(Sam\), \(Saam\), \(Saaam\) and so on.

+ Matches 1 or more times. For example, \(Sa+t\) matches \(Sam\), \(Saam\), \(Saaam\) and so on.

**Representation of special characters**
Note the following expressions.

| \r  | Carriage Return (CR). You can use either CR (\r) or LF (\n) to find or create a new line |
| \n  | Line Feed (LF). You can use either CR (\r) or LF (\n) to find or create a new line |
| \t  | Tab character |
Regular expression examples

This example illustrates how to find and replace text using regular expressions. In many cases, finding and replacing text is straightforward and does not require regular expressions at all. However, there may be instances where you need to manipulate text in a way that cannot be done with a standard find and replace operation. Consider, for example, that you have an XML file of several thousand lines where you need to rename certain elements in one operation, without affecting the content enclosed within them. Another example: you need to change the order of multiple attributes of an element. This is where regular expressions can help you, by eliminating a lot of work which would otherwise need to be done manually.

Example 1: Renaming elements

The sample XML code listing below contains a list of books. Let's suppose your goal is to replace the `<Category>` element of each book to `<Genre>`. One of the ways to achieve this goal is by using regular expressions.

```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>
  <book id="2">
    <author>Franz Kafka</author>
    <title>The Metamorphosis</title>
    <category>Fiction</category>
    <year>1912</year>
  </book>
  <book id="3">
    <author>Herman Melville</author>
    <title>Moby Dick</title>
    <category>Fiction</category>
    <year>1851</year>
  </book>
</books>
```

To solve the requirement, follow the steps below:

1. Press Ctrl+H to open the Find and Replace dialog box.
2. Click Use regular expressions.
3. In the Find field, enter the following text: `<category>(.+)</category>`. This regular expression matches all category elements, and they become highlighted.
To match the inner text of each element (which is not known in advance), we used the tagged expression \((.+)}\). The tagged expression \((.+)}\) means "match one or more occurrences of any character, that is \(+\), and remember this match". As shown in the next step, we will need the reference to the tagged expression later.

4. In the Replace field, enter the following text: \(<genre>\1</genre>\). This regular expression defines the replacement text. Notice it uses a back-reference \(\1\) to the previously tagged expression from the Find field. In other words, \(\1\) in this context means "the inner text of the currently matched <category> element".

5. Click **Replace All** and observe the results. All category elements have now been renamed to genre, which was the intended goal.

**Example 2: Changing the order of attributes**

The sample XML code listing below contains a list of products. Each product element has two attributes: id and size. Let's suppose your goal is to change the order of id and size attributes in each product element (in other words, the size attribute should come before id). One of the ways to solve this requirement is by using regular expressions.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<products xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="products.xsd">
    <product id="1" size="10"/>
    <product id="2" size="20"/>
    <product id="3" size="30"/>
    <product id="4" size="40"/>
    <product id="5" size="50"/>
    <product id="6" size="60"/>
</products>
```

To solve the requirement, follow the steps below:

1. Press **Ctrl+H** to open the Find and Replace dialog box.
2. Click **Use regular expressions**.
3. In the Find field, enter the following: \(<product id="(.+)" size="(.+)"/>\). This regular
expression matches a product element in the XML document. Notice that, in order to match the value of each attribute (which is not known in advance), a tagged expression \( (.+) \) is used twice. The tagged expression \( (.+) \) matches the value of each attribute (assumed to be one or more occurrences of any character, that is \( .+ \)).

4. In the Replace field, enter the following: `<product size="\2" id="\1"/>`. This regular expression contains the replacement text for each matched product element. Notice that it uses two references \( \1 \) and \( \2 \). These correspond to the tagged expressions from the Find field. In other words, \( \1 \) means "the value of attribute id" and \( \2 \) means "the value of attribute size".

6. Click Replace All and observe the results. All product elements have now been updated so that attribute size comes before attribute id.

### 4.1.6 Validating an XML Document

The XML | Validate (F8) command validates an XML document against an associated DTD, XML Schema, or other schema. If a document is valid, a successful-validation message is displayed in the Messages window. Otherwise, the causes of the error are displayed in the left-hand pane (see screenshot below). If a cause is selected in the left-hand pane, then smart fixes for it, if available, are displayed in the right-hand pane. Smart fix suggestions are based on information in the associated schema. To apply a smart fix, either (i) double-click it, or (ii) select it and click either the Fix or Fix + Validate options (see screenshot below).

In Text View, there are two additional indicators of a validation error (see screenshot below): (i) a red exclamation-mark icon in the line-numbering margin, and (ii) a red marker-square in the scroll bar (on the right of the window).
Note that the red marker-square appears on the left-hand side of the scroll bar (located at the right-hand side of the window; see screenshot above). This is mentioned because here because scroll bar displays two other kinds of marker-squares: (i) for highlighted text occurrences (brown, left-hand side of the scroll bar; see Navigating the Document); (ii) Find occurrences (brown, right-hand-side of the scroll bar; see Find and Replace).

If a smart fix is available for an error, then a light bulb icon is shown on the line that generates the error (see screenshot below). When you place the mouse over icon, a popup appears that lists available smart fixes (see screenshot). Select a fix to apply it immediately.

Note: Validation error indicators and quick fixes are available for document types that can be validated in XMLSpy, for example JSON documents.

Note: The validation error indicators and smart fixes described above are refreshed only when the XML | Validate (F8) command is executed; they are not updated in the background. So, after correcting an error, you must run the Validate (F8) command again to make sure that the error has indeed been fixed.

For more information about validating an XML document, see the description of the Validate command.

4.1.7 Entry Helpers in Text View

What entry helpers are available in Text View depends upon the type of document being edited. A list of entry helpers is given below for the most common document types. The general use of entry helpers is described below. Additional features for specific document types, if any, are described in the sections describing the respective document types.

- XML: Elements (screenshot below), Attributes, Entities
The entry helpers are context-sensitive and display items that may be inserted at that point.

- The item has already been inserted at the selected (or at another equivalent and valid location) and may not be inserted again at that location (for example, an XML attribute), it is displayed in gray.
- If the item is mandatory, an exclamation mark icon is displayed next to it.
- To insert an entry helper item at the cursor selection point in the text, double-click the entry-helper item.
- When an element is inserted via the Elements entry helper, its start and end tags are inserted in the document text. Mandatory elements are also inserted if this option has been specified in the Options dialog (Tools | Options | Editing).
- When an attribute is inserted via the Attributes entry helper, the attribute is inserted at the cursor point together with an equals-to sign and quotes to delimit the attribute value. The cursor is placed between the quotes, so you can start typing in the attribute value directly.

**Note:** For large files, Auto-completion and entry helpers can be disabled, thus enabling faster loading and editing. The threshold file size is specified by the user. For more details, see the reference section, Options | Editing.

### 4.1.8 Split View

Split View divides the main window of Text View in two and displays the active document in both views. You can navigate separately in each view, which enables you to see different parts of the document side-by-side. Editing changes, however, are made to the underlying document, and are reflected in both views.
Switching between Split View and Single View

Create split views of the **active document** as follows:

- **Horizontal split**: Drag down the horizontal-split icon at top right (circled in green in the screenshot below).
- **Vertical split**: Drag the vertical-split icon at bottom left (circled in blue in the screenshot below) to the right.

To return to single view from split view, do one of the following:

- Double-click the splitter bar, or
- Move the splitter bar to one of the main window's edges that are parallel to it.

**Note:** A split view is created for each document individually.

Navigating and editing in split view

The main benefit of working in Split View is that you can view different parts of a long document side-by-side, while being able to edit the document in both views. The screenshot below shows a document in a vertical Split View.
Note the following points:

- All the display features (like line numbering and source folding), editing features, navigation features, etc. that are available in the single view of a document are available in both views of Split View (see screenshot above).
- In each view of Split View, you can scroll and navigate separately.
- You can use source folding separately in each view.
- All editing actions on the document, including entry helper actions, are reflected in both views.

### 4.1.9 Text View Shortcuts

The default shortcuts of commonly used Text View commands are listed below. You can change the default shortcuts in the Keyboard tab of the Customize dialog.

#### Function-key shortcuts (incl. for validation and transformation)

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>Help Menu</td>
</tr>
<tr>
<td>F1 + Alt</td>
<td>Open Last File</td>
</tr>
<tr>
<td>F3</td>
<td>Find Next</td>
</tr>
<tr>
<td>F4 + CTRL</td>
<td>Close Active Window</td>
</tr>
<tr>
<td>F4 + Alt</td>
<td>Close XMLSpy</td>
</tr>
<tr>
<td>F5</td>
<td>Refresh</td>
</tr>
<tr>
<td>F6 + CTRL</td>
<td>Cycle through Open Windows</td>
</tr>
<tr>
<td>F7</td>
<td>Check Well-formedness</td>
</tr>
<tr>
<td>F8</td>
<td>Validate</td>
</tr>
<tr>
<td>F10</td>
<td>XSL Transformation</td>
</tr>
<tr>
<td>F10 + CTRL</td>
<td>XSL:FO Transformation</td>
</tr>
</tbody>
</table>

#### File and Application commands

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt + F1</td>
<td>Open Last File</td>
</tr>
</tbody>
</table>
CTRL + O  File Open
CTRL + N  File New
CTRL + P  File Print
CTRL + S  File Save
CTRL + F4  Close Active Window
CTRL + F6  Cycle through Open Windows
CTRL + TAB  Switch between Open Documents
Alt + F4  Close XMLSpy

Numeric keypad shortcuts (to expand/collapse nodes)

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num +</td>
<td>Expand</td>
</tr>
<tr>
<td>Num *</td>
<td>Expand Fully</td>
</tr>
<tr>
<td>Num –</td>
<td>Collapse</td>
</tr>
<tr>
<td>CTRL + Num –</td>
<td>Collapse Unselected</td>
</tr>
</tbody>
</table>

Miscellaneous keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up/Down Arrow Keys</td>
<td>Move Cursor or Selection Bar</td>
</tr>
<tr>
<td>Esc</td>
<td>Abandon Edits or Close Dialog Box</td>
</tr>
<tr>
<td>Return</td>
<td>Confirm Selection</td>
</tr>
<tr>
<td>Del</td>
<td>Delete Character or Selected</td>
</tr>
<tr>
<td>Shift + Del</td>
<td>Cut</td>
</tr>
</tbody>
</table>

Editing commands

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL + A</td>
<td>Select All</td>
</tr>
<tr>
<td>CTRL + F</td>
<td>Find</td>
</tr>
<tr>
<td>CTRL + G</td>
<td>Go to Line/Char</td>
</tr>
<tr>
<td>CTRL + H</td>
<td>Replace</td>
</tr>
<tr>
<td>CTRL + V</td>
<td>Paste</td>
</tr>
<tr>
<td>CTRL + X</td>
<td>Cut</td>
</tr>
<tr>
<td>CTRL + Y</td>
<td>Redo</td>
</tr>
<tr>
<td>CTRL + Z</td>
<td>Undo</td>
</tr>
</tbody>
</table>

Text View commands

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL + E</td>
<td>Jump between Start/End Tags</td>
</tr>
<tr>
<td>CTRL + Shift + E</td>
<td>Select Element that Contains Cursor</td>
</tr>
<tr>
<td>CTRL + Alt + E</td>
<td>Go to Parent Element</td>
</tr>
<tr>
<td>CTRL + &quot;+&quot;</td>
<td>Zoom In</td>
</tr>
<tr>
<td>CTRL + &quot;-&quot;</td>
<td>Zoom Out</td>
</tr>
<tr>
<td>Key Combination</td>
<td>Action</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>CTRL + 0</td>
<td>Reset Zoom</td>
</tr>
<tr>
<td>CTRL + mousewheel forwd</td>
<td>Zoom In</td>
</tr>
<tr>
<td>CTRL + mousewheel back</td>
<td>Zoom Out</td>
</tr>
</tbody>
</table>
4.2 Grid View

Grid View (screenshot below) shows the hierarchical structure of XML documents and DTDs through a set of nested containers, that can be easily expanded and collapsed to get a clear picture of the document's structure. In Grid View contents and structure can both be easily manipulated.

- A hierarchical item (such as the XML declaration, document type declaration, or element containing child elements) is represented with a gray bar on the left side containing a small upwards-pointing arrow at the top. Clicking the side bar expands or collapses the item. An element is denoted with the icon, an attribute with the icon.

**Display and navigation**

- The contents of an item depend on its kind. For example, in the case of elements, the contents will typically be attributes, character data, comments, and child elements. Attributes are always listed and are ordered as in the input file. Following the attributes, items appear exactly in the order found in the source file. This order can be changed using drag-and-drop, and the change will also be implemented in the source data.
- If an element contains only character data, the data will be shown in the same line as the element and the element will not be considered hierarchical by nature. The character data for any other element will be shown indented with the attributes and potential child elements and will be labeled as "Text".
- If an element is collapsed, its attributes and their values are shown in the same line in gray. This attribute preview is especially helpful, when editing XML documents that contain a large number of elements of the same name that differ only in their contents and attributes (for example, database-like applications).
- The arrow keys move the selection bar in the grid view.
- The + and – keys on the numeric keypad allow you to expand and collapse items.
Customizing Grid View

- To resize columns, place the cursor over the appropriate border and drag so as to achieve the desired width.
- To resize a column to the width of its largest entry, double-click on the grid line to the right of that column.
- To adjust column widths to display all content, select the menu item View | Optimal widths command, or click on the Optimal widths icon.
- The heights of cells are determined by their contents. They can be adjusted with the menu option Tools | Options | View | Grid View, "Limit cell height to xx lines".

Note: If you mark data in Grid View and switch to Text View, that data will be marked also in Text View.

Intelligent editing

Grid View enables intelligent editing when the XML document is based on a schema. See Editing in Grid View for details. The Entry Helpers used in Grid View are described below.

Grid View tables

Grid View allows you to display recurring elements in a table (Grid View tables). This function is available for any type of XML file: XML, XSD, XSLT, etc.

4.2.1 Editing in Grid View

When editing an XML document based on a schema (DTD or XML Schema), Grid View provides intelligent editing features based on information gathered from the schema. These features are listed below.

Inserting or appending element or attribute names

To insert or append an element or attribute relative to a selected item, you can use either commands in the XML menu or icons in the Attributes and Elements Toolbar. If this toolbar is not visible, select the menu option Tools | Customize, and, in the Toolbars tab, check the Attr & Elem entry. For a detailed explanation of how to use the toolbar icon commands and XML menu commands, see the XML menu section.

To insert or append an element or attribute:

1. Select the item relative to which the element or attribute is to be inserted or appended.
2. Click on the appropriate icon in the Attributes and Elements toolbar, or select the appropriate command from the XML menu (Insert, Append, or Add Child). This creates a new entry in the grid and opens a popup with available element or attribute options.
3. Select the desired element or attribute from the popup, and either click or press Enter. Alternatively, you can type in the name of the desired element or attribute.

You will notice that the various Entry Helpers are constantly updated depending on the current selection in the Grid View. The Info Window constantly shows important information regarding the selected element or attribute.
New lines and tabs in Grid View
To enter a new line in a Grid View cell, press Ctrl+Enter. To enter a tab, press Ctrl+Tab.

Editing an element or attribute name
When you edit the name of an element or attribute, a popup menu containing the available options opens. These options depend on the position of the element and the content model defined by the schema. A similar popup is displayed if the contents of an element or attribute are restricted by an enumeration or choice of some sort.

To edit an element or attribute name:

1. Double-click the element or attribute name. A popup with the available options appears.
2. Select the required element or attribute from the popup menu.
3. Accept the selection by hitting Return or clicking the name. (Esc causes the change to be abandoned.)

Adding namespace prefixes
The XML | Namespace Prefix command enables a namespace prefix to be added to the selected node in Grid View and all its descendant element or attribute elements. The namespace will have to be declared separately.

Grid View context menu
In addition to the commands available through the various menus and toolbars, Grid View also provides a context menu that contains the most frequently used commands collected from several menus in one convenient place. To open the context menu, right-click the item for which you wish to perform an action.

Spelling check
In Grid View, documents can be spellchecked with any of the built-in language dictionaries. A user dictionary can also be created and edited to allow words not contained in the language dictionary. For details, see the descriptions of the Spelling and Spelling Options commands.

XML Validation
User input in Grid View can be partially validated. Enumeration and pattern errors in attribute values, for example, will be picked up. But adding a child element to an element that can take only text will not be flagged as an error. To ensure that an XML document is valid, switch to Text View and validate.
4.2.2 Grid View Tables

Table View is integrated in the Grid View and allows you to view recurring elements in table form. Table View is different from the normal Grid View in that it creates a column for each child-type of the element displayed as a table. You can then modify properties for entire columns or selections. For instance, consider the following XML document:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE article SYSTEM "C:\Program Files\Altova\XMLSpy\v5\rel. 4 U\article.dtd">
<article>
  <title>Article Title</title>
  <sect1>
    <title>Section 1</title>
    <simpara/>
  </sect1>
  <sect1>
    <title>Section 2</title>
    <para/>
  </sect1>
  <sect1>
    <title>Section 3</title>
    <para/>
  </sect1>
  <sect1>
    <title>Section 4</title>
    <para/>
  </sect1>
  <appendix>
    <title/>
    <para/>
  </appendix>
</article>
```

The document element is article, and article has the following sequence of child elements: one title element, four sect1 elements, and one appendix element. Each sect1 and appendix element has one title element followed by any number of para or simpara elements. The normal Grid View of this document is as follows:

Now here is the Table View of this document—more correctly, that of the article element. (To get this view: select the article element in normal Grid View (by clicking on it) and click the Display as Table icon. Alternatively, select the menu item XML | Table | Display as Table (F12).)
Notice that each child element of article (the element displayed as a table)—that is, title, sect1, and appendix—has been assigned a column and that each occurrence of each child-type is listed in the appropriate column. Note also that the table structure extends only to the child level (the sect1 elements themselves are not displayed as a table). In order to display sect1 elements as a table, select any of the sect1 elements in the sect1 column, and click \[ \text{ } \]. The sect1 elements are now displayed in a table (see screenshot below): each of their child elements is assigned a column in the sect1 table.

In each column, if a child element (title, simpara, or para) exists for one of the four occurrences of sect1, then that cell has a white background (e.g. simpara in the first sect1). If a child does not exist for an occurrence then the corresponding cell has a gray background (e.g. para in first sect1). It should be apparent, therefore, that columns were assigned by taking a union of all child elements of all sect1 occurrences, and creating a column for each unique child-type.

Note: Attributes are also considered child nodes, and columns are also created for attributes.

You can switch between normal Grid View and Table View by selecting the desired element and clicking \[ \text{ } \] or F12. If you are viewing the document in Table View and you switch to Text View, when you switch back to Grid View the document will display in normal Grid View.

Manipulating table data

You can manipulate table data in the following ways:

- Drag-and-drop column headers to move columns.
- Sort column data for text nodes using the menu command XML | Table | Ascending Sort (also Descending Sort).
- Append (or insert) rows using the menu command XML | Table | Insert Row (also Append Row).
Moving data between Table View and external applications

You can take advantage of the table structure of data in Table View to exchange data between Table View and a spreadsheet application. To move data from Table View to another application, select the required nodes in the table and use the Copy as Structured Text option to copy/paste the data directly into, say, an Excel sheet. (You can select nodes in Table View by clicking the cells themselves, column headers, row headers, or the entire table. If you click the entire table or column headers, then the text of the column headers is also copied; otherwise it is not.)

The screenshot above shows six item elements displayed as a table in Table View, with two rows selected. To copy these two rows into an Excel sheet, copy them as structured text, and paste them into an Excel sheet. The result will be as shown below. For more details, see the description of the menu command Copy as Structured Text in the user reference section.

Data exchange works in both ways. So you can copy data from any spreadsheet-like application and insert it directly into a table in Table View. Do this as follows:

1. Select a range in the external application and copy it (to the clipboard, in Windows systems with Ctrl+C).
2. Select a single cell in Table View of your XML document.
3. Paste the copied data with Ctrl+V.

The data will be pasted into the table in XMLSpy with a cell structure corresponding to the original structure and starting from the cell selected in Table View. The following points need to be noted:

- If data already exists in these cells in Table View, the new data overwrites the original data.
- If more rows are required to accommodate the new data, these are created.
- If more columns are required to accommodate the new data, these are not created.
- The data in the cells becomes the contents of the elements represented by the respective cells.

For more complex data exchange tasks, XMLSpy also offers a set of unique conversion functions that let you directly import/export XML data from/to any text file, Word document or database.
4.2.3 Entry Helpers in Grid View

Elements Entry Helper

In Grid View, the Elements Entry Helper has three tabs: Append, Insert, and Add Child. The **Append** tab displays elements that can be appended after all the siblings of the current element; the **Insert** tab displays all elements that can be inserted before the current element; and the **Add Child** tab displays those elements you can insert as a child of the current element.

To insert an element, select the appropriate tab and double-click the required element. Note that mandatory elements are indicated with an exclamation mark. Siblings of allowed elements that cannot themselves be inserted at the cursor point are unavailable.

**Note:** In the Options dialog (Tools | Options | Editing), you can specify that mandatory child elements are inserted when an element is inserted.

Attributes Entry Helper

The Attributes Entry Helper displays a list of available attributes for the element you are currently editing. Mandatory attributes are indicated with an exclamation mark "!' before the name of the attribute. If an attribute has already been entered for that element, that attribute is shown in gray.

- To use the attributes in the Append and Insert tabs, select, in Grid View, an existing attribute or an element that is a child of the attribute's parent element, and double-click the required attribute in the Entry Helper.
- To use the attributes in the Add Child tab, select the attribute's parent element in Grid View and double-click the required attribute in the Entry Helper.

**Note:** Existing attributes, which cannot legally be added to the current element a second time, are shown in gray.
Entities Entry Helper

The Entities Entry Helper displays all parsed or unparsed entities that are declared inline or in an external DTD. If a text node or attribute node is selected in Grid View, the Add Child tab will appear empty—because, by definition, such nodes cannot have any children.

To use the cursor to insert an entity in Grid View, place the cursor at the required position in a text field or select the required field; then select the appropriate tab, and double-click the entity. Note the following rules:

- If the cursor is placed within a text field (including an attribute value field), the entity is inserted at the cursor insertion point.
- If an element with a text-only child (i.e. #PCDATA or simpleType) is selected but the cursor is not placed in the text field, then any existing text content is replaced by the entity.
- If a non-text field is selected, then the entity is created as text at a location corresponding to the Entry Helper tab that you select.

**Note:** If you add an internal entity, you will need to save and reopen your document before the entity appears in the Entities Entry Helper.

4.2.4 Grid View Shortcuts

The default shortcuts of commonly used Grid View commands are listed below. You can change the default shortcuts in the Keyboard tab of the Customize dialog.

**Function-key shortcuts (incl. for validation and transformation)**

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>Help Menu</td>
</tr>
<tr>
<td>F1 + Alt</td>
<td>Open Last File</td>
</tr>
<tr>
<td>F3</td>
<td>Find Next</td>
</tr>
<tr>
<td>F4 + CTRL</td>
<td>Close Active Window</td>
</tr>
<tr>
<td>F4 + Alt</td>
<td>Close XMLSpy</td>
</tr>
<tr>
<td>F5</td>
<td>Refresh</td>
</tr>
<tr>
<td>F6 + CTRL</td>
<td>Cycle through Open Windows</td>
</tr>
<tr>
<td>F7</td>
<td>Check Well-formedness</td>
</tr>
<tr>
<td>F8</td>
<td>Validate</td>
</tr>
<tr>
<td>F10</td>
<td>XSL Transformation</td>
</tr>
<tr>
<td>F10 + CTRL</td>
<td>XSL:FO Transformation</td>
</tr>
</tbody>
</table>

**File and Application commands**

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt + F1</td>
<td>Open Last File</td>
</tr>
<tr>
<td>CTRL + O</td>
<td>File Open</td>
</tr>
<tr>
<td>CTRL + N</td>
<td>File New</td>
</tr>
<tr>
<td>CTRL + P</td>
<td>File Print</td>
</tr>
</tbody>
</table>
CTRL + S | File Save
CTRL + F4 | Close Active Window
CTRL + F6 | Cycle through Open Windows
CTRL + TAB | Switch between Open Documents
Alt + F4 | Close XMLSpy

**Numeric keypad shortcuts (to expand/collapse nodes)**
- Num + | Expand
- Num * | Expand Fully
- Num – | Collapse
- CTRL + Num – | Collapse Unselected

**Miscellaneous keys**
- Up/Down Arrow Keys | Move Cursor or Selection Bar
- Esc | Abandon Edits or Close Dialog Box
- Return | Confirm Selection
- Del | Delete Character or Selected
- Shift + Del | Cut

**Editing commands**
- CTRL + A | Select All
- CTRL + F | Find
- CTRL + G | Go to Line/Char
- CTRL + H | Replace
- CTRL + V | Paste
- CTRL + X | Cut
- CTRL + Y | Redo
- CTRL + Z | Undo

**Grid View commands**
- CTRL + D | Append CDATA
- CTRL + E | Append Element
- CTRL + I | Append Attribute
- CTRL + M | Append Comment
- CTRL + T | Append Text
- CTRL + Shift + D | Insert CDATA
- CTRL + Shift + E | Insert Element
<table>
<thead>
<tr>
<th>Keyboard Combination</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL + Shift + I</td>
<td>Insert Attribute</td>
</tr>
<tr>
<td>CTRL + Shift + M</td>
<td>Insert Comment</td>
</tr>
<tr>
<td>CTRL + Shift + T</td>
<td>Insert Text</td>
</tr>
<tr>
<td>CTRL + Alt + D</td>
<td>Add Child CDATA</td>
</tr>
<tr>
<td>CTRL + Alt + E</td>
<td>Add Child Element</td>
</tr>
<tr>
<td>CTRL + Alt + I</td>
<td>Add Child Attribute</td>
</tr>
<tr>
<td>CTRL + Alt + M</td>
<td>Add Child Comment</td>
</tr>
<tr>
<td>CTRL + Alt + T</td>
<td>Add Child Text</td>
</tr>
</tbody>
</table>
4.3 Schema View

XML Schemas can be viewed and edited graphically in Schema View (screenshot below). The graphical interface enables you to build schemas quickly and accurately using typical GUI features. Schema View has two panes: (i) an upper pane for designing the structural relationships between schema components; and (ii) a lower pane for definitions related to the component selected in the upper pane. There are also three entry helpers that greatly facilitate the creation of valid schemas: the Components, Details, and Facets entry helpers.

Upper pane: schema design

The upper pane of Schema View can be switched between two views:

- **Schema Overview** displays all global components of the schema (such as global elements and complex types) in a simple tabular list (see screenshot). By clicking the icon of a global component you can switch to the Content Model View of that global component. Note that not all global components can have a content model (for example, simple types).
- **Content Model View** displays the content model of the selected global component (see screenshot). To return to Schema Overview, click the Show Globals icon at the top left of the upper pane.
Switch to Content Model View: Available for global components that have a content model. Opens the global component's content model in Content Model View.

Show Globals: Available in Content Model View. Switches to Schema Overview.

Lower pane: Attributes, Assertions, and Identity Constraints

The lower pane of Schema View (see screenshot) contains tabs for the definitions of Attributes, Assertions, and Identity Constraints of the component selected in the design (upper pane). We call this pane the AAIDC pane for short.

- In XSD 1.0 mode, the lower pane has two tabs: (i) Attributes, and (ii) Identity Constraints.
- In XSD 1.1 mode, the lower pane has three tabs: (i) Attributes, (ii) Assertions, and (iii) Identity Constraints.

The AAIDC pane is always present in Schema Overview and may be present in Content Model View. In Content Model View, all three types of definitions (attributes, assertions, IDCs) can be displayed in the diagram instead of in the AAIDC pane. To do this, toggle the respective Schema Design toolbar buttons on: (i) Display attributes in diagram, (ii) Display assertions in diagram, and (iii) Display identity constraints in diagram. Alternatively, you can specify these preferences in the Schema Display Configuration dialog (Schema Design | Configure View). When all the definition-types of the AAIDC pane are displayed in the diagram, the lower pane will no longer be visible in Content Model View.

Schema settings

The Schema Settings dialog (Schema Design | Schema Settings) is accessed from Schema View and lets you define global settings for the active schema. These settings are the attributes of the xs:schema element.

Organization of this section

This section is organized into the following sub-sections

- XSD Mode: XSD 1.0 or 1.1: Select between the two editing modes
- Schema Overview: Edit the properties of global components
- Content Model View: Edit the content models of individual global components
- Attributes, Assertions, and Identity Constraints: Define these particular properties of components
- Entry Helpers: Use these to quickly define various properties of components
- Smart Restrictions: Graphically create and edit derived types from base types
- Using xml: prefixed attributes: Add the base, id, lang, and space attributes graphically to schema components
- Back and Forward: Moving through Positions explains a Schema View feature that enables you to move through previously viewed positions

Connecting to SchemaAgent

From XMLSpy you can also connect to SchemaAgent in order to display components from other schemas in the GUI and to use these components in the schema being currently edited. How to work with SchemaAgent in XMLSpy is described in the section Working with SchemaAgent.
Find in schemas
The Find in Schemas features enables intelligent searches in schemas, i.e., searches that are restricted by various schema-related criteria. For example, searches may be restricted to certain component types, thus making the search more efficient. Find in Schemas is described in the DTDs and XML Schemas section.

4.3.1 XSD Mode: XSD 1.0 or 1.1
In Schema View you can select whether the XML Schema (XSD) should be edited and validated according to the XML Schema 1.0 specification (XSD 1.0) or the XML Schema 1.1 specification (XSD 1.1). The XSD mode that is used for editing a file is based on two settings: one in the application, the other in the XSD document.

Selecting XSD mode
The XSD mode determines the Schema View editing and validation features (XSD 1.0 or 1.1) available for the active document. You can either: (i) make an application-wide setting, in which case all XSD documents in Schema View will be edited in the selected mode, or (ii) you can save the XSD version number in the XSD document and let the application automatically select the XSD mode according to this information.

Application-wide mode
The application-wide setting is made in the File section of the Options dialog (Tools | Options, see screenshot below). If you select the Version 1.0 or Version 1.1 radio button, then the selected mode becomes the application-wide mode. All XML Schema documents opened in Schema View will now be edited in this mode. (If you select the v1.1 if <xs:schema vc:minVersion="1.1", v1.0 otherwise setting, the mode will depend on information in the document and will not be application-wide. See Document-specific mode and the other sections below for information about this.)

You can switch between the two application-wide modes (Version 1.0 and Version 1.1) at any time by selecting the option you want in the XML Schema Version setting of the Options dialog (screenshot above).

Note: If the current setting is an application-wide setting and you switch modes using the XSD 1.0 or XSD 1.1 button in the Schema Design toolbar (see next section), the mode switch will be temporary, and the mode will revert to the application-wide mode when the document is reloaded. A reload happens each time the view is changed or when Schema View is refreshed (via File | Reload).
Document-specific mode

You can also choose to save the XSD mode information in the XSD document itself. This would enable Schema View to automatically switch to the document's XSD mode when the document is loaded. You can add XSD mode information to an XSD document by clicking the **XSD 1.0** or **XSD 1.1** button in the Schema Design toolbar (screenshot below). On doing this, the selected mode is saved in the `vc:minVersion` attribute of the top-level `xs:schema` element. (The value of the `vc:minVersion` can also be added manually in Text View.)

![Schema Design toolbar](screenshot)

**Note:** The `vc:minVersion` attribute, if present, must be in the namespace `http://www.w3.org/2007/XMLSchema-versioning`. In this case, the XML Schema document must have a namespace declaration binding the `vc:` namespace prefix to this namespace. If you use the **XSD 1.1** toolbar button (screenshot above), the namespace is added automatically. Clicking the **XSD 1.0** toolbar button removes this namespace declaration if no other node name in the document is in this namespace.

To activate the document-specific mode and specify a document's XSD mode, do the following:

1. **Activate document-specific mode:** In the File section of the Options dialog (Tools | Options), set the XML Schema Version option to `v1.1` if `<xs:schema vc:minVersion="1.1", v1.0 otherwise (see screenshot below). This indicates to XMLSpy that the XSD mode in Schema View should be set according to the `vc:minVersion` attribute of the `xs:schema` element.

![XML Schema Version](screenshot)

2. **Specify the document's XSD version:** In the Schema Design toolbar (screenshot above), click the **XSD 1.0** or **XSD 1.1** button. A confirmation dialog (screenshot below) pops up.

![XMLSpy confirmation dialog](screenshot)

3. Clicking **Yes** results in the following: (i) enters the corresponding value in the `vc:minVersion` attribute of the `xs:schema` element, and (ii) if **XSD 1.1** was selected,
declares the XMLSchema-versioning namespace with a binding to the vc: namespace prefix; if XSD 1.0 was selected, the namespace declaration is removed if no other node is in the XMLSchema-versioning namespace. The XML Schema document now contains the XSD version number. On saving the file, the XSD mode information is saved with it. When you reopen or reload the file, Schema View will automatically switch to the document’s XSD mode as contained in the vc:minVersion attribute of the xs:schema element.

**Note:** If the document-specific mode option is selected, and 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 Schema View defaults to XSD 1.0 mode.

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

### XSD mode of new documents

When you create a new XSD document you will be prompted about whether you wish to create it as an XSD 1.0 or XSD 1.1 document. If XSD 1.1 is selected, the new document is created with the attribute /xs:schema/@vc:minVersion="1.1" and the XMLSchema-versioning namespace with a binding to the vc: namespace prefix is declared. If XSD 1.0 is selected, then neither the vc:minVersion attribute nor the XMLSchema-versioning namespace declaration is added. However, which XSD mode is actually enabled in Schema View depends also on the XML Schema Version selected in the File section of the Options dialog (Tools | Options). See the next section for details about how these two settings interact.

### The enabled XSD mode

The XSD mode that is enabled in Schema View depends on both (i) the presence/absence—and, if present, the value—of the /xs:schema/@vc:minVersion attribute of the XSD document, and (ii) the XML Schema Version option selected in the File section of the Options dialog (Tools | Options, screenshot below).

The following situations are possible. XML Schema Version in the table below refers to the selection in the XML Schema Version pane shown above. The vc:minVersion values in the table below refer to the value of the xs:schema/@vc:minVersion attribute in the XML Schema document.

<table>
<thead>
<tr>
<th>XML Schema Version</th>
<th>vc:minVersion attribute</th>
<th>XSD mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always v1.0</td>
<td>Is absent, or is present with any value</td>
<td>1.0</td>
</tr>
<tr>
<td>Always v1.1</td>
<td>Is absent, or is present with any value</td>
<td>1.1</td>
</tr>
<tr>
<td>Value of @vc:minVersion</td>
<td>Attribute has value of 1.1</td>
<td>1.1</td>
</tr>
</tbody>
</table>
Value of `@vc:minVersion` | Attribute is absent, or attribute is present with a value other than 1.1 | 1.0
--- | --- | ---

**Note:** In the situations described in the first two rows, it is possible that an XSD 1.1 schema is opened in XSD 1.0 mode and vice versa. The inconsistencies will be handled as described further below.

**XSD mode features**
The interface and editing features of Schema View will change according to which XSD mode (XSD 1.0 or XSD 1.1) is enabled.

**If XSD 1.0 mode** is enabled:
- Editing support for new XML Schema 1.1 components and properties is not available. However, if XSD 1.1 components or properties are already present in the XSD document, these will be displayed and will be available for deletion.
- Validation is performed against the XSD 1.0 specification. So, if an exclusively XSD 1.1 component or property (already) exists in the schema, a validation error is reported.

**If XSD 1.1 mode** is enabled, editing support is provided for all features of XML Schema 1.1. Validation is with respect to the XML Schema 1.1 specification.

**Handling of XSD 1.1 features in XSD 1.0 mode**
If an XSD 1.1 feature that is not supported in XSD 1.0 is present in the document (for example, an assertion), how such a feature is displayed and handled in XSD 1.0 mode is described below.

- **Assertions:** If at least one assertion is present on the selected simple type, the **Assertions** tab is present in the Facets entry helper. No editing is possible except for deletion of the assertion.
- **Asserts:** The assertion is displayed in the diagram of the complex type if present. No assertions tab is available in the AAIDC pane. Assertion cannot be added via context menu. No editing of properties is possible except for deletion.
- **Attributes:** New property **inheritable** is displayed if present. No editing is possible except for selecting the empty value (this is effectively a removal of properties).
- **Complex types:** The new property **defaultAttributesApply**, if present, is displayed in the Details entry helper. No editing is possible except for selecting the empty value (this is effectively a removal of properties).
- **Documentation:** New XSD 1.1-specific components and properties are not included in Schema View documentation.
- **Facets:** Unknown facets cause validation errors and are displayed in red.
- **Find in schemas:** New XSD 1.1-specific components and properties are ignored.
- **Identity constraints (IDCs):** The property **isRef** is displayed in case of reference and can be switched off. It will be switched off as soon as the IDC’s name is modified.
- **Multiple substitution groups:** Combo box to select single substitution group (only single substitution groups allowed in XSD 1.0).
- **Open content:** Displayed in diagram if present. Cannot be added via context menu. No editing is possible except for deletion. Default Open Content is not displayed within complex types.
- **Override:** Displayed in globals grid if present. Cannot be added via menu. No editing (of location) is possible except for deletion. Overriding components (that is, children of
xs:override) are ignored and will not be included in the Components entry helper.

- **Schema settings:** New properties defaultAttributes and xpathDefaultNamespace are displayed in dialog if present. No editing is possible except for selecting the empty value (this is effectively a removal of properties).
- **Simple types:** Unknown types cause validation errors and are displayed in red.
- **Type Alternatives:** Displayed in diagram if present. Cannot be added via context menu. No editing (of properties) is possible except for deletion.
- **Wildcards:** New properties displayed if present. No editing is possible except for the selection of empty value (effectively a removal of properties).

### 4.3.2 Schema Overview

Schema Overview (screenshot below) displays a list of all the global components of the schema (import elements, global elements, complex types, etc).

You can insert, append, or delete global components, as well as modify their properties. To modify properties, select the global component in the Schema Overview list. Depending on what kind of global component it is, its properties can be edited in the Details entry helper, the Facets entry helper, and/or the Attributes/Assertions/Identity Constraints (AAIDC) pane.

A global component that can have a content model has a Switch to Content Model View icon to its left in the global components list. Clicking this icon switches to the Content Model View of that component, where the content model of that component can be edited.
Switch to Content Model View: Available for global components that have a content model. Opens the global component's content model in Content Model View.

Show Globals: Available in Content Model View. Switches to Schema Overview.

In this section, we first describe the GUI mechanisms of Schema Overview, then describe the particulars of the various global components.

GUI Mechanisms

Global components are added as children of the top level \texttt{x:schema} element. Add a global component by clicking the Append icon or Insert icon at the top left of the upper pane (see list of icons further below), and then selecting, from the global components menu (screenshots below), the global component you want to add.

The screenshots below show the global components that can be added: XSD 1.1 mode on the left, XSD 1.0 mode on the right. (Override and Default Open Content are XSD 1.1 features.)

You can add as many global components as you like. All the global components in the schema are displayed in a tabular list in Schema Overview (screenshot below).
Editing in Schema Overview

Note the following editing features of Schema Overview:

- You can reposition components in the Schema Overview list using drag-and-drop.
- You can navigate using the arrow keys and **Tab** key of your keyboard.
- You can use cut/copy-and-paste to copy or move global components, attributes, assertions, and identity constraints from one diagram to a different position in the diagram, to other diagrams, and from one schema to another.
- Right-clicking a component opens a context menu that allows you to cut, copy, paste, delete, or edit the annotation data of that component.
- To enter a new line in global comments and global annotations, press **Ctrl+Enter**. To enter a tab, press **Ctrl+Tab**.

Schema Overview and related icons

- **Append Global Component**: Adds global components to the bottom of the global components list. If the component must, by definition, occur at the beginning of the document, it is added to the top of the list.
- **Insert Global Component**: Adds global components above the selected component. If the component must, by definition, occur at the beginning of the document, it is added to the top of the list.
- **Sort**: Pops up the Sort Components dialog, in which the precedence of sort criteria can be set (name before kind, or vice versa), before going ahead with the sorting. See description below.
- **Comments**: Pops up a menu to select between multi-line and single-line display of global comments. See description below.
- **Switch to Content Model View**: Available for global components that have a content model. Opens the global component’s content model in [Content Model View](#).
Switching between Schema Overview and Content Model View

Some global components (such as complex types, element declarations, and model groups) have a **Switch to Content Model View** icon to the left of the component name (see list of icons above). This indicates that these global components can have a content model which describes the component's structure and contents.

Clicking this icon switches the view from Schema Overview to the **Content Model View** of that global component. Other global components (such as annotations, simple types, and attribute groups) do not have a content model, and therefore do not have the **Switch to Content Model View** icon. You can switch back to Schema Overview from Content Model View by clicking the **Show Globals** icon (see list of icons above).

Sorting global components

You can sort global components by clicking the **Sort** icon in the Schema Overview toolbar (see list of icons above). In the Sort Components dialog that pops up (screenshot below), you can choose to sort either all sortable global components, or the set of selected components. You can use **Shift**+click to select a range and **Ctrl**+click to add additional components to the selection.

**Note:** Global components that must occur at the start of the document (such as *include* and *import*) are not affected by the sorting feature. They are not part of the range of global components that may be sorted.

After setting the range you can choose to sort the sortable range alphabetically (**Name before kind**), or organized first by kind and then by name.

The sort order is implemented in the text of the schema.

**Global comments: line display mode**

Global comments can be displayed in a multi-line text field (default) or a single-line text field (see screenshots below).
To switch between these two display modes of comments, click the **Comments** icon at the top of the Schema Overview pane and select the option you want. Within the text of a comment, if you wish to create a new line (and so make the comment a multi-line comment), press **Ctrl+Enter**. When comments are in single-line text-field display mode, placing the cursor over a multi-line comment pops up a multi-line box that displays all the lines.

### Global Components

Global components are those that are added as children of the top-level `xs:schema` element (as opposed to local components, which are created within other components). Some global components, such as complex types, elements and attributes can be referenced by other components in the schema.

**Creating global components in Schema Overview**

Global components are typically created and edited in **Schema Overview**. In Schema Overview, they are added via the **Append** or **Insert** icons. The content model of a global component (if the global component can have one, see table below) is created and edited in the **Content Model View** of that global component. (Click the **Switch to Content Model View** icon to the left of a component's name to go to **Content Model View**.)

Some global components, on being created in Schema Overview, are also added to the **Components entry helper**. If a component has a content model, double-clicking its name in the Components entry helper will open the content model for editing in **Content Model View**.

**Note:** You can also create some global components (elements, attributes, simple types, complex types, and model groups) while editing in Content Model View. Right-click anywhere in the window and select **New Global | < Type of Global Component >**.

**Note:** While editing in Content Model View, you can make a local element a global element—or a global complex type if the element has an element or attribute child. Select the local element, right-click anywhere in the window, and select **Make Global | Element** or **Make Global | Complex type**.

<table>
<thead>
<tr>
<th>Global component</th>
<th>Location in Schema</th>
<th>Content Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>include</code></td>
<td>Beginning</td>
<td>No</td>
</tr>
<tr>
<td><code>import</code></td>
<td>Beginning</td>
<td>No</td>
</tr>
</tbody>
</table>
Given below are key points about editing these components in Schema View.

**Includes, Imports, Redefines, and Overrides**

These four global components allow other schema documents to be reused within the current schema document.

- Includes reuse documents that have the same target namespace as that of the current document.
- Imports reuse documents that have other target namespaces as that of the current document.
- Redefines and Overrides are types of Includes in that they have the same target namespace as the current document. They, however, modify parts of the included schemas. Redefines are a 1.0 feature and are deprecated in 1.1. Overrides, which are a 1.1 feature, are more flexible and have been designed to replace Redefines in 1.1.

All four have a `schemaLocation` attribute that points to the schema to be reused. In Schema View, when you double-click in the `loc` field of these components, you can browse for the file to reuse and set its path relative to the current document. The `import` component additionally has a `namespace` attribute that holds the target namespace of the imported schema.

When a schema is reused in the current schema document (via includes, imports, redefines, or overrides), its global components, namespaces, and identity constraints are displayed in the `Components entry helper` of the current document.

**Redefines**

In a `redefine` component, you can modify complex types, simple types, model groups and attribute groups. The component to be redefined will be in the schema specified in the `loc` field of the `redefine` component (in the screenshot below the components to be redefined are in the...
After a redefine component is added, you must add the component to be redefined into a position between the redefine and end of redefine rows of the global components list (see screenshot below, where the components New and emailType are redefined). These two components exist in the schema ExpReport.xsd and are being redefined for the current schema.

<table>
<thead>
<tr>
<th>import</th>
<th>loc:address.xsd</th>
<th>ns: <a href="http://www.altova.com/IpO">http://www.altova.com/IpO</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>import</td>
<td>loc:TextState.xsd</td>
<td>ns: <a href="http://www.xmlspy.com/schemas/textstate">http://www.xmlspy.com/schemas/textstate</a></td>
</tr>
<tr>
<td>redefine</td>
<td>ExpReport.xsd</td>
<td></td>
</tr>
<tr>
<td>attributeGroup</td>
<td>New</td>
<td></td>
</tr>
<tr>
<td>simpleType</td>
<td>emailType</td>
<td></td>
</tr>
<tr>
<td>end of redefine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>notation</td>
<td>Altova-Orgchart</td>
<td></td>
</tr>
</tbody>
</table>

To redefine a component, do the following:

1. Select the end of redefine row.
2. Click the Insert icon in the top left of Schema Overview.
3. Select the kind of component you wish to define (complex type, simple type, model group or attribute group). The component is added within the redefine component.
4. Give it the same name as the component you wish to redefine. The component will now have all the properties of the component from the schema that is being reused.
5. Redefine the component by selecting it and modifying its properties in the Details and Facets entry helpers, or by modifying its content model in Content Model View (if it has a content model).

Note: You might also be able to insert the components to be redefined as follows: either from elsewhere in the global components list or from the Components entry helper, using drag-and-drop or copy-paste.

Redefined components can be referenced by other components in the schema.

Overrides

In an override element you can define the following components: complex types, simple types, global elements, global attributes, model groups, attribute groups, and notations. If, within an override element, one of these components is defined, then this component will replace, in the overridden schema, all components of the same kind that have the same name as the overriding component. The overridden schema is specified in the loc field of the Override.

Overrides differ from Redefines (see above) in that they are components defined from scratch and not based on any reused component. In Schema View, you add components for overriding similarly to how you add components for redefining. Insert the overriding component above the end of override row and then define its properties. See the ‘Redefine’ section above. The main difference between an Override and a Redefine is that when a component is added to an Override, it is not based on any component from the reused schema.

Default open content

The defaultOpenContent element is new in XSD 1.1 and specifies that one or more undefined
elements can be added to any complex type of mixed or element-only content. It is similar to the
`openContent` element (also new to XSD 1.1), the main difference being that while the
`openContent` element applies to a single complex type, the `defaultOpenContent` element
applies to all complex types in the schema.

The `defaultOpenContent` element occurs once in the document (see screenshot below), after
Includes, Imports, Redefines, and Overrides, and before the definitions of components. It has a
mode attribute which can take a value of either `interleave` or `suffix`. The default is `interleave`.

The `defaultOpenContent` element has a content model that you can edit in Content Model View.
Once declared, the `defaultOpenContent` element will apply to all complex types in the schema.
In the screenshot below, you can see that the `defaultOpenContent` has been applied
automatically to the `OfficeType` and `Desc` complex types.

To override the `defaultOpenContent` element when it is applied to a particular complex type, add
a child `openContent` element to that complex type. In the screenshot below, the `Desc` element
with the `defaultOpenContent` element (see screenshot above) has had an `openContent` element
added to it that overrides the `defaultOpenContent` element.
Global elements (element)

In Schema Overview, you can create a global element. If the global element is to have a content model, then this is defined in the Content Model View of the global element. With the element selected in either view, you can define attributes, assertions, and identity constraints in the respective tabs of the AAIDC pane. Facets and other properties can be defined in the element's Facets and Details entry helpers. Global elements can then be referenced by complex types.

Model groups (group)

In Schema Overview, you can create named model groups that can then be referenced in complex types. A named model group (the xs:group element) allows you to predefine a content model that can be reused. It can contain one of three kinds of child model group: a sequence group, a choice group, or an all group.

You create a named model group in Schema Overview by adding a Group component, giving it a name, and then defining its content model in Content Model View. The named model group can then be added to the content model of a complex type.

Named simple types (simpleType)

In Schema Overview you can create named simple types (see screenshot below), which can then be referenced in element and attribute declarations.

In the Details entry helper you specify the content of the simple type (restriction, list, union) and the corresponding type: respectively, the base type, item type, and member type. In the screenshot below, for example, the base type of the simple type's restriction is xs:integer. See the Details entry helper section for more information. To restrict a simple type with facets, use the options in the Facets entry helper.
Anonymous types can be declared on an element or attribute of simple content in either Schema Overview or Content Model View. When you set the derivedBy property (in the Details entry helper) to restriction, list or union, you create an anonymous simple type within that element or attribute declaration. You can define restriction facets (in the Facets entry helper) and other properties in the Details entry helper.

Named complex types (complexType)

In Schema Overview you can create named complex types, which can then be referenced in element declarations. With the named complex type selected in either view, you can define its attributes and assertions in the respective tabs of the AAIDC pane.

A complex type can have four types of content (see list below). You specify the various types of content in the Details entry helper as described below and, if desired (and allowed), a content model in Content Model View.

- Simple content: Set the base type of the simple content (see screenshot below). The mixed attribute (for mixed content) must have a value of false (the default value); this is why true in the screenshot below is displayed in red. No content model is allowed.
Element-only content: Create child elements in the content model diagram. There will be no base type.

Mixed content: The mixed attribute must be set to true. Character data can be present anywhere in the element among child element nodes. The character data does not have any datatype, so there must be no base type (see screenshot above). Child elements can be created in the content model diagram.

Empty content: The element will have neither character data nor child elements. There must be no base type and mixed must be false. Data in empty-content elements is typically stored in attributes.

Note: Attributes and assertions can be set (in the AAIDC pane) on all four types of content.

Note: Anonymous complex types are created within an element by creating a content model for that element in Content Model View.

Global attributes and attribute groups (attribute, attributeGroup)
Global attributes and attribute groups are added in Schema Overview.

- Properties of a global attribute are defined in the attribute's Details entry helper.
- After creating a global attribute group, you can add attributes to the group as follows: (i) Select the global attribute group in the global components list; (ii) Add attributes in the Attributes tab of the AAIDC pane; and (iii) Define the properties of each attribute in the Details entry helper of the selected attribute.

After global attributes and attribute groups have been created, they can be referenced in the declarations of elements and complex types.

Notations (notation)
Notations are always global; there are no local notations. The properties of a notation are specified in the Details entry helper of the notation. The notation's name can be specified directly in the global components list. All notations in the schema are displayed in the Components entry helper for ease of reference.
Global annotations

Global annotations are global components and are not the same as the optional annotations that are available for some global components. You can edit a global annotation in the Annotation dialog (screenshot below), which is accessed by right-clicking the Annotation global component and selecting Whole Annotation Data.

![Annotation dialog screenshot]

Each annotation can have an id attribute and multiple child documentation and/or appinfo elements. You can add documentation or appinfo elements by clicking the Append or Insert buttons at the top left of the dialog and then selecting the doc or app item from the respective combo boxes. Select a doc or app item in the top pane of the dialog and enter its content in the Content pane. If you wish to create a new line in the content (and so make the content multi-line content), press Enter. In the screenshot above, the documentation element is selected and can be seen to have two-line content. For each documentation or appinfo element, you can also enter optional source and xml:lang attributes.

In Schema Overview, only the first documentation or appinfo element of the global annotation is displayed and can be edited directly in the global components list. If that content is multi-line, placing the cursor over it reveals all the lines in a multi-line popup box. To display or edit the contents of the other documentation and/or appinfo elements, go to the Annotation dialog of that global annotation.

Note: The optional annotations that are available for some global components can also be edited via the Annotation dialog in exactly the same way as for global annotations as described above.
Comments and processing instructions

Comments and processing instructions can be inserted anywhere in the global components list in Schema Overview. They cannot be added in Content Model View. If one or more comments or processing instructions are present within simple types or complex types, they are collected and moved to the end of the enclosing object. It is therefore recommended that you use annotations instead of comments in such cases.

4.3.3 Content Model View

A content model is a description of the structure and content of a component. The following components can have content models:

- Complex types
- Elements
- Model groups
- Default open content

They are indicated in the global components list in Schema Overview with a Switch to Content Model View icon to the left of the component name.

Switch to Content Model View: Available for global components that have a content model. Opens the global component’s content model in Content Model View.

Show Globals: Available in Content Model View. Switches to Schema Overview.

Clicking on the Switch to Content Model View icon opens the Content Model View for that global component (see screenshot below). Alternatively, in Schema Overview: (i) select a component and then select the menu option Schema Design | Display Diagram, or (ii) double-click on a component’s name in the Components entry helper. Note that only one content model in the schema can be open at a time. When a content model is open, you can jump to the content model of a component within the current content model by holding down Ctrl and double-clicking the required component.

General description of Content Model View

The content model is displayed in the Content Model View as a tree (see screenshot below). You can configure the appearance of the tree in the Schema Display Configuration dialog (menu item Schema Design | Configure view).
Note the following:

- Objects in the content model tree are of two types: compositors and components. Additionally, attributes, assertions, identity constraints, and open content can be shown in boxes attached to the component.
- Each level in the tree is joined to adjacent levels with a compositor. The content model can extend an unlimited number of layers deep.
- An object can be added relative to another object via the latter's context menu (accessed by right-clicking the latter object).
- Components in the content model can be local components or can reference global components.
- Drag-and-drop functionality enables objects to be moved around.
- Keyboard shortcuts can be used to copy (Ctrl+C) and paste (Ctrl+V) objects.
- The properties of an object can be edited in the Details entry helper and in the AAIDC pane.
- The attributes, assertions, and identity constraints of a component are displayed in a pane below Content Model View, the AAIDC pane. Attributes and identity constraints can also be displayed in the Content Model diagram instead of in the AAIDC pane. This viewing option can be set in the Schema Display Configuration dialog. Alternatively, you can use the three Display in Diagram buttons of the Schema Design toolbar (screenshot below).

- Sibling components can be sorted by selecting them, right-clicking, and selecting the Sort Components command from the context menu. You can prioritize by one of two sort criteria: (i) local name, and (ii) component kind.
These features are explained in detail in the subsections of this section and in the tutorial.

To return to Schema Overview, click the Show Globals icon or select the menu option Schema design | Display All Globals.

**Content Model Objects**

In Content Model View, the objects shown in the diagram are best organized in three broad groups:

- **Compositors**: (i) sequence, (ii) choice, (iii) all
- **Components**: (i) element, (ii) complex type, (iii) model group, (iv) wildcard
- **Miscellaneous**: (i) attribute, (ii) attribute group, (iii) assertion, (iv) constraint, (v) open content

The graphical representations of these objects are described individually below.

**Compositors**

A compositor defines the order in which child elements occur. There are three compositors: sequence, choice, and all.

To insert a compositor:

1. Right-click the element to which you wish to add child elements
2. Select **Add Child | Sequence** (or **Choice** or **All**).

The compositor is added, and will look as below:

- **Sequence**

![Sequence Diagram]

- **Choice**

![Choice Diagram]

- **All**

![All Diagram]
To change the compositor, right-click the compositor and select **Change Model | Sequence** (or **Choice** or **All**). After you have added the compositor, you will need to add child element/s or a model group.

**Components**

Given below is a list of components that are used in content models. The graphical representation of each provides detailed information about the component’s type and structural properties.

- **Mandatory single element**

  ![Country](image)

  **Details:** The rectangle indicates an element and the solid border indicates that the element is required. The absence of a number range indicates a single element (i.e. minOcc=1 and maxOcc=1). The name of the element is Country. The blue color indicates that the element is currently selected; (a component is selected by clicking it). When a component is not selected, it is white.

- **Single optional element**

  ![Location](image)

  **Details:** The rectangle indicates an element and the dashed border means the element is optional. The absence of a number range indicates a single element (i.e. minOcc=0 and maxOcc=1). Element name is Location.

  **Note:** The context menu option **Optional** converts a mandatory element into an optional one.

- **Mandatory multiple element**

  ![Alias](image)

  **Details:** The rectangle indicates an element and the solid border indicates that the element is required. The number range 1..5 means that minOcc=1 and maxOcc=5. Element name is Alias.

- **Mandatory multiple element containing child elements**
Details: The rectangle indicates an element and the solid border indicates that the element is required. The number range 1..infinity means that minOcc=1 and maxOcc=unbounded. The plus sign means complex content (i.e. at least one element or attribute child). Element name is Division.

Note: The context menu option Unbounded changes maxOcc to unbounded. Clicking on the + sign of the element expands the tree view and shows the child elements.

- Element referencing global element

Details: The arrow in the bottom-left means the element references a global element. The rectangle indicates an element and the solid border indicates that the element is required. The number range 1..infinity means that minOcc=1 and maxOcc=unbounded. The plus sign indicates complex content (i.e. at least one element or attribute child). Element name is xs:field.

Note: A global element can be referenced from within simple and complex type definitions, thus enabling you to re-use a global declaration at multiple locations in your schema. You can create a reference to a global element in two ways: (i) by entering a name for the local element that is the same as that of the global element; and (ii) by right-clicking the local element and selecting the option Reference from the context menu. You can view the definition of a global element by holding down Ctrl and double-clicking the element. Alternatively, right-click, and select Go to Definition. If you create a reference to an element that does not exist, the element name appears in red as a warning that there is no definition to refer to.

- Complex type

Details: The irregular hexagon with a plus sign indicates a complex type. The complex type shown here has the name keybase. This symbol (the irregular hexagon with a plus
sign) indicates a global complex type. A global complex type is declared in the Schema
Overview, and its content model is typically defined in Content Model View. A global
complex type can be used either as (i) the data type of an element, or (ii) the base type
of another complex type by assigning it to the element or complex type, respectively, in
the Details entry helper (in either Content Model View or in Schema Overview).

The keybase complex type shown above was declared in Schema Overview with a base
type of xs:annotated. The base type is displayed as a rectangle with a dashed gray
border and a yellow background color. Then, in Content Model View, the child elements
xs:selector and xs:field were created. (Note the tiny arrows in the bottom left corner
of the xs:selector and xs:field rectangles. These indicate that both elements
reference global elements of those names.)

A local complex type is defined directly in Content Model View by creating a child
element or attribute for an element. There is no separate symbol for local complex types.

Note: The base type of a content model is displayed as a rectangle with a dashed gray
border and a yellow background color. You can go to the content model of the base type
by double-clicking its name.

- Model group

Details: The irregular octagon with a plus sign indicates a model group. A model group
allows you to define and reuse element declarations.
Note: When the model group is declared (in Schema Overview) it is given a name. You
subsequently define its content model (in Content Model View) by assigning it a child
compositor that contains the element declarations. When the model group is used, it is
inserted as a child, or inserted or appended within the content model of some other
component (in Content Model View).

- Wildcards

Details: The irregular octagon with any at left indicates a wildcard.
Note: Wildcards are used as placeholders to allow elements not specified in the schema
or from other namespaces. **other** = elements can belong to any namespace other than the target namespace defined in the schema; **any** = elements can belong to any namespace; **targetNamespace** = elements must belong to the target namespace defined in the schema; **local** = elements cannot belong to any namespace; **anyURI** = elements belong to the namespace you specify.

### Miscellaneous objects

Miscellaneous objects are attributes, attribute groups, assertions, identity constraints, and open content.

- **Attributes, Attribute Groups**

  ![Attributes Diagram]

  *Details:* Indicated with the word ‘attributes’ in italics in a rectangle that can be expanded. Each attribute is shown in a rectangle with a (i) dashed border if the attribute is optional, or (ii) a solid border if the attribute is required (mandatory). Attribute groups and attribute wildcards are also included in the ‘attributes’ rectangle.

  *Note:* Attributes can be edited in the diagram and in the Details Entry Helper. Attributes can be displayed in the Content Model View diagram or in the AAIDC pane below the Content Model View. You can toggle between these two views by clicking the Display Attributes icon. To change the order of attributes of an element, drag the attribute and drop when the arrow appears at the required location.

- **Assertions**

  ![Assertions Diagram]

  *Details:* Indicated with the word ‘assertions’ in italics in a rectangle that can be expanded. Each assertion is shown in a rectangle within the Assertions box.

  *Note:* Assertions can be edited in the diagram and in the Details Entry Helper. They can be displayed in the Content Model View diagram or in the AAIDC pane below the Content Model View. You can toggle between these two views by clicking the Display Assertions icon. To change the order of assertions on an element, drag the assertion and drop when the arrow appears at the required location.

- **Identity constraints**
Details: Indicated with the word ‘constraints’ in italics in a rectangle that can be expanded.

Note: The identity constraints listed in the content model of a component show constraints as defined with the `key` and `keyref` elements, and with the `unique` element. Identity constraints defined using the ID datatype are not shown in the content model diagram, but in the Details Entry Helper. Identity constraints can be displayed and edited in the Content Model View or in the Identity Constraints tab of Schema Overview. In Content Model View, you can toggle the Constraints box on and off with the Display Constraints icon.

- Conditional Type Assignment

Details: The `alternative` element is a rectangle containing the XPath expression that will be tested (see screenshot above). The type of the `alternative` element is specified in the Details entry helper. If the type is a complex type, it is shown in the `alternative` element's expanded rectangle and can be further edited there (see screenshot below). Simple types are not shown in the diagram, but can be defined in the Simple Type tab of the Details entry helper.
Note: The alternative element is new in XSD 1.1. If the XPath expression evaluates to true, the type specified by the alternative element will be the selected type. The first alternative element from among the alternative siblings to evaluate to true is selected. So the order of alternative elements is important. The order can be changed by dragging the alternative element boxes into the desired order. See the section Conditional Type Assignment for a detailed description.

- Default Open Content, Open Content

```
defOpenCont suffix any #local
```

Details: The defaultOpenContent and openContent elements are indicated in Content Model View with the labels openContent and defOpenContent. Wildcard element content is indicated with an any box (see screenshot above).

Note: The defaultOpenContent and openContent elements are new in XSD 1.1. Default Open Content is a global component and is created in Schema Overview. In the Content Model View of a particular component's content model, you can replace the Default Open Content with Open Content specific to that component that overrides the schema's Default Open Content. Simply add Open Content as a child of the component. The Default Open Content box will be replaced by an Open Content box. In Content Model View, you can edit the mode attribute of the Open Content and the namespace of its wildcard element, both in the diagram and in the Details entry helper. You can also modify the Default Open Content (for the whole schema) from within its representation in the Content Model View of any complex type.

Note:

- Predefined details you have specified in the Schema Display Configuration dialog can be turned on and off by clicking the Add Predefined Details toolbar icon.
- You can toggle Attributes, Assertions, and Identity Constraints to appear either in the diagram of the content model itself or in the AAIDC pane (below Content Model View) by clicking the Display in Diagram icons for attributes, assertions, and identity constraints, respectively.
- In Content Model View, you can jump to the content model view of any global component within the current content model by holding down the Ctrl key and double-clicking the required component. You can go to the content model of a base type by double-clicking the name of the base type.

Editing in Content Model View

The description of how to edit in Content Model View is organized into the following sections:

- Configuring Content Model View
- Attributes, Assertions, and Identity Constraints
- Content Model View icons
- Context menu operations
- Keyboard shortcuts and drag-and-drop
• Component properties
• Annotations
• Comments and processing instructions
• Documenting the content model

Configuring Content Model View
You can configure the content model view for the entire schema in the Schema display configuration dialog (Schema Design | Configure View). For details about configuration options, see the Configure View section later in the User Reference. Note that the settings you define here apply to the whole schema, and to the schema documentation output as well the printer output.

Attributes, Assertions, and Identity Constraints
The attributes, assertions, and identity constraints of a component can appear in a pane below the Content Model View, the AAIDC pane, or as boxes in the Content Model View itself, that is, in the diagram. This second viewing option can be set in the Schema Display Configuration dialog. Alternatively, you can use the three Display in Diagram toolbar buttons in the Schema Design toolbar (screenshot below, also see icon list below).

For a description of how to insert and edit attributes, assertions, and identity constraints, see the section, Attributes, Assertions, and Identity Constraints.

Content Model View icons

Show Globals: Available in Content Model View. Switches to Schema Overview.

Add Predefined Details: In the Schema Design toolbar and enabled in Content Model View. Toggles the display of predefined details in components on and off.

Display Attributes in Diagram: In the Schema Design toolbar and enabled in Content Model View. Toggles the display of attributes between the diagram (toggled on) and the Attributes tab.

Display Assertions in Diagram: In the Schema Design toolbar and enabled in Content Model View. Toggles the display of assertions between the diagram (toggled on) and the Assertions tab.

Display Constraints in Diagram: In the Schema Design toolbar and enabled in Content Model View. Toggles the display of IDCs between the diagram (toggled on) and the Identity Constraints tab.

Visualize Identity Constraints: In the Schema Design toolbar and enabled in Content Model View. Toggles the display of IDC information on and off.

Context menu operations
Several editing operations in Content Model View are carried out via the context menu (screenshot below) that appears when you right-click within Content Model View. Only commands for
operations allowed at that point in the content model diagram are enabled. Operations are carried out relative to the right-clicked object. For example, when a child is added, it is added relative to the right-clicked object.

Given below is a list of operations available via the context menu.

- **Add child compositors and components**: The **Add Child** command pops up a sub-menu. Click the required compositor or component in the sub-menu.

- **Insert/Append compositors and components**: Inserts the compositor or component at the same hierarchical level as the right-clicked object, before the selected object (**Insert**) or after the last sibling of the selected object (**Append**).

- **Change a compositor**: Right-click a compositor, select **Change Model | <new compositor>**.

- **Create global components**: (i) The **New Global** command can be accessed by clicking anywhere in Content Model View. It pops out a sub-menu in which you can select the new global component you wish to create. (ii) If an object can be created as a global component, the **Make Global** command in its context menu is enabled. On selecting this command, the object in Content Model View is created as a global component and the object itself will contain a reference to the newly created global component.
- **Change the occurrence definition:** Use the Optional and Unbounded toggle commands together to obtain the desired occurrence setting: (i) optional = 0 or 1; (ii) optional + unbounded = 0 to infinity; (iii) unbounded = 1 to infinity; (iv) not optional + not unbounded = 1. (Note: optional sets the minOccurs attribute of the component, unbounded sets the maxOccurs attribute.)

- **Toggle between local and global definitions:** If a global element exists that has the same name as a local element, use the Reference toggle command to switch between referencing the global definition (toggle on) and using the local definition (toggle off).

- **Jump to another content model:** Right-click the component whose content model you wish to jump to and select Go to Definition. (The command will be enabled only for those components that can have a content model.) Alternatively, you can press Ctrl and double-click the component.

- **Edit predefined details:** If predefined details have been set to be displayed in the diagram (with the Add Predefined Details icon in the Schema Design toolbar), then the Edit command pops up a submenu containing the predefined details that can be edited. Select the required predefined detail, and edit its value in the diagram.

- **Create and edit compositor/component annotation:** The Edit Annotation command creates annotation space below the compositor/component (see screenshot below). You can enter and edit the annotation here. If the annotation already exists, clicking the command highlights the annotation text for editing. Double-clicking existing annotation text is a faster way of starting an edit.

In the XML Schema document, the annotation is created inside the compositor or component's annotation/documentation element. Also see the section below about documentation.

- **Copy XPath of instance node:** The command Copy XPath of Instance Node is enabled for elements and attributes defined within a global element or global complex type. It copies to the clipboard an XPath expression that locates the selected node. The location path expression starts at the global component whose content model is currently being displayed in Content Model View.

- **Find and rename component:** The commands Find All References and Rename with All References are enabled for global elements. These, respectively, find all occurrences of the selected component and rename all occurrences of the selected component in the active document and, optionally, in all schema files related to the active document.
In the screenshot above the name Email will replace the name of the right-clicked component and of all its references within the search scope. See Finding and Renaming Globals for details.

- **Sort declarations and references**: Using the Sort command, all selected components or the siblings of the selected component can be sorted. Make your sort settings in the Sort Components dialog (screenshot below) and click OK.

To select multiple components, press the **Shift** or **Ctrl** key while clicking. You can sort using component names as the first sort key and component kind as the second sort key, or vice versa.

**Keyboard shortcuts and drag-and-drop**

You can copy and paste elements in Content Model View using the shortcuts **Ctrl+c** and **Ctrl+v**. Copied objects are pasted as child objects of the selected object. Where this is not possible for structural reasons, a message to this effect is displayed.

You can also drag-and-drop: (i) objects to other locations in the diagram, (ii) some components, such as attributes, from the Components entry helper into the diagram.
Component properties
If Content Model View is configured so that components are displayed with predefined details in the component box, then you can edit this information directly in the diagram. The display of predefined details can be turned on and off by clicking the Add Predefined Details toolbar icon (see icon list above).

Alternatively, you can edit a component’s properties in the Details entry helper, and changes will be reflected in the placeholder fields—if these are configured to be displayed.

Annotations
XML Schema annotations are held in the annotation element. There are two types of annotation, each of which is contained in a different child element of annotation:

- documentation child: Contains information that could be useful for editors of the schema
- appinfo child: Allows you to insert a script or information that a processing application may use

Given below is the text of an annotation element that contains both types of child elements.

```xml
<xs:element name="session_date" type="xs:dateTime" nillable="true">
  <xs:annotation>
    <xs:documentation>Date and time when interview was held</xs:documentation>
  </xs:annotation>
</xs:element>
```

In Content Model View, you can create annotation for individual compositors and components as follows.

1. Right-click the compositor or component.

   ![Annotation Example](image)

   Date and time when interview was held

2. Select the context menu option Whole Annotation Data. The Annotation dialog box opens (see screenshot below). If an annotation (either documentation or appinfo) exists for that element, then this is indicated by a corresponding row in the dialog.
3. To create an appinfo element, click the Append or Insert icon at top left to append or insert a new row, respectively.
4. In the Kind field of the new row, select the app option from the dropdown menu.
5. In the Content pane of the dialog, enter the script or info that you want to have processed by a processing application.
6. Optionally, in the Source field, you can enter a source URI where further information can be made available to the processing application.

Comments and processing Instructions
When XML Schema documents are loaded in XMLSpy, or when views are changed, comments and processing instructions within simple types and complex types are collected and moved to the end of the enclosing object. It is therefore recommended that you use annotations instead of comments in such cases.

Documenting the content model
You can generate detailed documentation about your schema in HTML and MS Word formats. Detailed documentation is generated for each global component, and the list of global components is displayed in a table-of-contents page that allows you to link to the content models of individual components. Additionally, related elements (such as child elements or complex types) are referenced by hyperlinks, thus enabling you to navigate from element to element. To generate schema documentation, select the menu command Schema design | Generate documentation.

Conditional Type Assignment
Conditional type assignment is an XSD 1.1 feature that allows the type of an element to be determined by content in the XML document, specifically by the value of the element's attributes or by the presence or absence of attributes. For example, say the XML document has the following element:

```
<publication kind="magazine">
```
In the schema, the type of the publication element can be specified to vary according to the value of the instance element's @kind attribute value. In the schema, this is done using the alternative element, which is new in XSD 1.1. Multiple types are specified, each in an alternative element.

In the screenshot below, the Publication element is declared with three alternative child elements, two of which have test attributes (@kind eq 'magazine' and @kind eq 'book'). The third alternative element has no test attribute and a simple type assignment of xs:error (assigned in the Details entry helper, not shown in the diagram), which, if triggered, returns an XML validation error.

The listing for the above declarations is given below:

```xml
<xs:complexType name="PublicationType">
    <xs:sequence>
        <xs:element name="Title" type="xs:string"/>
        <xs:element name="Author" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Date" type="xs:gYear"/>
    </xs:sequence>
    <xs:attribute name="kind" type="xs:string"/>
</xs:complexType>

<xs:complexType name="MagazineType">
    <xs:complexContent>
        <xs:restriction base="PublicationType">
            <xs:sequence>
                <xs:element name="Title" type="xs:string"/>
                <xs:element name="Date" type="xs:gYear"/>
            </xs:sequence>
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>
```
Note the following points:

- The first alternative element from among the alternative siblings to evaluate to true is selected. So the order of alternative elements is important. In Content Model View, the order can be changed by dragging the alternative element boxes into the desired order.
- Notice that the Publication element has a type (PublicationType). This type serves as the default type if none of the alternative elements are used. In our example above, however, the alternative element of type xs:error will be used if both the previous alternative element conditions return false.
- If no alternative element condition is met and if the element has no default type, then the element is assigned a type of anyType. In this event, the element may have any well-formed XML content.
- The alternative element and the simple type xs:error are new in XSD 1.1.

Content Model View editing

You can add an alternative type to an element declaration as a child via the element's context menu (see the content model in screenshot above).

The type of the alternative element is specified in the Details entry helper. If the type is a complex type, it is shown in the alternative element's expanded rectangle and can be further edited there (see screenshot below). Simple types are not shown in the diagram, but can be defined in the Simple Type tab of the Details entry helper.
Note: You can specify a namespace for the XPath expression via the `xpathDefaultNamespace` attribute in the Details entry helper. For more information about XPath default namespaces, see the section below.

Using `xpathDefaultNamespace`

A default namespace declared in the XML Schema document is the default namespace of the XML Schema document. It applies to unprefixed element names in the schema document but not to unprefixed element names in XPath expressions in the schema document.

The `xpathDefaultNamespace` attribute (a new feature in XSD 1.1) is the mechanism used to specify the namespace to which unprefixed element names in XPath expressions belong. XPath default namespaces are scoped on the XML Schema elements on which they are declared. The `xpathDefaultNamespace` attribute can occur on the following XML Schema 1.1 elements:

- `xs:schema`
- `xs:assert` and `xs:assertion`
- `xs:alternative`
- `xs:selector` and `xs:field` (in identity constraints)

The `xpathDefaultNamespace` on `xs:schema` is set, in XSD 1.1 mode, in the Schema Settings dialog (`Schema Design | Schema Settings`). For the other elements listed above, the `xpathDefaultNamespace` attribute is set in their respective Details entry helpers (see screenshot below for example).
Declaring the XPath default namespace on `xs:schema`, declares the XPath default namespace for the scope of the entire schema. You can override this declaration on elements where the `xpathDefaultNamespace` attribute is allowed (see list above).

Instead of containing an actual namespace, the `xpathDefaultNamespace` attribute can contain one of three keywords:

- `##targetNamespace`: The XPath default namespace will be the same as the target namespace of the schema
- `##defaultNamespace`: The XPath default namespace will be the same as the default namespace of the schema
- `##local`: There is no XPath default namespace

If no XPath default namespace is declared in the document, unprefixed elements in XPath expressions will be in no namespace.

**Note:** The XPath default namespace declaration does not apply to attributes.

**Open Content Models**

Open content models are new to XSD 1.1. They are declared on complex types and allow any element (that is, an element undefined in the content model of the complex type) to occur any number of times either (i) between elements defined in the content model, or (ii) after elements defined in the content model.

The `openContent` element is a child of the complex type and occurs once before the content model of the complex type (see screenshot below).

**Mode**

The `openContent` element has a mandatory `mode` attribute, which can take the values `interleave`, `suffix`, or `none` (see screenshot below). The default value is `interleave`.
The significance of these values is as follows:

- **If mode="interleave" or mode="suffix"**, then wildcard element content (xs:any) with no minimum or maximum number of occurrences must be present. This implies that any number of undefined elements (wildcards) is allowed.
- If the mode is **interleave**, any number of undefined elements can occur before or after individual defined elements in the content model. They are interleaved between defined elements.
- If the mode is **suffix**, any number of undefined elements can occur after the last defined element of the content model.
- If the mode is **none**, no undefined element (xs:any child) may occur; the content model is not open. The **none** value is used to override the defaultOpenContent element that is scoped on the entire schema.

In Content Model View, you add open content as a child of the complex type (via Add Child in the context menu). Specify the mode either by double-clicking in the openContent box in the diagram (see screenshot above) and selecting a value (interleave, suffix, or none), or by selecting a value in the Details entry helper.

**Wildcard (xs:any) properties**

Wildcard properties are specified in the wildcard's Details entry helper. Select the wildcard in the diagram and enter property values in the Details entry helper.

**Default open content**

The defaultOpenContent element is new in XSD 1.1 and specifies that one or more undefined elements can be added to any complex type of mixed or element-only content. It is similar to the openContent element (also new to XSD 1.1), the main difference being that while the openContent element applies to a single complex type, the defaultOpenContent element applies to all complex types in the schema.

The defaultOpenContent element is a **global component** and occurs once in the document (see screenshot below), after Includes, Imports, Redefines, and Overrides, and before the definitions of components. It has a **mode** attribute which can take a value of either **interleave** or **suffix**. The default is **interleave**.
The `defaultOpenContent` element has a content model that you can edit in Content Model View, in exactly the same way as the `openContent` element is defined (see above). Once declared, the `defaultOpenContent` element will apply automatically to all complex types in the schema and will be displayed in their content models. In the screenshot below, you can see that the `defaultOpenContent` has been applied automatically to the `OfficeType` and `Desc` complex types.

To override the `defaultOpenContent` element when it is applied to a particular complex type, add a child `openContent` element to that complex type. In the screenshot below, the `Desc` element with the `defaultOpenContent` element (see screenshot above) has had an `openContent` element added to it that overrides the `defaultOpenContent` element.
4.3.4 Attributes, Assertions, and Identity Constraints

The Attributes/Assertions/Identity Constraints (AAIDC) pane (screenshot below) is located below the main pane in Schema Overview and Content Model View. The pane and its tabs are fixed. In Content Model View, however, the view of each tab can be switched individually so that the tab's components can be viewed and edited in the diagram in Content Model View rather than in the AAIDC pane. When the views of all three tabs are switched to the diagram, the AAIDC pane disappears.

![Attributes, Assertions, Identity Constraints Pane](image)

Views can be switched between the AAIDC pane and the diagram via the Schema Display Configuration dialog (Schema Design | Configure View | Element tab) or by clicking the respective icon in the Schema Design toolbar (shown below).

- **Display Attributes in Diagram**: Enabled in Content Model View. Toggles the display of attributes between the diagram (toggled on) and the Attributes tab.
- **Display Assertions in Diagram**: Enabled in Content Model View. Toggles the display of assertions between the diagram (toggled on) and the Assertions tab.
- **Display Constraints in Diagram**: Enabled in Content Model View. Toggles the display of IDCs between the diagram (toggled on) and the Identity Constraints tab.

Using the tabs

The tabs in the AAIDC become enabled individually according to what component is selected in the upper main pane of Schema Overview or Content Model View. For example, since it is possible to add an attribute to a complex type, the Attributes tab will be enabled when a complex type is selected in the main pane. (A tab is considered to be enabled when its commands are enabled.)

How to use each of the tabs is discussed in the sub-sections of this section:

- Attributes, Attribute Groups, Attribute Wildcards
- Assertions
- Identity Constraints
Sorting attributes and identity constraints

You can sort the attributes and identity constraints in their respective tabs by clicking the Sort icon in the tab's toolbar. In the Sort Components dialog that pops up (screenshot below), you can choose to sort either the selected single component and its siblings or the set of selected components. In the screenshot above, for example, three attributes have been selected (highlighted blue). You can use click+Shift to select a range and click+Ctrl to add additional components to the selection.

After selecting the set of components to sort you can choose to sort alphabetically first on name and then on kind (Name before kind), or vice versa (Kind before name). The sort order is immediately implemented in the text of the schema document; it is not just an interface mask.

Attributes, Attribute Groups, Attribute Wildcards

In the Attributes tab of the Attributes/Assertions/Identity Constraints (AAIDC) pane (screenshot below), you can:

- Declare attributes locally on the selected complex type
- Reference attribute groups for use on the selected complex type
- Define attribute wildcards on the selected complex type

Note: If you have chosen the option to display attributes in the diagram (Schema Design | Configure View) rather than in the AAIDC pane, you can edit the properties of attributes, attribute group references, and attribute wildcards in the diagram and Details entry helper.

Display Attributes in Diagram: Enabled in Content Model View. Toggles the display of attributes between the diagram (toggled on) and the Attributes tab.

Attributes

In the Attributes tab of the Attributes/Assertions/Identity Constraints (AAIDC) pane (screenshot below), you can declare local attributes of elements and complex types, and the attributes that constitute attribute groups.
To create attributes, do the following:

1. In Schema Overview, select the complex type or attribute group for which you wish to create the attribute.
2. In the Attributes tab, click the Append or Insert icon at top left and select Attribute.
3. In the row that is created for the attribute, enter the attribute's details (name, type, use, and default or fixed value). The name property is mandatory, and the default value of use is optional. The datatype and default/fixed value properties are optional.

Note: Attributes can be added to attribute groups only in Schema Overview, but to complex types in both Schema Overview and Content Model View.

Default values and fixed values
A default value or fixed value, if specified in an attribute declaration, is applied in the instance document when that attribute is absent in the instance document. Either a default value or a fixed value can be specified, not both (see screenshot above). The default or fixed value must be valid according to the attribute's datatype. If use is set to required, then neither default nor fixed value is allowed.

Note the following:

- Default values: A default value is inserted only if the attribute is missing. If the attribute is present and has a valid value, the default value is not inserted. If the value in the instance documents is invalid, an error is raised.
- Fixed values: A fixed value is applied not only when the attribute is missing but also if the value in the instance document is not equal to the fixed value specified in the attribute's declaration.

Note: Default and fixed values can be specified on both local and global attributes. On local attributes they can be defined in both the Attributes tab of the AAIDC pane (screenshot above) and in the Details entry helper. On global attributes, they can be specified in the Details entry helper.

Attribute group references
If a global attribute group has been declared, you can add a reference to this attribute group in the definition of a complex type. Do this by selecting the complex type component in Schema Overview or Content Model View, then clicking the Append or Insert icon at top left of the
Attributes tab of the AAIDC pane and selecting **Attribute Group**. In the attribute group row that is created, enter the name of the attribute group to be referenced (see screenshot below, which has **two attribute group references**). You can add multiple attribute groups.

![Screenshot of Attributes tab with two attribute groups](image1)

When the attribute group is selected in the Attributes tab, its properties can also be edited in the Details entry helper.

**Attribute wildcards: anyAttribute**

An attribute wildcard can be added to a complex type to allow the use of any attribute on an element. An attribute wildcard is defined with a single `anyAttribute` element. It would allow any number of attributes from the specified namespace to occur on the element in the instance document.

Add an attribute wildcard by selecting the complex type component in Schema Overview or Content Model View, then clicking the **Append** or **Insert** icon at top left of the Attributes tab of the AAIDC pane and selecting **Any Attribute**. A row for the attribute wildcard `anyAttribute` is created (see screenshot below).

![Screenshot of Attributes tab with anyAttribute](image2)

In the Attributes tab, you can set the `namespace` property of `anyAttribute`. With the attribute wildcard selected in the Attributes tab, you can set additional properties in the Details entry helper (see screenshot above). Note that the `notNamespace` and `notQName` properties are **XSD 1.1 features** and so will not be available in **XSD 1.0 mode**.

**Sorting attributes and attribute groups**

You can sort the attributes and attribute groups in the Attributes tab by clicking the **Sort** icon in the tab’s toolbar. In the Sort Components dialog that pops up (screenshot below), you can choose to sort either the selected single component and its siblings or the set of selected components. You can use **Shift+**click to select a range and **Ctrl+**click to add additional components to the selection.
After setting the range you can choose to sort the entire range of attributes and attribute groups alphabetically (*Name before kind*), or attributes sorted alphabetically before attribute groups sorted alphabetically.

The sort order is immediately implemented in the text of the schema document; it is not just an interface mask.

**Note:** Attribute wildcards will not be included in the range to sort since they must always occur at the end of a complex type declaration and only one attribute wildcard is allowed in a single complex type declaration.

**Assertions**

The assertions described in this section are **assertions on complex types**. Such an assertion is defined in an `xs:assert` element and serves as a validity constraint on the complex type. (The other kind of assertion is an assertion on a simple type, which is defined in an `xs:assertion` element and is created and edited in the **Facets entry helper** of a simple type.)

In Schema View, complex type assertions can be created and edited via the following GUI access points:

- **In Schema Overview**: In the Assertions tab of the Attributes/Assertions/Identity-Constraints (AAIDC) pane (screenshot below).

- **In Content Model View**: Assertions can be edited in the Assertions tab (screenshot above) or in the diagram (screenshot below). To select the diagram display option, click
the **Display Assertions in Diagram** icon in the Schema Design toolbar. In the diagram, select the Assertion box of the complex type or complex-content element. Then enter or edit the Assertion's definition in the Details entry helper.

![Display Assertions in Diagram](image)

**Display Assertions in Diagram**: Enabled in Content Model View. Toggles the display of assertions between the diagram (toggled on) and the Assertions tab.

**Note**: Assertions are an XSD 1.1 feature. So the Assertions editing features will be available only in [XSD 1.1 mode](https://www.w3.org/TR/xmlschema-2/).

**Scope of the assertion**

The XPath expression used to define the assertion's constraint must be within the scope of the complex type on which it is defined. So if the XPath expression is required to access a particular node, then the assertion must be defined on an ancestor of that node.

**Adding and deleting assertions**

A complex type can have multiple assertions. The XPath expression of each assertion must evaluate to boolean `true` for the element in the instance document to be valid. To add an assertion to a complex type, do the following:

- **In Schema Overview**: Select the complex type. Then, in the Assertions tab of the AAIDC pane (see screenshot above), click the **Add** or **Insert** icon at the top left of the tab. You can add multiple assertions. To delete an assertion, select it and click the **Delete** icon at the top right of the tab.

- **In Content Model View (see screenshot above)**: Right-click the complex type and select **Add Child | Assertion**. Alternatively, right-click an existing assertion in the diagram of the complex type and select **Append | Assertion** or **Insert | Assertion**. You can add multiple assertions to a complex type. To delete an assertion, select it and press the **Delete** key.

**Defining the assertion's XPath expression**

The XPath expression of a complex type assertion defines the validation constraint to be applied on the complex type element in the instance document. For example, in the screenshots above, the assertion is on the complex-type element `team` and the assertion's XPath expression is: `@region="US"`. In the XML Schema document, the assertion appears as:

```xml
<xs:assert test='@region="US"'/>
```

The assertion specifies that, in the instance document, the `team` element must have a `region` attribute with a value of `US`. If it does not, the document will be invalid.
Note the following points:

- XPath expressions must be written in the XPath 2.0 language
- Nodes tested in the XPath expression must be within the scope of the assertion (see above)
- If an expression does not evaluate to boolean true/false, the returned value is converted to a boolean value. A non-empty sequence is converted to true, while an empty sequence is converted to false.
- Syntax errors in the expression flagged by displaying the expression in red. Context errors are not flagged. For example, if the XPath expression tests an attribute and that attribute is not defined in the schema, no error is flagged.

The assertion's message

It is very useful if an explanation of the assertion is supplied together with its definition, so that in case the assertion is not fulfilled when the XML instance document is validated, an appropriate message can be displayed. Since the XML Schema specification does not make provision for such a message, XMLSpy allows a message in the Altova xml-schema-extensions namespace http://www.altova.com/xml-schema-extensions (or any other namespace) to be provided with the definition of the assertion and to be used in the validation of the XML instance document. For example:

```xml
<xs:assert test="count(//MyNode) ge 1" altova:message="There must be at least one MyNode element"/>
```

If the restriction specified in the assertion is not fulfilled, XMLSpy's validation engine will display, along with the validation-error message, the message associated with the assertion as a hint. The validator will report the value of an assert/@message attribute or of an assertion/@message attribute regardless of the namespace in which the message attribute is. However, in Schema View, you can edit only message attributes that are in the Altova xml-schema-extension namespace. To edit message attributes in other namespaces, use Text View.

See Assertion Messages for details.

Using xpathDefaultNamespace

A default namespace declared in the XML Schema document is the default namespace of the XML Schema document. It applies to unprefixed element names in the schema document but not to unprefixed element names in XPath expressions in the schema document.

The xpathDefaultNamespace attribute (a new feature in XSD 1.1) is the mechanism used to specify the namespace to which unprefixed element names in XPath expressions belong. XPath default namespaces are scoped on the XML Schema elements on which they are declared. The xpathDefaultNamespace attribute can occur on the following XML Schema 1.1 elements:

- xs:schema
- xs:assert and xs:assertion
- xs:alternative
- xs:selector and xs:field (in identity constraints)
The `xpathDefaultNamespace` on `xs:schema` is set, in XSD 1.1 mode, in the Schema Settings dialog (Schema Design | Schema Settings). For the other elements listed above, the `xpathDefaultNamespace` attribute is set in their respective Details entry helpers (see screenshot below for example).

![Screenshot of Details window showing xpathDefaultNamespace and its options](image)

Declaring the XPath default namespace on `xs:schema`, declares the XPath default namespace for the scope of the entire schema. You can override this declaration on elements where the `xpathDefaultNamespace` attribute is allowed (see list above).

Instead of containing an actual namespace, the `xpathDefaultNamespace` attribute can contain one of three keywords:

- `##targetNamespace`: The XPath default namespace will be the same as the target namespace of the schema
- `##defaultNamespace`: The XPath default namespace will be the same as the default namespace of the schema
- `##local`: There is no XPath default namespace

If no XPath default namespace is declared in the document, unprefixed elements in XPath expressions will be in no namespace.

**Note:** The XPath default namespace declaration does not apply to attributes.

For XPath expressions in assertions, you can also specify the XPath default namespace on the definition of the assertion. In the Assertions tab of the Attributes/Assertions/Identity-Constraints (AAIDC) pane (screenshot below), select the required keyword from the dropdown list of the `XPathDefaultNS` field.

![Screenshot of Assertions tab showing XPathDefaultNS and its options](image)

The selected namespace will be in scope on the assertion.
Identity Constraints
Identity constraints (IDCs) can be defined on global or local element declarations. They specify the uniqueness of nodes and enable correct referencing between unique nodes.

Declaration mechanisms
The following mechanisms are available for defining an IDC (unique, key, keyref):

- In **Schema Overview**, IDCs can be declared on global elements. Select a global element and define IDCs in the Identity Constraints tab of the Attributes/Assertions/Identity-Constraints (AAIDC) pane (screenshot below).

  ![Identity Constraints Tab](screenshot)

  Add an IDC (unique, key, keyref) using the Insert or Append icon of the Identity Constraints tab. These icons can also be used to add a field to the selected IDC. Use the Delete icon to delete the selected field or IDC.

- In the **Content Model View** of a global element, IDCs can be defined on the global element or on a local descendant element. In this view, IDCs can be edited either in the Identity Constraints tab (screenshot above) or in an element's Constraints box in the diagram (screenshot below, in which the match element has a uniqueness constraint that has a team selector). The latter alternative can be selected in the Schema Display Configuration dialog (Schema Design | Configure View). Alternatively, you can click the **Display Constraints in Diagram** icon in the Schema Design toolbar. The diagram provides a graphical representation of IDCs and drag-and-drop editing functionality.
To add an IDC (unique, key, keyref) in the diagram when diagram mode for IDCs is switched on, right-click the element to be constrained and select Add Child [ [ IDC ] ] from the context menu. The field item will be enabled in the context menu only when an IDC is selected in the diagram. Press the Delete key to delete the selected field or IDC.

The XPath expression can be entered in the selector and field boxes in one of three ways: (i) by typing it in, (ii) by selecting the required node from a dropdown list that appears automatically when you click in the selector or field box, or (iii) by dragging the target node into the selector or field box and dropping it when the box becomes highlighted; the XPath expression will be created automatically.

Note: Additionally, an overview of all identity constraints in the schema is available in the Identity Constraints tab of the Components entry helper.

Identity constraint icons

- **Display Constraints in Diagram**: Enabled in Content Model View. Toggles the display of IDCs between the diagram (toggled on) and the Identity Constraints tab.

- **Visualize Identity Constraints**: Enabled in Content Model View. Toggles the display of IDC information on and off.

- **Selector node, Field node**: Seen in node boxes in the diagram, these two icons identify, respectively, the node selected (in IDCs) by the XPath expression for selector and for field.

Visualizing IDCs

When the Visualize Identity Constraints icon is toggled on, IDC information is displayed in the diagram and can be visualized better. When visualization is toggled on, nodes selected by the selector and field XPath expressions are indicated with icons in their boxes (see icons section.
above), and the IDC box is connected to its selector and field boxes with green lines (see screenshot above).

The Visualize ID Constraints icon also switches on IDC validation functionality in Schema View. If an XPath expression is incorrect or an IDC is otherwise incorrect, errors are indicated with red text, warnings with orange text. On validating the XML Schema document, error or warning messages are displayed in the Messages window.

You can also disable validation by toggling the Visualize ID Constraints icon off.

XML listing
The IDC examples further below in this section are based on the following valid instance document.

```xml
<results xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="Scores.xsd">
  <!----- Groups ----->
  <group id="A">
    <team name="Brazil"/>
    <team name="Germany"/>
    <team name="Italy"/>
    <team name="Holland"/>
  </group>
  <group id="B">
    <team name="Argentina"/>
    <team name="France"/>
    <team name="England"/>
    <team name="Spain"/>
  </group>
  <!----- Matches ----->
  <match group="A" date="2012-06-12" location="Munich">
    <team name="Brazil" for="2" points="3"/>
    <team name="Germany" for="1" points="0"/>
  </match>
  <match group="A" date="2012-06-12" location="Frankfurt">
    <team name="Italy" for="2" points="1"/>
    <team name="Holland" for="2" points="1"/>
  </match>
  <match group="B" date="2012-06-13" location="Munich">
    <team name="Argentina" for="2" points="3"/>
    <team name="France" for="0" points="0"/>
  </match>
  <match group="B" date="2012-06-13" location="Berlin">
    <team name="England" for="0" points="1"/>
    <team name="Spain" for="0" points="1"/>
  </match>
</results>
```

Uniqueness constraints (unique)
A uniqueness constraint specifies that the value of an element or attribute (or of a set of elements and/or attributes) must be unique within a defined scope. In the XML listing shown above, we wish to ensure that the two teams playing a match are not the same team. So, within the scope of
each match element, we constrain the values of the team/@name node to be unique. We do this as follows.

1. In Schema Overview, select the match element. The match element will therefore be the scope of the identity constraint definition.
2. In the Identity Constraints tab, click the Add or Insert icon at the top left of the tab, and, in the menu that pops up, click Unique. This adds a row for the uniqueness constraint (see screenshot below).

3. Give the identity constraint a name. (In the screenshot above, MatchTeams is the name.)
4. Enter an XPath expression in the Selector column to select the team element. Note that the match element is the context node. The team element will now be the IDC’s selector, that is, the node to which the uniqueness constraint applies.
5. In the Field column, enter the @name node that must be unique. The value of this node is the value that must be unique.

The uniqueness constraint described above specifies that within the scope of each match element, every team element must have a unique @name attribute-value.

You can use additional fields to check for uniqueness. For example, a uniqueness constraint can be defined on the results element to check that all matches have a unique combination of date and location: Not more than one match may occur at one location on the same date. The uniqueness constraint must have, for each match element (the selector), its combination of @date and @location values unique within the scope of the results element.

Define the uniqueness constraint on the results element in a similar way to that described above. The selector will be match, and the fields will be @date and @location (see screenshot below). Add the second field by clicking the Append icon and then Field.

Note: The Refer column in the Identity Constraints tab is enabled for keyref constraints only, not for unique or key constraints.
Key constraints (key)

A key constraint specifies: (i) that the value of an element or attribute (or of a set of elements and/or attributes) must be unique within a defined scope, and (ii) that these field elements and/or attributes must be present in the instance XML document; therefore, optional elements or attributes should not be selected as fields of a key constraint. A key constraint is thus (in point (i) above) exactly the same as a uniqueness constraint. It stipulates one additional constraint: that its field elements/attributes must be present in the XML document.

The screenshot below shows a key constraint defined on a match element that is similar to the first uniqueness constraint described above.

```
<table>
<thead>
<tr>
<th>Attributes</th>
<th>Assertions</th>
<th>Identity constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Refer</td>
<td>Selector</td>
</tr>
<tr>
<td>key</td>
<td>UniqueTeams</td>
<td>team</td>
</tr>
</tbody>
</table>
```

This key constraint specifies that within the scope of each match element, every team element must have a unique @name attribute-value. Additionally, it specifies that the @name attribute must be present on every match/team element.

**Note:** The Refer column in the Identity Constraints tab is enabled for keyref constraints only, not for unique or key constraints.

Key references (keyref)

Key references check one set of values in an instance document against another. In our XML listing, for example (see listing above), we can use a key reference to check whether the teams playing in matches are among the teams listed in the respective groups. If not, the XML document will be invalid.

First, we create a uniqueness constant or key constraint. The screenshot below shows a uniqueness constraint (unique), TeamsInGroups, created on the results element. This constraint stipulates that each team in group has a unique @name attribute.

```
<table>
<thead>
<tr>
<th>Attributes</th>
<th>Assertions</th>
<th>Identity constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Refer</td>
<td>Selector</td>
</tr>
<tr>
<td>unique</td>
<td>TeamsInGroups</td>
<td>group/team</td>
</tr>
</tbody>
</table>
```

Next, we create the key reference (keyref), TeamCheck, which selects the team child of match
and checks whether its `@name` attribute-value is present among the values returned by `TeamsInGroups`, which it references (in the `Refer` column).

The screenshot below shows the graphical display of this key reference (highlighted in blue) together with the Details entry helper (in which you can also select the referenced IDC). The relations of the selected IDC are shown with a solid green line, while unselected IDCs are shown with a dotted green line. Also, for each identity constraint the node selected by the XPath expression for `selector` and `field` are show with the icons 🌈 and 🌈 respectively. If a node is collapsed, the relationship line to it ends with an ellipsis.
Using xpathDefaultNamespace

A default namespace declared in the XML Schema document is the default namespace of the XML Schema document. It applies to unprefixed element names in the schema document but not to unprefixed element names in XPath expressions in the schema document.

The `xpathDefaultNamespace` attribute (a new feature in XSD 1.1) is the mechanism used to specify the namespace to which unprefixed element names in XPath expressions belong. XPath default namespaces are scoped on the XML Schema elements on which they are declared. The `xpathDefaultNamespace` attribute can occur on the following XML Schema 1.1 elements:

- `xs:schema`
- `xs:assert` and `xs:assertion`
- `xs:alternative`
- `xs:selector` and `xs:field` (in identity constraints)

The `xpathDefaultNamespace` on `xs:schema` is set, in XSD 1.1 mode, in the Schema Settings dialog (Schema Design | Schema Settings). For the other elements listed above, the `xpathDefaultNamespace` attribute is set in their respective Details entry helpers (see screenshot below for example).

Declaring the XPath default namespace on `xs:schema`, declares the XPath default namespace for the scope of the entire schema. You can override this declaration on elements where the `xpathDefaultNamespace` attribute is allowed (see list above).

Instead of containing an actual namespace, the `xpathDefaultNamespace` attribute can contain one of three keywords:

- `##targetNamespace`: The XPath default namespace will be the same as the target namespace of the schema
- `##defaultNamespace`: The XPath default namespace will be the same as the default namespace of the schema
- `##local`: There is no XPath default namespace

If no XPath default namespace is declared in the document, unprefixed elements in XPath expressions will be in no namespace.

**Note:** The XPath default namespace declaration does not apply to attributes.
**IDs of identity constraints**

An ID can be assigned to an identity constraint, its selector, and/or field/s. To assign an ID, select the required component and, in the Details entry helper, enter the ID in the `id` row.

**Sorting identity constraints**

You can sort the IDCs in the Identity Constraints tab by clicking the `Sort` icon in the tab's toolbar. In the Sort Components dialog that pops up (screenshot below), you can choose to sort either the selected single component and its siblings, or the set of selected components. You can use click +Shift to select a range and click +Ctrl to add additional components to the selection.

After setting the range you can choose to sort the entire range alphabetically (**Name before kind**), or organized alphabetically by kind (that is: uniqueness constraints first, then key constraints, then key references).

The sort order is implemented in the text of the schema.

### 4.3.5 Entry Helpers in Schema View

There are three entry helpers in Schema View. They are described in detail in the sub-section of this section:

- [Components entry helper](#)
- [Details entry helper](#)
- [Facets entry helper](#)

The entry helpers are the same in both Schema Overview and Content Model View. They enable you to graphically add and edit definitions of schema components. Typically you can drag components from an entry helper, or select a component in the design and then define properties for it in an entry helper.
Components

The Components entry helper in Schema View (see screenshots below) serves three purposes:

- To organize global components in a tree view by component type and namespace (see screenshots below). This provides organized overviews of all global components and global components according to namespace.
- To enable you to navigate to and display the Content Model View of a global component—if the component has a content model. If a component does not have a content model, the component is highlighted in the Schema Overview. Global components that are included or imported from other schemas are also displayed in the Components entry helper.
- To provide an overview of the identity constraints defined in the schema document. For a description of the Identity Constraints tab, see Identity Constraints.

Note: Whether the built-in datatypes of XSD 1.0 or 1.1 are displayed depends on which XSD mode (XSD 1.0 or 1.1) is selected.

Globals tab

In theGlobals tab (see screenshot below) global components are grouped in a tree according to their component type. The number of each global component type present in the schema is given next to each component type.

In the tree display, global components are organized into the following seven groups. Note that a component type is listed in a tree only if at least one component of that type exists in the schema.

- Element Declarations (Elements)
- Model Groups (Groups)
- Complex Types
- Simple Types
- Attribute Declarations (Attributes)
- Attribute Groups
- Notations

Namespaces tab

In the Namespaces tab (see screenshot below), components are organized first according to
namespaces and then according to component type.

In the tree display, global components are organized into the following seven groups. Note that a component type is listed in a tree only if at least one component of that type exists in the schema.

- Element Declarations (Elements)
- Model Groups (Groups)
- Complex Types
- Simple Types
- Attribute Declarations (Attributes)
- Attribute Groups
- Notations

Component-type groups in the Globals and Namespaces tabs
Expanding a component-type group in the Globals tab or Namespaces tab displays all the components in that group (see screenshot below). This enables you to easily navigate to a user-defined component. When you double-click the component in the Components tab, its definition is displayed in the main window.
If a component has a content model (that is, if it is an Element, Group, or Complex Type), double-clicking it will cause the component's content model to be displayed in Content Model View (in the Main Window). If the component does not have a content model (i.e. if it is a Simple Type, Attribute, Attribute Group, or Notation), then the component is highlighted in Schema Overview (in the Main Window).

Note: If the component is in an included or imported schema, then the included/imported schema is opened (if it is not already open), and either the component's content model is displayed in Content Model View or the component is highlighted in Schema Overview.

Identity constraints
The Identity Constraints tab of the Components entry helper (screenshot below) provides an overview of a document's identity constraints. In this tab, identity constraints are listed by the kind of identity constraint (unique, key, keyref) and displayed as an expandable/collapsible tree.
Entries in bold are present in the current schema, while those in normal face are present in sub-schemas. Double-clicking an entry in the Identity Constraints tab selects that schema component in **Content Model View**.

The following context menu commands are available when an item in the Identity Constraints tab is selected:

- **Show in Diagram**: selects the schema component in **Content Model View**.
- **Show Selector/Field Target in Diagram**: selects, in **Content Model View**, the schema component targeted by the selector or field of the identity constraint. In the case of multiple fields, a dialog prompts the user for the required field.
- **Go to Identity Constraint**: selects the identity constraint in **Schema Overview**.
- **Expand/Collapse All**: expands or collapses the tree, respectively.

For a description of the Identity Constraints tab, see the section, **Identity Constraints**.

**Details**

The Details entry helper of Schema View displays editable information about the component or compositor currently selected in the Main Window. If you are editing a schema file which contains database extensions, an additional tab with information about the DB extensions may be visible.
To change the properties of the currently selected component or compositor, double-click the field to be edited and edit or enter text directly. If a combo box is available in the field to be edited, select the desired value from the dropdown list. Changes you make via the Details entry helper are immediately reflected in the design.

**Simple type derivations**

You can use the Details entry helper to quickly and accurately create derived simple types: restriction, list, and union. When a simple type is selected in the design, the Details entry helper will have a Simple Type tab in it (see screenshot below).

In the derivation-type combo box of the SimpleType tab, select the derivation type (restriction, list, or union) and, in the corresponding member type combo box to its right, select a simple type from the available simple types. Use the icons in the toolbar to append or insert a type on the same level, to add another derivation sub-level, or to delete a derivation type.
Facets

A new simple type (named or anonymous) is created by restricting the simple type's base type (which is an existing simple type). Such a restriction is effected by adding facets to restrict the values of the base type. In Schema View, the Facets entry helper (see screenshots below) enables you to graphically and easily edit the facets of a simple type. The available facets are organized in tabs of the Facets entry helper as listed in the table below.

<table>
<thead>
<tr>
<th>Tab</th>
<th>Available facets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facets</td>
<td>minInclusive, maxInclusive, minExclusive, maxExclusive, length, minLength, maxLength, totalDigits, fractionDigits, whiteSpace, explicitTimezone</td>
</tr>
<tr>
<td>Patterns</td>
<td>pattern</td>
</tr>
<tr>
<td>Enumerations</td>
<td>enumeration</td>
</tr>
<tr>
<td>Assertions</td>
<td>assertion</td>
</tr>
<tr>
<td>Samples</td>
<td>altova:exampleValues is an annotation, not a facet. This annotation is used to generate sample values in the instance XML document generated by XMLSpy from the XML Schema.</td>
</tr>
</tbody>
</table>

Each of these tabs is described in the sections below.

Selecting the simple type in the design

A simple type (named or anonymous) can be selected in the following design environments:

- In **Schema Overview** (either in the global components list or in the Attributes tab below the global components list), or
- In **Content Model View** (either in the diagram or in the Attributes tab below the diagram).

When a simple type is selected in the design in any of the design environments listed above, applicable facets in the Facets entry helper become enabled and can be edited in the Facets entry helper.

Facets tab

In the Facets tab, only facets applicable to the type selected in the design will be displayed. For example, if it is the `xs:string` type that is being restricted, then non-applicable facets like totalDigits will not be displayed.

- The four bounds facets (minInclusive, maxInclusive, minExclusive, maxExclusive) are applicable only to the numeric and date/time types and to types derived from these types.
- The three length facets (length, minLength, maxLength) are applicable only to string-based types, the binary types, and anyURI.
- The totalDigits facet apply to `xs:decimal` and integer types, and to any types derived from them. The fractionDigits facet can be applied only to `xs:decimal`. 
To enter a value, either select a value from the dropdown list of a combo box (if present) or double-click in the value field and enter a value. If an invalid value is entered, the resulting conflicts are displayed in red. Valid values are displayed in blue. For example, a minInclusive facet and a maxInclusive facet cannot exist together; so if a value is entered for the second of these facets, then the values of both facets are displayed in red.

To specify a fixed facet (giving the facet an attribute-value of fixed="true"), click the open-lock symbol to the right of the facet so that the symbol becomes a closed-lock. In the screenshot above, the totalDigits facet has been set as a fixed facet. More than one facet can be fixed. To unfix a facet, click the closed-lock symbol to make it an open-lock symbol.

Patterns tab
In the Patterns tab (screenshot below), you can add one or more pattern facets to a restriction. The pattern (of a pattern facet) is specified with the regular expression syntax. The pattern in the screenshot below specifies the pattern of email addresses.

If multiple pattern facets are specified, then the XML instance value must match one of the specified patterns. For example, a pattern to restrict postcodes could have two pattern facets, one each for US and EU postcodes. An XML instance value must then match one of the patterns for it to be valid.

Add a pattern facet by clicking the Append or Insert icon at top left and then entering a regular expression to define the required pattern. To delete a pattern, select it and click the Delete icon at top right.

Enumerations tab
In the Enumerations tab (screenshot below), you can add one or more enumeration facets to a
restriction. Each enumeration facet specifies a valid value for the type. Taken together, a set of enumeration facets specifies a range of allowed values. In the screenshot below, enumeration facets specify the allowed range of size values for the restriction.

Add an enumeration facet by clicking the Append or Insert icon at top left and then entering the enumeration value. To delete an enumeration, select it and click the Delete icon at top right.

Assertions tab
Assertions are an XSD 1.1 feature. So the Assertions tab will be enabled only in XSD 1.1 mode. Assertion facets defined in the Assertions tab of the Facets entry helper are assertions for simple types—as opposed to assertions for complex types (which can be defined and edited in Schema Overview or Content Model View, not in the Facets entry helper).

When a simple type (element or attribute of simple content) is selected in the design, an assertion can be specified for it by switching to the Assertions tab (see screenshot below), clicking the Append or Insert icon at top left, and then entering the XPath 2.0 expression that will be used to define the assertion. A special variable called $value must be used in the XPath expression to hold the value of the simple type. (Note that, since there are no descendants to test but only a value, the normal self::node() path step (or the period abbreviation of this path step '.') cannot be used in the XPath expression.)

For example, the XPath expression string-length($value) = 6 (see screenshot below) tests whether the value of the simple type has six characters. If the element or attribute in the instance document does have six characters, then it is valid according to the assertion.

Note: Syntax errors in the XPath expression will be flagged by the expression turning red. But, since the datatype is determined at runtime, type errors will not be flagged when you enter the XPath expression. You must take care to construct types as required. For an example of type construction, see the third XPath expression in the screenshot above,
which converts a string value (assuming that the assertion is defined on an `xs:string` simple type) into a number before doing a numeric comparison.

Multiple assertions can be specified on a single simple type, as in the screenshot above. In this case, all the assertions must be satisfied for the element or attribute in the instance document to be valid. The assertions in the screenshot above specify that the instance document value must be a six-character string starting with the characters `EU` and having numeric characters that have a number value of `0000` to `4999` as its final four characters. To edit the details of an assertion, right-click the assertion in the Facets entry helper, and click **Details** in the menu that pops up. This brings up the Assertion Details modal window (see screenshot below).

It is very useful if an explanation of the assertion is supplied together with its definition, so that in case the assertion is not fulfilled when the XML instance document is validated, an appropriate message can be displayed. Since the XML Schema specification does not make provision for such a message, XMLSpy allows a message in the Altova xml-schema-extensions namespace `http://www.altova.com/xml-schema-extensions` (or any other namespace) to be provided with the definition of the assertion and to be used in the validation of the XML instance document. For example:

```xml
<xs:assert test="count(//MyNode) ge 1" altova:message="There must be at least one MyNode element"/>
```

If the restriction specified in the assertion is not fulfilled, XMLSpy’s validation engine will display, along with the validation-error message, the message associated with the assertion as a hint. The validator will report the value of an `assert/@message` attribute or of an `assertion/@message` attribute regardless of the namespace in which the `message` attribute is. However, in Schema View, you can edit only `message` attributes that are in the Altova xml-schema-extension namespace. To edit `message` attributes in other namespaces, use Text View.

See **Assertion Messages** for details.

**Note:** It is a good practice recommendation to use other facets in preference to assertions where possible. For example, the restriction specified by the first assertion in the screenshot above would be better specified by the `length` facet (in the Facets tab).

**Samples tab**

In the Samples tab (screenshot below), you can specify sample values that can be used when generating an XML file from the XML Schema (with the menu command **DTD/Schema | Generate**...).
Sample XML File). If a sample value is invalid, a warning is indicated by displaying the sample value in orange. In the screenshot below, the first value is invalid because it does not match the pattern facet specified for emails (see Patterns tab above).

Note: Click the Display Validation Warnings icon in the toolbar to switch on the display of invalid sample-value warnings. An invalid sample value does not invalidate the XSD file if the file is valid in other respects.

Sample values are placed in an altova:example annotation element that is in the http://www.altova.com/xml-schema-extensions namespace. Add an altova:example annotation by clicking the Append or Insert icon at top left and then entering the altova:example value. To delete an altova:example annotation, select it and click the Delete icon at top right.

4.3.6 Validation and Smart Fixes

An XML Schema document can be validated for correctness. Do this by clicking the menu command XML | Validate XML (F8).

If the document is valid, a message to this effect is displayed in the Messages window.

If the document is invalid, the Messages window will change to display two panes (see screenshot below). The left-hand pane (the Errors pane) lists the first errors, or all errors. The right-hand pane is the Smart Fix pane; it contains a list of possible fixes for the error selected in the left-hand pane. For example, in the screenshot below, selection of the second error in the Errors pane has caused possible fixes for this error to be listed in the right-hand Smart Fix pane. If you select one of the fixes and then click either Fix+Validate or Fix, the error in the document is corrected with this particular fix.
Errors pane
The toolbar of the window provides the following functionality:

- Scroll through the errors using the **Up** and **Down** arrows.
- Copy a message, or a message and its descendants, or all messages to the clipboard.
- Search for words you want using the Find, Find Next, and Find Previous functionality. This is useful if several errors have been reported.
- Clear all errors from the Errors pane.
- Set a limit to the number of found and displayed errors (1 to 999). The default is 100. Click the button to edit the limit.
- Show/Hide Smart Fix pane. When the Smart Fix pane is hidden, the **Show Smart Fix** button appears in the toolbar; clicking it causes the Smart Fix pane to be displayed, and the button changes to **Hide Smart Fix**. If the **Show/Hide Smart Fix** button is disabled, no smart fix is available.

Smart Fix pane
The toolbar of the window provides the following functionality:

- The **Fix+Validate** button corrects the selected error with the selected Smart Fix and re-validates the document. Any other errors will be reported in the Errors pane.
- Clicking the **Fix** button fixes the error but does not re-validate.
- The **Options** button drops down a list containing a choice of behavior on double-clicking a Smart Fix: whether double-clicking carries out a **Fix+Validate** or a **Fix**.

4.3.7 Assertion Messages
In XML Schema 1.1, assertions can be defined for complex types (using `xs:assert` elements) and simple types (using `xs:assertion` elements).

It is very useful if an explanation of the assertion is supplied together with its definition, so that in case the assertion is not fulfilled when the XML instance document is validated, an appropriate message can be displayed. Since the XML Schema specification does not make provision for such a message, XMLSpy allows a message in the Altova xml-schema-extensions namespace `http://www.altova.com/xml-schema-extensions` (or any other namespace) to be provided with the definition of the assertion and to be used in the validation of the XML instance document. For example:

```xml
<xs:assert test="count(/MyNode) ge 1" altova:message="There must be at least one MyNode element"/> or
<xs:assertion test="count(/MyNode) ge 1" altova:message="There must be at least one MyNode element"/>
```

If the restriction specified in the assertion is not fulfilled, XMLSpy's validation engine will display, along with the validation-error message, the message associated with the assertion as a hint. The validator will report the value of an `assert/@message` attribute or of an `assertion/@message` attribute regardless of the namespace in which the `message` attribute is. However, in Schema View, you can edit only `message` attributes that are in the Altova xml-schema-extension namespace. To edit `message` attributes in other namespaces, use Text View.
Editing xs:assert messages

In Schema View, xs:assert elements (for complex types) can be created and edited in the Attributes/Assertions/Identity Constraints (AAIDC) pane or Details entry helper of the relevant complex type. The screenshot below shows an assertion for the complex type orderType. The assertion (an xs:assert in this case) is defined in the Assertions tab (of Schema Overview) together with an assertion message.

If the Display Assertions in Diagram option is selected, assertions on complex types can also be created and edited in Content Model View. To add or edit an assertion message, select the assertion and enter the assertion message in the Details entry helper (see screenshot below).

Note that assertion messages created in this way are in the Altova xml-schema-extensions namespace http://www.altova.com/xml-schema-extensions. When you add the first assertion message in the XML schema document via the AAIDC pane or Details entry helper, the Altova xml-schema-extensions namespace is automatically declared on the xs:schema element.

If an XML file is validated and the assertion test is not fulfilled, the message defined for the assertion is displayed together with an error message (see screenshot below).
Editing xs:assertion messages

In Schema View, xs:assertion elements (for simple types) can be created and edited in the Facets entry helper of the relevant simple type. To edit the assertion message, right-click the assertion in the Facets entry helper (see screenshot below), click Details in the menu that pops up, and edit the message in the Assertion Details modal window (see screenshot below). Note that assertion messages created in this way are in the Altova xml-schema-extensions namespace http://www.altova.com/xml-schema-extensions. When you add the first assertion message in the XML schema document via the Assertion Details modal window, the namespace is automatically declared on the xs:schema element.

If an XML file is validated and the assertion test is not fulfilled, the message defined for the assertion is displayed together in the error message (see screenshot below).
4.3.8 **Base Type Modification**

If the base type of a derived type is changed in Schema View, content, attributes, facets and sample values defined within the derived type can be handled in one of two ways:

- They can be preserved if they are still applicable in combination with the new base type.
- They can be removed automatically whether or not they are still applicable in combination with the new base type.

When changing the base type of a derived type which contains content, attributes, facets or sample values the Base Type Modification dialog (*screenshot below*) is displayed.

If the *Request Confirmation* check box is de-selected a pop-up (*screenshot below*) indicates that the confirmation can be turned on again in the View section of the Options dialog (**Tools** | **Options | View**).
In the Schema View pane (screenshot below) of the View section of the Options dialog (Tools | Options | View), you can specify whether content should be preserved and whether user confirmation is required for every base type modification.

Check the respective check boxes to preserve content and require confirmation if you wish these to be the default options.

4.3.9 Smart Restrictions

When restricting a complex type, parts of the content model of the base type are rewritten in the derived type. This can be confusing if the content model is complex because while editing the derived type it might be hard to correctly remember exactly what the content model of the base type looks like.

Smart Restrictions combine and correlate the two content models in the graphical view of the derived content model. In the derived complex type, all particles of the base complex type, and how they relate to the derived type, can be seen. Additionally, Smart Restrictions provide visual hints to show you all possible ways to restrict the base type. This makes it easy to correctly restrict the derived type.

To switch on Smart Restrictions:

- Click the Smart Restrictions icon in the Schema Design toolbar.

The example that follows illustrates the features of Smart Restrictions.

The following complex type is the base type used in this example:
The complex type "derived" is derived from the "base" type as follows:

1. Create a new complex type in the schema and call it "derived".
2. In the Details Entry Helper select "base" from the base drop-down list and "restriction" from the derivedBy drop-down list.

With Smart Restrictions switched on, the new derived type looks like this:

Notice the following controls that can be used to restrict the derived type in this example:
• Use this icon to remove elements that are in the base type from the derived type. Here, elem1 has been deleted. To add it again, click this icon.

• Click the down arrow on the Choice compositor to get the following list, which allows you to change the Choice model group to a Sequence model group:

It is also possible to change wildcards in the same way, as seen in this example:
For a complete list of which particles can be replaced by which other particles, see the XML schema specification.

- Change the number of occurrences of the model group using the following control

Here you can see that the minimum occurrence for this element has been changed to 2. Notice that the model group now has a blue background, which means that it is no longer the same as the model group in the base complex type. Also, the permitted occurrence range of the model group in the base particle is now displayed in parentheses.

- It is possible to change the data types of attributes or elements if the new data type is a valid restriction of the base data type as defined in the XML schema specification. For example, you can change the data type of elem3 in the "derived" data type from decimal to integer. After you do this, the element has a blue background to show that is different from the element in the base type, and the type that the element has in the base type is
This example shows attributes whose data types have been restricted in the derived complex type:

- Smart Restrictions alert you to *pointless occurrences* in the content model. A pointless occurrence happens, for example, when a sequence that is present in the content model is unnecessary. This example shows a pointless occurrence:
Please note: Pointless occurrences are only shown if the content model contains an error. It is possible for a content model to contain a pointless occurrence and be valid, in which case the pointless occurrence is not explicitly shown in order to avoid confusion.

See the XML schema specification for more information about pointless occurrences.

4.3.10 **xml:base, xml:id, xml:lang, xml:space**

The namespace [http://www.w3.org/XML/1998/namespace](http://www.w3.org/XML/1998/namespace) is, according to the XML Namespaces specification, bound by definition to the xml: prefix. What this means is that this is the namespace that must be used with the xml: prefix and that is reserved for it. There are four attributes in this namespace that can be children of any XML element in any XML document (schema or instance):

- **xml:base** (for setting the base URI of an element)
- **xml:id** (for specifying the unique ID of an element)
- **xml:lang** (for identifying the language used within that element)
- **xml:space** (for specifying how whitespace in the element should be handled)

In Schema View, once the XML Namespaces namespace has been imported into the XML Schema document, these four xml: attributes can be referenced for use on any element in the schema.

In order to declare one of these attributes on an element, do the following:

1. Declare the XML Namespaces namespace for that schema document and bind the namespace to the xml: prefix. When any of the four xml: attributes is used in the document, its name would then be expanded to include the correct namespace part.
2. Import the XML Namespaces namespace. XMLSpy's validator will recognize the namespace and make the four xml: attributes available as global attributes, which can be referenced within that schema.
3. Insert the required xml: attribute as the child of an element. The attribute is declared as a reference to the "imported" global attribute.

Declare the XML Namespaces namespace

You can declare the XML Namespaces namespace ([http://www.w3.org/XML/1998/namespace](http://www.w3.org/XML/1998/namespace)) by entering it via the Schema Settings dialog, where all namespaces declared for that schema...
are stored and can be edited. The namespace must be bound to the xml: prefix. (Alternatively, you could declare the namespace (with the xml: prefix) on the xs:schema element in Text View.)

**Import the XML Namespaces namespace**

In Schema Overview, create a global import declaration for the XML Namespaces namespace. Do this by clicking the Insert or Append icon at the top of the Schema Overview window and selecting Import from the menu that pops up. Enter the XML Namespaces namespace as the namespace to be imported. In Text View, the import declaration should look like this:

```
schemaLocation="http://www.w3.org/XML/1998/namespace"/>
```

**Adding the xml: attribute**

In Schema Overview, select the element for which the xml: attribute is to be added, and add an attribute for it. In the Details entry helper (screenshot below), click the down arrow of the name combo box and select the required xml: attribute, for example xml:base. When you are prompted whether you wish to reference the global attribute, click Yes. The attribute is added as a reference.

**XInclude and xml:base**

When XInclude's include element is replaced by the XML file specified in the href attribute of the include element, the top-level element of the parsed XML document is included with an xml:base attribute. If this XML document is going to be validated, then the schema must define an xml:base attribute on the relevant element's.

### 4.3.11 Back and Forward: Moving through Positions

The Back and Forward commands in Schema View enable you to move through previously viewed positions in Schema View. This is useful because, while clicking through schema components in Schema View, you might wish to view a previously viewed component. Clicking the Back button once in the toolbar takes you to the previously viewed position. By repeatedly clicking the Back button, you can view up to 500 of the last visited positions. After moving back through previous positions, you can move forward through these positions by using the Forward button in the toolbar.
The shortcut keys for the two commands are:

- ▶️ Back: Alt + Left Arrow
- ▶️ Forward: Alt + Right Arrow

**Back/Forward versus Undo/Redo**

Note that the **Back** and **Forward** commands are not the same as the **Undo (Ctrl+Z)** and **Redo (Ctrl+Y)** commands. These two sets of commands make up two different series of steps. Clicking the **Back** command once takes you to the previously viewed component as previously displayed. Clicking the **Undo** command once undoes the last editing change regardless of when that editing change was made.

**Additional notes**

Note the following points:

- The **Back** button enables you to re-view the previous 500 positions.
- The Back/Forward feature is enabled across schemas. If a schema has since been closed or is currently open in another view, it will be opened in Schema View or switched to Schema View, respectively.
- If a component that was viewed in a previous position is deleted, then that component will not be able to be viewed. If such a component was part of a previous position, this position will be displayed without the deleted component. If the component comprised the entire position, the entire position will be unavailable, and clicking the **Back** button at this point in the Back series will take you to the position previous to the unavailable position.
4.4 Authentic View

Authentic View has a menu bar and toolbar running across the top of the window, and three areas that cover the rest of the interface: the Project Window, Main Window, and Entry Helpers Window. These areas are shown below.

The Authentic View interface is described in detail in the section, Authentic View Interface.
4.5 Browser View

Browser View is typically used to view:

- XML files that have an associated XSLT file. When you switch to Browser View, the XML file is transformed on the fly using the associated XSLT stylesheet and the result is displayed directly in Browser View.
- HTML files which are either created directly as HTML or created via an XSLT transformation of an XML file.

To view XML and HTML files in Browser View, click the Browser tab.

Note about Microsoft Internet Explorer and XSLT

Browser View requires Microsoft's Internet Explorer 5.0 or later. If you wish to use Browser View for viewing XML files transformed by an XSLT stylesheet, we strongly recommend Internet Explorer 6.0 or later, which uses MSXML 3.0, an XML parser that fully supports the XSLT 1.0 standard. You might also wish to install MSXML 4.0. Please see our Download Center for more details. (Note that support for XSLT in IE 5 is not 100% compatible with the official XSLT Recommendation. So if you encounter problems in Browser View with IE 5, you should upgrade to IE 6 or later.) You should also check the support for XSLT of your version of Internet Explorer.

Browser View features

The following features are available in Browser View. They can be accessed via the Browser menu, File menu, and Edit menu.

- **Open in separate window**: When Browser View is a separate window, it can be positioned side-by-side with an editing view of the same document. This command is in the Browser menu and is a toggle-command that can be used to return a separate Browser View window as a tabbed view. In the View tab of the Options dialog, you can set whether Browser View should, by default, be a separate window.
- **Forward and Back**: The common browser commands to navigate through pages that were loaded in Browser View. These commands are in the Browser menu.
- **Font size**: Can be adjusted via the Browser menu command.
- **Stop, Refresh, Print**: More standard browser commands, these can be found in the Browser and File menus.
- **Find**: Enables searches for text strings, this command is in the Edit menu.
4.6 Archive View

An Office Open XML (OOXML) file, ZIP file (for example, WinZip or WinRAR), or EPUB file can be opened and edited in Archive View. Not only can OOXML, ZIP, and EPUB archives be structurally modified in Archive View, but individual files in the archive can be opened from Archive View, edited in one of XMLSpy's editing views, and then saved directly back to the archive.

Archive files and Archive View

When an archive file (OOXML, ZIP, or EPUB file) is created or opened in XMLSpy, it is opened in Archive View (screenshot below). Multiple archive files can be open at a time, with each archive file being in a separate Archive View window. The type of the archive file appears in the top right-hand corner of Archive View. In the screenshot below, the type of the archive file is MS Office Word Open XML.

Folder View

The Folder View is located on the left-hand side of the Archive View window and displays the folder structure of the zipped archive. On each level, folders are listed alphabetically. To view the sub-folders of a folder, click the plus symbol to the left of the folder. If a folder does not have a plus symbol to the left of it, then it has no sub-folder. To view the document files (hereafter called documents) contained in a folder, select the folder; the files will be displayed in the Main Window. In the screenshot above, the documents displayed in the Main Window are in the word folder, which also has two sub-folders: _rels and theme.

Main Window

The Main Window lists the documents in the folder that is selected in Folder View. Documents are displayed in alphabetical order, each with its respective uncompressed size and the date and time of last modification. To open a Document from Archive View, double-click it. The document opens in a separate XMLSpy window.

Command buttons

The command buttons are located along the top of the Archive View window.

- **Open document**: Enabled when a document in the Main Window is selected. Clicking it opens the selected document. A document can also be opened by double-clicking the
document listing in the Main Window.

- **New folder:** Adds a new folder to the folder that is currently selected in Folder View. The folder must be named immediately upon its being created in Folder View. It is not possible to rename a folder subsequently. The new folder is saved in the archive when the archive file is saved.

- **Add new document:** Adds a new document to the folder currently selected in Folder View. Clicking this button opens the Create New Document dialog of XMLSpy. The newly created document opens in a separate XMLSpy window. The document must be named immediately upon its being listed in the document listing of the selected folder. The document is saved in the archive only when it is saved in its own editing window or when the archive file is saved.

- **Add document:** Opens a Browse dialog in which you can browse for a document to add. The document is added to the listing in the Main Window of documents currently in the selected folder, and the document is opened in a separate XMLSpy window. For the document to be saved to the archive, it must either be saved in its own window, or the archive file must be saved.

- **Delete from archive:** Deletes the selected document (in Main Window) or selected folder (in Folder View) from the archive. The archive file must be saved in order for the deletion to take effect.

- **Info:** Toggles the Info Window on and off. See below.

**Info Window**

The Info Window is toggled on and off by clicking the Info command button. The Info Window provides general information about the archive file, such as the number of files it contains, its uncompressed and compressed sizes, and the compression ratio.
5 XML

This section describes how to work with XML documents in XMLSpy. It covers the following aspects:

- **How to create, open, and save XML documents.** In this section, some important XMLSpy settings relating to file creation are also explained.
- XML documents can be edited in **Text View**, **Grid View**, and **Authentic View**. You can select the view that is most useful for you and switch among the views while editing. Each of the views offers different advantages.
- **Entry helpers** for XML documents have certain specific features, and these are described.
- How to **process XML documents with XSLT and XQuery**. Various XMLSpy features related to processing are explained. A section on **PDF Fonts** explains how fonts are processed when generating PDF output.
- Miscellaneous **other features** for working with XML documents are described.

Altova website: XML Editor
5.1 Creating, Opening, and Saving XML Documents

When creating, opening, or saving XML documents, the following issues are involved:

- In what view will the XML document open: Text View, Grid View, or Authentic View?
- When a new XML document is created, whether a schema (XML Schema or DTD) will be automatically assigned, manually assigned, or not assigned.
- If a schema is assigned to the XML document, whether the document will be validated automatically on opening and/or saving.

**Default view**

There are application-wide settings for specifying in what view XML documents (new and existing) should open. These settings are in the Options dialog (Tools | Options).

In the *File Types* section of the Options dialog, select a file type of *xml* and, in the Default View pane, check the required editing view (Text or Grid). Note that: (i) Schema View can be used only for XML Schema; and (ii) Browser View is a display view, not an editing view.

In the *File Types* tab, you can also set XMLSpy as the default editor for the selected file type.

An XML document can be edited in Authentic View if a StyleVision Power Stylesheet (SPS) has been assigned to it. When an XML file with an associated SPS is opened, you can specify that it opens directly in Authentic View. Do this by checking the *Always open in Authentic View* option in the *View* section of the Options dialog. If this option is not checked, the file will open in the default view specified for *xml* files in the *File Types* tab (see above).

**Assigning schemas**

When a new XML file is to be created, select the menu command File | New. This pops up the Create New Document dialog (screenshot below).

![Create new document dialog](image-url)
Notice that there are several options for the XML document type. The option marked *Extensible Markup Language* creates a generic XML document. Each of the other options is associated with a schema, for example the DocBook DTD. If you select one of these options, an XML document is created that has (i) the corresponding schema automatically assigned to it, and (ii) a skeleton document structure that is valid according to the assigned schema. Note that you can create your own skeleton XML document. If you save it in the Template folder of the application folder, your skeleton document will be available for selection in the Create New Document dialog.

If you select the generic Extensible Markup Language document type, you will be prompted for a schema (DTD or XML Schema) to assign to the document. At this point, you can choose to browse for a schema or go ahead and create an XML document with no schema assigned to it.

You can, of course, assign a schema via the DTD/Schema menu at any subsequent time during editing.

**Automatic validation**

If an existing XML document has a schema assigned to it, then it can be automatically validated on opening and/or saving. The setting for this is in the *File* section of the Options dialog (Tools | Options).

The automatic validation settings in the *File* tab can be combined with a setting in the File Types tab to disable automatic validation for specific file types. Using the settings in the two tabs together enables you to specify automatic validation for specific file types.
5.2 Assigning Schemas and Validating

A schema (DTD or XML Schema) can be assigned to an XML document when it is first created. A schema can also be assigned, or changed, at any subsequent time using the Assign DTD or Assign Schema commands in the DTD/Schema menu.

Global resources for schemas

A global resource is an alias for a file or folder. The target file or folder can be changed within the GUI by changing the active configuration of the global resource (via the menu command Tools | Active Configuration). Global resources therefore enable the assigned schema to be switched among multiple schemas, which can be useful for testing. How to use global resources is described in the section Altova Global Resources.

XML Schema plus DTD

One very useful DTD feature that XML Schema does not have is the use of entities. However, if you wish to use entities in your XML-Schema-validated XML document, you can add a DOCTYPE declaration to the XML document and include your entity declarations in it.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE OrgChart [
  <!ENTITY name-int "value">
  <!ENTITY name-ext SYSTEM "extfile.xml">
]>
<OrgChart xmlns="http://www.xs.com/org"
  xsi:schemaLocation="http://www.xs.com/org OrgChart.xsd">
  ...
</OrgChart>
```

After declaring the entities in the DTD, they can be used in the XML document. The document will be well-formed and valid. Note, however, that external parsed entities are not supported in Authentic View..

Going to schema definitions

With the XML document open, you can directly open the DTD or XML Schema on which it is based by clicking the Go to DTD or Go to Schema commands in the DTD/Schema menu.
Additionally, you can place the cursor within a node in the XML document and go to the schema definition of that node via the **Go to Definition** command in the **DTD/Schema** menu.

**Validating and checking well-formedness**
To validate and/or check for well-formedness, use the **Validate XML (F8)** and **Check Well-Formedness (F7)** commands in the **XML** menu or the corresponding commands in the toolbar. Any error is reported in the Messages window. If an XML document is invalid, the XML validator provides **smart fixes to correct the error** based on the information in the schema.

You can also use a **RaptorXML Server** to validate XML documents.
5.3 Editing XML in Text View

XMLSpy offers some specialized XML text editing features in addition to the generally available editing features in Text View, which are described in Text View in the Editing Views section.

- Commenting text in and out
- Escaping and unescaping XML characters
- Inserting file paths
- Inserting XML fragments via XInclude
- Copying XPath and XPointer expressions to the clipboard

Commenting text in/out

Text in an XML document can be commented out using the XML start-comment and end-comment delimiters, respectively <!-- and -->. In XMLSpy, these comment delimiters can be easily inserted using the Edit | Comment In/Out menu command.

To comment out a block of text, select the text to be commented out and then select the command Comment In/Out, either from the Edit menu or the context menu that you get on right-clicking the selected text. The commented text will be grayed out (see screenshot below).

```xml
<Department>
    <Name>Administration</Name>
    <Person>
        <Person>
            <!--<First>
                <Last></Last>
                <PhoneExt></PhoneExt>
                <EMail></EMail>
                <LeaveTotal></LeaveTotal>
                <LeaveUsed></LeaveUsed>
                <LeaveLeft></LeaveLeft>
            </First-->
        </Person>
    </Person>
</Department>
```

To uncomment a commented block of text, select the commented block excluding the comment delimiters, and select the command Comment In/Out, either from the Edit menu or the context menu that you get on right-clicking the selected text. The comment delimiters will be removed and the text will no longer be grayed out.

Note about empty lines

In XML documents, empty lines are discarded when you change views or save the document. If you wish to retain empty lines, enclose them in comment delimiters.

Escaping and unescaping XML characters

The five XML special characters (listed below) can be escaped and unescaped with the corresponding entity references (listed below) by highlighting a block of text and selecting the context menu command Escape XML Characters or Unescape XML Characters. The XML special characters in that block of text will then be escaped or unescaped according to the command selected.

```
<  &lt;
```
For example:

\[<a></a>\] can be escaped with the \textbf{Escape XML Characters} command to \&lt;a\&gt;&lt;/a&gt; and
\&lt;a\&gt;&lt;/a\&gt; can be unescaped with the \textbf{Unescape XML Characters} command to \(<a></a>\)

**Inserting file paths**

The \textbf{Edit | Insert File Path} command enables you to browse for the file in question and insert its file path at the selected location in the XML document being edited. This command enables you to quickly and accurately enter a file path. See the \textit{command description} for more details.

**Inserting XML fragments via XInclude**

The \textbf{Edit | Insert XInclude} enables you, via XInclude, to insert the contents of an entire XML document, or a fragment of one, in the XML document being edited. This command enables you to quickly and accurately enter entire XML documents (via the XInclude mechanism) or fragments of XML documents (via an XPointer extension of the XInclude mechanism). See the \textit{command description} for more details.

**Copying XPath and XPointer expressions to the clipboard**

The XPath and XPointer expressions of the selected node (expressing the node's position in the XML document) can be copied to the clipboard using the \textbf{Edit | Copy XPath} and \textbf{Edit | Copy XPointer} commands, respectively. This enables you to obtain the correct XPath and XPointer expressions targeting the selected node.

For example, let the selected node in Text View or Grid View be the third Office element of a document element called Offices. In this case, the copied XPath expression will be \texttt{/Offices/Office[3]}. And the copied XPointer expression, if the Office elements have no other-named sibling that occurs before the third Office element, will be \texttt{element(/1/3)}.

The copied expressions can then be inserted at any required location. For example, an XPath expression can be inserted in an XSLT stylesheet and an XPointer expression in the \texttt{href} attribute of an \texttt{xinclude} element.

For more detailed descriptions of the commands, see their descriptions in the User Reference section.
5.4 Editing XML in Grid View

Grid View shows the hierarchical structure of XML documents through a set of nested containers that can be expanded and collapsed. This provides a clear picture of the document's structure. In Grid View, both structure and contents can be easily manipulated.

Display as table
If there are several instances of a repeating element, then, in standard Grid View, each complete instance is displayed, one after the other, progressing vertically downward in document order (screenshot below).
Such a structure of multiple instances can also be displayed as a table (screenshot below), in which the child nodes form the columns and the multiple instances form the rows.

Table View offers a unique editing advantage in that whole rows and columns can be manipulated relative to other columns and rows in the table. This enables such operations as sorting data with respect to the value of one common child node. For example, in the screenshot above, the six Person elements can be sorted on the basis of their Last child elements via a simple GUI operation. Such an operation is much simpler than running an XSLT transformation, which would be the usual way to sort an XML nodeset.

**Editing the document structure**

In Grid View, the XML document structure can be edited graphically. For example, you can collapse and expand individual segments of the document structure, insert, append and delete nodes, drag-and-drop nodes to different locations, and convert one type of node to another type.

The XML menu offers commands to insert, append, and add empty child nodes. For example, you can add an empty child node by selecting an element and then adding an empty child.
element. You can then enter a name and content for the newly added node by double-clicking in the respective field (name or content field) and entering the required string.

The **Elements and Attributes entry helpers** enable you to insert, append, and add child nodes that are allowed at the selected location. For example, you select a node in the Main Window. The element and attribute nodes that may be validly inserted, appended, or added as a child at this location are listed in the Elements and Attributes entry helpers.

The commands available in the XML menu and entry helpers that are applicable to a selected node are also available in the context menu of that node.

**Editing content**
To edit content, double-click in the content field and type in the content text. Entities can be inserted via the Entities entry helper.

**More about Grid View**
For a more detailed description of Grid View, see under [Editing Views](#).
5.5 Editing XML in Authentic View

Authentic View enables a user to edit an XML document as if it were a text document (screenshot below). The XML markup and all other non-content text can be hidden from the person editing the document. This can be useful for people who are unfamiliar with XML, enabling them to a valid XML document even while concentrating entirely on the content of the document.

The Authentic View of a document is enabled when a StyleVision Power Stylesheet (SPS) is assigned to an XML document. An SPS is based on the same schema source as that on which the XML document is based, and it defines the structure of the XML document. The SPS also defines the layout and formatting of the document in Authentic View. For example, in the document shown in the screenshot above, the following Authentic formatting and editing features are used:

- Paragraph and other block formatting
- Table structures
- Text formatting, such as color and font face
- Combo boxes (see the State and Zip fields) enable the user to select from a group of valid choices, which can be taken from schema enumerations, as has been done in the case above
- Additional information can be calculated from the data in the document and be presented (in the example above, the office summary details have not been entered by the user but calculated from other data in the document)
SPSs are created specifically for viewing and editing XML documents in Authentic View and for generating standard output (such as HTML, PDF, RTF, and Word 2007 documents) from XML. SPSs are created with Altova StyleVision.

Editing document structure
Valid nodes can be added to the document at any time by selecting a location and then adding the required node via the entry helpers (Elements and Attributes) or context menu. The nodes available at any given location are restricted to the nodes that can be validly added as siblings or children at the selected location. For example, when the cursor is located within a paragraph, you can append another paragraph if this is allowed by the schema.

When editing the structure of an XML document in Authentic View, it could be useful to see the markup of the document. Markup can therefore be switched on as tags (screenshot below) using the Authentic | Show Large Markup command (or the corresponding toolbar icon).

The company was established in Vereno in 1995 as a privately held software company. Since 1996, Nanonull has been actively involved in developing nanoelectronic software technologies. It released the first version of its acclaimed NanoSoft Development Suite in February 1999. Also in 1999, Nanonull increased its capital base with investment from a consortium of private investment firms. The company has been expanding rapidly ever since.

Editing content
Content is created and edited by typing it into the nodes of the document. Entities and CDATA sections can be added via the context menu (entities also via the Entities entry helper).

More about editing in Authentic View
For more details of how to edit in Authentic View, see the Authentic View section.
5.6 Entry Helpers for XML Documents

For XML documents, there are three entry helpers: Elements entry helper; Attributes entry helper; and Entities entry helper. When an element is added via the Elements entry helper, it can be added together with mandatory child elements, mandatory attributes, all child elements, or no child element or attribute, according to the respective settings in the Editing section of the Options dialog. When empty attributes are added, they are added with quotes.

Note that in the different views, the entry helpers are designed differently, in accordance with the functionality of the respective view.

Elements entry helper
The following points should be noted:

- **Text View:** Elements are inserted at the cursor insertion point. Unused elements are displayed in red, used elements in gray. Mandatory elements are indicated with an exclamation mark "!' before the name of the element.
- **Grid View:** Elements can be appended after, inserted before, or added as a child of the selected element. There are therefore three tabs, each displaying the elements that may be added. Unused elements are displayed in black, used elements in gray.
- **Authentic View:** Elements can be inserted before, after, or within the selected element. Additionally, there is a document tree that shows the location of the currently selected element in the document's tree structure. For more details of how to edit in Authentic View, see the Authentic View section.

Attributes entry helper
The following points should be noted:

- **Text View:** When the cursor is placed inside the start tag of an element and after a space, the attributes declared for that element become visible. Unused attributes are displayed in red, used attributes in gray. Mandatory attributes are indicated with an exclamation mark "!' before the name of the attribute.

  To insert an attribute, double-click the required attribute. The attribute is inserted at the cursor point together with an equals-to sign and quotes to delimit the attribute value. The cursor is placed between the quotes, so you can start typing in the attribute value directly.

- **Grid View:** When an element is selected, the attributes that can be added as a child are listed in the Add Child tab of the entry helper. When an attribute is selected, the available attributes are listed in the Append (after) and Insert (before) tabs. Unused attributes are displayed in black, used attributes in gray.

- **Authentic View:** When an element is selected, the attributes declared for that element become visible. Enter the value of the attribute in the entry helper.
Entities entry helper

Any parsed or unparsed entity that is declared inline (within the XML document) or in an external DTD, is displayed in the Entities entry helper. In all three views (Text, Grid, and Authentic), an entity is inserted at the cursor insertion point by double-clicking it. In Grid View, entities are displayed in the Append, Insert, and Add Child tabs.

Note that if you add an internal entity, you will need to save and reopen your document before the entity appears in the Entities entry helper.
5.7 Processing with XSLT and XQuery

XML documents can be processed with XSLT or XQuery documents to produce output documents. XMLSpy has built-in XSLT 1.0, XSLT 2.0, XSLT 3.0, XQuery 1.0, and XQuery 3.0 processors. The following processing-related features are available in the GUI:

- Assigning XSLT stylesheets
- Go to XSLT
- XSLT parameters and XQuery variables
- XSLT transformations
- XQuery executions
- Automating XML tasks with RaptorXML

Assigning XSLT stylesheets
You can assign an XSLT stylesheet to an XML document via the XSL/XQuery | Assign XSL command (browse for the file in the dialog (screenshot below) that pops up). The assignment is entered in the XML document as a processing instruction (PI) having the standard XSLT target defined by the W3C: xml-stylesheet. This assignment is used when an XSLT transformation is invoked (XSL/XQuery | XSL Transformation).

Additionally, an XSLT-for-FO stylesheet can be assigned with the XSL/XQuery | Assign XSL:FO command (browse for the file in the dialog (screenshot below) that pops up). The assignment is entered in the XML document as a processing instruction (PI) having the Altova-defined target: altova_xslfo. This assignment is used when an XSLT-for-FO transformation is invoked (XSL/XQuery | XS:FO Transformation).

You can also select a global resource to specify the XSLT file. A global resource is an alias for a file or folder. The target file or folder can be changed within the GUI by changing the active configuration of the global resource (via the menu command Tools | Active Configuration). Global resources therefore enable the assigned XSLT file to be switched from one to another, which can be useful for testing. How to use global resources is described in the section Altova Global Resources.

If a previous assignment using either of these PI targets exists, then you are asked whether you wish to overwrite the existing assignment.

Go to XSLT
The XSL/XQuery | Go to XSL command opens the XSLT file that has been assigned to the XML document.
XSLT parameters and XQuery variables

XSLT parameters and XQuery variables can be defined, edited, and deleted in the dialog that appears on clicking the command XSL/XQuery | XSLT Parameters / XQuery Variables. The parameter/variable values defined here are used for all XSLT transformations and XQuery executions in XMLSpy. However, these values will not be passed to external engines such as MSXML. For the details of how to use this feature, see the User Reference section.

XSLT transformations

Two types of XSLT transformation are available:

- **Standard XSLT transformation** (XSL/XQuery | XSL Transformation): The output of the transformation is displayed in a new window or, if specified in the stylesheet, is saved to a file location. The engine used for the transformation is specified in the XSL tab of the Options dialog (Tools | Options).

- **XSL-for-FO transformation** (XSL/XQuery | XSL-FO Transformation): The XML document is transformed to PDF in a two-step process. In the first step, the XML document is transformed to an FO document using the XSLT processor specified in the XSL tab of the Options dialog (Tools | Options); note that you can also select (at the bottom of the tab) the XSLT engine that comes with some FO processors such as FOP. In the second step, the FO document is processed by the FO processor specified in the XSL tab of the Options dialog (Tools | Options) to produce PDF output.

**Note:** An FO document (which is a particular type of XML document) can be transformed to PDF by clicking the XSL:FO transformation command. When the source document is an FO document, the second step of the two-step process for this command is executed directly.

XQuery executions

An XQuery document can be executed on the active XML document by clicking the command XSL/XQuery | XQuery Execution. You are prompted for the XQuery file, and the result document is displayed in a new window in the GUI.

Automating XML tasks with RaptorXML

Altova RaptorXML is an application that provides XML validation, XSLT transformations, and XQuery executions. It can be used from the command line, via a COM interface, in Java programs, and in .NET applications. Tasks such as XSLT transformation can therefore be automated with the use of RaptorXML. For example, you can create a batch file that calls RaptorXML to transform a set of documents. See the RaptorXML documentation for details.
5.8 PDF Fonts

How the formatter and PDF Viewer use fonts

The formatter (for example, FOP) creates the PDF and the PDF Viewer (typically Adobe's Adobe Reader) reads it.

In order to lay out the PDF, the formatter needs to know details about the fonts used in the document, particularly the widths of all the glyphs used. It needs this information to calculate line lengths, hyphenation, justification, etc. This information is known as the metrics of the font, and it is stored with each font. Some formatters can read the metrics directly from the system's font folder. Others (such as FOP) need the metrics in a special format it can understand. When the metrics of a font are available to the formatter, the formatter can successfully lay out the PDF. You must ensure that the font metrics files of all the fonts you use in your document are available to the formatter you are using.

The formatter can either reference a font or embed it in the PDF file. If the font is referenced, then the PDF Viewer (for example, Adobe Reader) typically will look for that font in its own font resource folder (which contains the Base 14 fonts) first, and then in the system's font folder. If the font is available, it will be used when the PDF is displayed. Otherwise the Viewer will use an alternative from its resource folder or generate an error. An alternative font may have different metrics and could therefore generate display errors.

If the formatter embeds a font in the PDF file, then the PDF Viewer uses the embedded font. The formatter may embed the entire character set of a font or only a subset that contains the glyphs used in the document. This factor affects the size of the PDF file and, possibly, copyright issues surrounding font use (see note below). You might be able to influence the choice between these two options when you set the options for your formatter.

XMLSpy and PDF fonts

In XMLSpy, a PDF is generated from an XSL-FO document (from now on FO document) by processing the XSL-FO document with an external FO processor such as FOP. (In the Options dialog, you can specify the location of the FO processor. This allows the FO processing to be started from within the XMLSpy GUI.)

The XSL-FO document itself is generated by processing an XML document with an XSLT stylesheet. (You can use either Altova's XSLT engine (which is built into XMLSpy) or an external XSLT engine to do this.)

The formatting for the PDF document, including the font properties of all text, is specified in the XSL-FO document. If the formatter you are using can read the metrics of the required font directly from the font, then all you need to do is to set up the formatter to access the font. If, however, you are using FOP as your formatter, you will need to provide it with the correct font metrics files for fonts other than the Base-14 fonts.

Making fonts available to the formatter

Most formatters (including FOP) already have available to them the Base 14 fonts. It is important to know the names by which the formatter recognizes these fonts so that you correctly indicate them to the formatter. This is the basic font support provided by formatters. You can, however, increase the number of fonts available to the formatter by carrying out a few straightforward steps specific to the formatter you are using. The steps for FOP are given below.
General procedure for setting up additional font support in FOP

To make additional fonts available to FOP, you would need to do the following:

1. Generate a font metrics file for the required font from the PostScript or TrueType font files. FOP provides PFM Reader and TTF Reader utilities to convert PostScript and TrueType fonts, respectively, to XML font metrics file. For details of how to do this, see the FOP: Fonts page.

2. Set up the FOP configuration file to use the required font metrics files. You do this by entering information about the font files in an FOP configuration file. See FOP: Fonts.

3. In the file fop.bat, change the last line:

   "%JAVACMD%" [...] org.apache.fop.cli.Main %FOP_CMD_LINE_ARGS%

   to include the location of the configuration file:

   "%JAVACMD%" [...] org.apache.fop.cli.Main %FOP_CMD_LINE_ARGS% -c conf \fop.xconf

   After the metrics files are registered with FOP (in a FOP configuration file) and the FOP executable is set to read the configuration file, the additional fonts are available for PDF creation.

Setting up the FOP configuration file

The FOP configuration file is called fop.xconf and is located in the conf folder in the FOP installation folder. This file, which is an XML document, must be edited so that FOP reads the font metrics files correctly. For each font that you wish to have FOP render, add a font element at the location indicated by the font-element placeholder in the document:

```xml
<font metrics-url="arial.xml" kerning="yes" embed-url="arial.ttf">
  <font-triplet name="Arial" style="normal" weight="normal"/>
  <font-triplet name="ArialMT" style="normal" weight="normal"/>
</font>
```

In the example above,

- arial.xml is the URL of the metrics file; it is best to use an absolute path.
- arial.ttf is the name of the TTF file (usually located in %WINDIR%\Fonts).
- Arial specifies that the above metrics and TTF files will be used if the font-family is defined as Arial.
- style="normal" specifies that the above metrics and TTF files will be used if the font-style is defined as normal (not, say, italic).
- weight="normal" specifies that the above metrics and TTF files will be used if the font-weight is defined as normal (not, say, bold).

**Note on font copyrights:** Font usage is subject to copyright laws, and the conditions for use vary. Before embedding a font—especially if you are embedding the entire font—make sure that you are allowed to do so under the license you have purchased for that font.
Character sets

Note that the character sets of fonts differ from each other. The Base 14 fonts cover the ISO-8859-1 characters plus the glyphs in the Symbol and Zapf Dingbats fonts. If your document contains a character that is not covered by the Base 14 fonts, then you will have to use a font that contains this character in its character set. Some fonts, such as Arial Unicode, offer the characters covered by Unicode.
5.9 Additional Features

Additional features for working with XML files are listed below.

- Encoding
- Generating DTDs and XML Schemas
- Find and Replace
- Evaluating XPath
- Importing and exporting text

Encoding

The encoding of XML files (and other types of documents) can be set via the menu command `File | Encoding`. The default encoding of XML and non-XML files can be specified in the `Options | Encoding` section.

Generating DTDs and XML Schemas

If you wish to create a schema that describes the structure of an XML document, use the `DTD/Schema | Generate DTD/Schema` menu command. In the Generate DTD/Schema dialog that appears, you can select whether to generate a DTD or an XML Schema as well as certain XML Schema options, such as whether to generate enumerations from the values contained in the XML document.

Find and Replace

The `Find` and `Replace` features (accessed via the `Edit` menu) has powerful search capabilities. The search term can be defined additionally in terms of casing and whether whole words should be matched, and it can also be expressed as a regular expression. The search range can be restricted to a selection in the document and to particular node types (see screenshot below).

The screenshot above shows the Find dialog of Grid View. The dialog and functionality vary according to the view that is active. For the Find and Replace functionality of Text View, see here.
Evaluate XPath
An XPath expression, which you enter in the XPath/XQuery Window, can be evaluated against the active XML document. The results of the evaluation are displayed in the XPath/XQuery Window, and clicking a node in the result highlights that node in the document display in the Main Window. Note that the XPath/XQuery Window can be made active by clicking XML | Evaluate XPath command.

Importing and exporting text
Text data can be imported from, and exported to, other application formats. Commands for these features are in the Convert menu.
6 DTDs and XML Schemas

Altova website: XML Schema Editor

This section provides an overview of how to work with DTDs and XML Schemas. It also describes SchemaAgent and the powerful Find in Schemas feature. In addition to the editing features, XMLSpy provides the following powerful DTD/Schema features:

Catalog mechanism
Support for the OASIS catalog mechanism enables the re-direction of URIs to local addresses, thus facilitating use across multiple workstations.

Schema subsets
Components of a large schema can, in Schema View, be created as a separate file. These smaller schema subsets can then be included in the larger schema. The reverse operation, known as flattening a schema, puts the components of included files directly in the larger schema. How to generate schema subsets and flatten schemas is described in the section, Schema Subsets.

Converting DTDs to XML Schemas and vice versa
A DTD can be converted to an XML Schema and vice versa, and both types of documents can be flattened via commands in the the DTD/Schema menu. When a DTD is flattened, components in included/imported modules are saved directly in the parent file, and unused components are deleted.

Generate Sample XML file
You can generate, via the DTD/Schema | Generate Sample XML/JSON File menu command, a skeleton XML document based on the active DTD or XML Schema file. This is very useful for quickly creating an XML file based on the active schema.

Go to definition
When the cursor is located within a node in an XML document, clicking the DTD/Schema | Go to Definition menu command opens the schema file and highlights the definition of the selected XML node.
6.1 DTDs

A DTD document can be edited in Text View and Grid View. The default view can be set in the File Types section of the Options dialog.

Text View

In Text View, the document is displayed with syntax coloring and must be typed in. Given below is a sample of a DTD fragment:

```xml
<!-- Element declarations -->
<!ELEMENT document (header, para, img, link)>
<!ELEMENT header (#PCDATA)>
<!ELEMENT img EMPTY>
  <!ATTLIST img
    src CDATA #REQUIRED
  >

<!-- Notation Declarations -->
<!NOTATION GIF PUBLIC "urn:mime:img/gif">
```

Indentation is indicated by indentation guides and is best obtained by using the tab key. The amount of tab indentation can be set in the Text View Settings dialog.

Grid View

In Grid View, the DTD document is displayed as a table. The screenshot below shows the Grid View display of the DTD listed above.

When the cursor is inside a row of the table, or if a row is selected, DTD editing commands in the XML menu become enabled. You can insert, append, and add child nodes to the graphical representation of the DTD. The DTD items available at a particular selection point are enabled in the respective sub-menu of the XML menu (Insert, Append, Add Child). You can also convert a selected DTD item to another item, and move the item left or right in order to change its position in the document hierarchy. When a node is selected, available DTD items are also displayed as items in the entry helpers.
DTD features in XMLSpy

XMLSpy offers the following very useful features:

- **Convert DTD to XML Schema:** With the [DTD/Schema | Convert DTD to Schema](#) command, DTDs can be converted to XML Schemas.
- **Generate sample XML file from DTD:** With the [DTD/Schema | Generate Sample XML/JSON File](#) command, an XML document can be generated that is based on the active DTD.
6.2 XML Schemas

XML Schema documents can be edited in Text View, Grid View, and Schema View. The default view in which XML Schema documents open can be set in the File Types section of the Options dialog. You can switch between views while you edit, using the view that is most useful for the current purpose. XML Schema documents are typically saved with the extension .xsd or .xs.

Editing in Text View
In Text View an XML Schema is edited as an XML document; the editing features available for XML documents are also available for XML Schemas. As with all XML documents where the schema is identified and accessible, Text View entry helpers display the items available for addition at the cursor location point.

Editing in Grid View
In Grid View an XML Schema is edited as an XML document; the editing features available for XML documents are also available for XML Schemas. When an item in Grid View is selected, Grid View entry helpers display the items available for addition at the cursor location point.

Editing in Schema View
Schema View is a graphical interface for designing schemas. While you create/edit the schema in Schema View, XMLSpy generates a corresponding text document behind the interface. How to create and edit XML Schema documents in Schema View is described in detail in the section Editing Views | Schema View.

Altova website: XML Schema Editor

XML Schema features in XMLSpy
Additionally, XMLSpy offers the following very useful features:

- Convert XML Schema to DTD: With the DTD/Schema | Convert Schema to DTD command, XML Schemas can be converted to DTDs.
- Generate sample XML file from XML Schema: With the DTD/Schema | Generate Sample XML File command, an XML document can be generated that is based on the active XML Schema. Sample values can also be specified for elements and attributes in the sample XML.
6.3 Schema Subsets

One or more components of an XML Schema can be created as a separate schema file, known as a schema subset. The advantage of using smaller schema subsets to compose the larger schema (by means of Includes) is that the smaller files are more manageable than the single full schema.

In Schema View, one possible work scenario that describes various aspects of the Schema Subsets feature is as follows:

1. Create a schema subset that contains one or more components of the active schema. How to do this is described below.
2. Create additional schema subsets as required.
3. Include the newly created schema subset/s to compose the larger schema. Do this for each schema subset by appending or inserting an Include component in the Schema Overview window, and selecting the newly created schema subset file.
4. Delete any components that were present in the original full schema but are now duplicated because of the included subset/s.

You can also do the reverse in Schema View, that is, flatten the included schema subsets so that: (i) the components contained in the schema subsets are added directly to the main schema, and (ii) the included schema subsets are deleted from the main schema. How to flatten a schema is described further below.

Creating schema subsets

To create a schema subset, do the following:

1. With the required XML Schema active in Schema View, select the command Schema Design | Create Schema Subset. This pops up the Select Schema Components dialog (screenshot below).
2. In the dialog, check the component or components you wish to create as a single schema subset, then click Next. (Note that a check box below the pane enables components from all referenced files to also be listed for selection.)
3. In the Schema Subset Generation dialog that now appears (screenshot below), enter the name/s you want the file/s of the schema subset package to have. You must also specify the folder in which the new schema subset files are to be saved. A schema subset package could have multiple files if one or more of the components being created is an imported component in the original schema. A separate schema file is created for each namespace in the schema subset. The filenames displayed in the dialog are, by default, the names of the original files. But since you are not allowed to overwrite the original files, use new filenames if you wish to save the files in the same folder as the original files.
4. On clicking **OK**, the schema subset file with the namespace corresponding to that of the active file is opened in Schema View. Any other files in the package are created but not opened in Schema View.

**Flattening a schema**

Flattening the active schema in Schema View is the process of: (i) adding the components of all included schemas as global components of the active schema, and (ii) deleting the included schemas.

To flatten the active schema, select the command **Schema Design | Flatten Schema**. This pops up the Flatten Schema dialog (*screenshot below*), which contains the names of separate files, one for each namespace that will be in the flattened schema. These default names are the same as the original filenames. But since you are not allowed to overwrite the original files, the filenames must be changed if you wish to save in the same folder as the active file. You can browse for a folder in which the flattened schema and its associated files will be saved.
On clicking **OK**, the flattened schema file will be opened in Schema View.
6.4 Catalogs in XMLSpy

XMLSpy supports a subset of the OASIS XML catalogs mechanism. The catalog mechanism enables XMLSpy 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 XMLSpy works as outlined below.

RootCatalog.xml

When XMLSpy 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
<?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"
    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) to URIs that point to locally saved copies of the respective schemas. These schemas are installed in the Altova Common Folder when XMLSpy 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 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 XMLSpy application folder. The file CustomCatalog.xml is located in your MyDocuments\Altova\XMLSpy folder. The catalog.xml files are each in a specific schema folder, these schema folders being inside the folders: %AltovaCommonFolder%\Schemas.

### 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:

```bash
%AltovaCommonFolder%
%DesktopFolder%
%ProgramMenuFolder%
%StartMenuFolder%
%StartUpFolder%
%TemplateFolder%
%AdminToolsFolder%
%AppDataFolder%
%CommonAppDataFolder%
%FavoritesFolder%
%PersonalFolder%
```

- `%AltovaCommonFolder%` C:\Program Files\Altova\Common2019
- `%DesktopFolder%` Full path to the Desktop folder for the current user.
- `%ProgramMenuFolder%` Full path to the Program Menu folder for the current user.
- `%StartMenuFolder%` Full path to Start Menu folder for the current user.
- `%StartUpFolder%` Full path to Start Up folder for the current user.
- `%TemplateFolder%` Full path to the Template folder for the current user.
- `%AdminToolsFolder%` Full path to the file system directory that stores administrative tools for the current user.
- `%AppDataFolder%` Full path to the Application Data folder for the current user.
- `%CommonAppDataFolder%` Full path to the file directory containing application data for all users.
- `%FavoritesFolder%` Full path of the Favorites folder for the current user.
- `%PersonalFolder%` Full path to the Personal folder for the current user.
How catalogs work: DTDs

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 XMLSpy:

```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:

```xml
<catalog>
  ...
  <public publicId="-//W3C//DTD SVG 1.1//EN" uri="schemas/svg/svg11.dtd"/>
  ...
</catalog>
```
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).

How catalogs work: Schemas

In XMLSpy, you can also use catalogs to redirect to an XML Schema. In the XML instance file, the reference to the schema will occur in the xsi:schemaLocation attribute of the top-level document element of the XML document. For example,

```xml
<xsi:schemaLocation="http://www.xmlspy.com/schemas/orgchart OrgChart.xsd"/>
```

Normally, the URI part of the attribute's value (bold in the example above) is a path to the actual schema location. However, if the schema is referenced via a catalog, the URI part need not point to an actual XML Schema, but it does need to exist so that the lexical validity of the xsi:schemaLocation attribute is maintained. A value of foo, for example, would be sufficient for the URI part of the attribute's value. The schema is located in the catalog by means of the namespace part of the xsi:schemaLocation attribute's value. In the example above, the namespace part is http://www.xmlspy.com/schemas/orgchart. In the catalog, the following entry would locate the schema on the basis of that namespace part.

```xml
<uri name="http://www.xmlspy.com/schemas/orgchart" uri="C:\MySchemas\OrgChart.xsd"/>
```

The catalog subset supported by XMLSpy

When creating entries in CustomCatalog.xml (or any other catalog file that is to be read by XMLSpy), 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.

From release 2014 onwards, XMLSpy adheres closely to the XML Catalogs specification (OASIS Standard V1.1, 7 October 2005) specification. This specification strictly separates external-identifier look-ups (those with a Public ID or System ID) from URI look-ups (URIs that are not Public IDs or System IDs). Namespace URIs must therefore be considered simply URIs—not
Public IDs or System IDs—and must be used in URI look-ups rather than external-identifier look-ups. In XMLSpy versions prior to version 2014, schema namespace URIs were translated through <public> mappings. From version 2014 onwards, <uri> mappings have to be used.

Prior to v2014:  
uri="file:///C:/MyDocs/Catalog/test.xsd"/>

V-2014 onwards:  
.uri name="http://www.MyMapping.com/ref"  
uri="file:///C:/MyDocs/Catalog/test.xsd"/>

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 XMLSpy’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.

<spy:fileExtHelper ext="myhtml" uri="schemas/xhtml/xhtml1-transitional.dtd"/>

This would enable intelligent editing (auto-completion, entry helpers, etc) of .myhtml files in XMLSpy 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 specifications

XML Schema specification information is built into XMLSpy and the validity of XML Schema (.xsd) documents is checked against this internal information. In an XML Schema document, therefore, no references should be made to any schema that defines the XML Schema specification.

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 these schemas. The referenced files are included solely to provide XMLSpy 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.
6.5 Working with SchemaAgent

XMLSpy can be set up to work with Altova's SchemaAgent technology.

SchemaAgent technology
The SchemaAgent technology enables users to build and edit relationships between multiple schemas. It consists of:

- A SchemaAgent Server, which holds and serves information about the relationships among schemas in one or more search path/s (folder/s on the network) that you specify.
- A SchemaAgent client, Altova's SchemaAgent product, which uses schema information from the SchemaAgent server to which it is connected (i) to build relationships between these schemas; and (ii) to manage these schemas (rename, move, delete schemas, etc).

Two types of SchemaAgent server are available:

- Altova SchemaAgent Server, which can be installed on, and accessed from, a network, and
- Altova SchemaAgent, which is the SchemaAgent client product. It includes a lighter server version, called LocalServer, which can only be used on the same machine on which SchemaAgent is installed.

XMLSpy uses SchemaAgent technology to directly edit schemas in Schema View using information about other schemas it gets from a SchemaAgent server. In this setup, XMLSpy is connected to a SchemaAgent server, and, in interaction with SchemaAgent Client, sends requests to SchemaAgent Server. When XMLSpy has been set up to work with SchemaAgent, the Entry Helpers in Schema View not only list components from the schema currently active in Schema View but also list components from other schemas in the search paths of the SchemaAgent server to which it is connected. This provides you with direct access to these components. You can view the content model of a component belonging to another schema in Schema View, and reuse this component with or without modifications. You can also build relationships between schemas, thereby enabling you to modularize and manage complex schemas directly from within XMLSpy.

Installing SchemaAgent and SchemaAgent Server
For details about installing SchemaAgent and SchemaAgent Server and configuring search paths on servers, see the SchemaAgent user manual.

Setting up XMLSpy as a SchemaAgent client
In order for XMLSpy to work as a SchemaAgent client, you must do the following:

- Download SchemaAgent from the Altova website. You can now use SchemaAgent’s LocalServer to serve schemas. For information about configuring search paths on LocalServer, see the SchemaAgent user manual.
  Please note: SchemaAgent requires a valid license, which must be purchased after the free trial period runs out. Also note that Altova MissionKit product packages each includes the SchemaAgent product and a license key for it. (The SchemaAgent Server application, however, is not included in Altova MissionKit packages.)
- Additionally, you might want to download and install the network-based SchemaAgent Server from the Altova website.
• Define the search path(s) for SchemaAgent server (also known as configuring SchemaAgent Server). A detailed description of how to do this is given in the SchemaAgent user manual. (A search path is a path to the folder containing the XML schemas that will be mapped for their relationships with each other.)
• Start a connection from within XMLSpy to a SchemaAgent server.

**Important:** All SchemaAgent and SchemaAgent-related products from Altova (including XMLSpy) starting with Version 2005 release 3 are **not compatible** with previous versions of SchemaAgent or SchemaAgent-related products.

**SchemaAgent commands in XMLSpy**
The SchemaAgent functionality in XMLSpy is available only in Schema View and is accessed via menu commands in the Schema Design menu (see screenshot) and by using the Entry Helpers in Schema View.

The menu commands provide general administrative functionality. The Entry Helpers (and standard GUI mechanisms, such as drag-and-drop) are used to actually edit schemas.

This section describes how to use the SchemaAgent functionality available in Schema View.

6.5.1 **Connecting to SchemaAgent Server**

**Please note:** SchemaAgent Client must be installed in order for you to be able to make a connection.

Before you connect to SchemaAgent Server, only the **Connect to SchemaAgent Server** command is enabled in the **Schema Design** menu; other SchemaAgent commands in the **Schema Design** menu are disabled (see screenshot). The other menu items become enabled once a connection to a SchemaAgent Server has been successfully made.

**Connection steps**
To connect to a SchemaAgent server:

1. Click the Connect to SchemaAgent server toolbar icon (Schema Design | Connect)
to SchemaAgent Server). The Connect to SchemaAgent Server dialog (screenshot below) opens:

2. You can use either the local server (the SchemaAgent server that is packaged with Altova SchemaAgent) or a network server (the Altova SchemaAgent Server product, which is available free of charge). If you select Work Locally, the local server of SchemaAgent will be started when you click OK and a connection to it will be established. If you select Connect to Network Server, the selected SchemaAgent Server must be running in order for a connection to be made.

Note on servers running with Windows XP SP2
If the SchemaAgent Server name is listed in the Connect to SchemaAgent Server dialog but you cannot connect to it, it is possible that your server is not taking part in the name resolution process of your network. Name resolution is blocked by the default settings of the Windows XP SP2 Firewall.

To connect to such a server, do one of the following:

- Change the server settings to enable the name resolution process, or
- Enter the IP address of the server in the Edit field of the Connect Dialog box.

This need be done only once as SchemaAgent Client stores the connection string of the last successful connection.

Schema View after connecting to SchemaAgent server
After a connection to a SchemaAgent server is established, Schema View will look something like this:
Please note:

- At the top of the Globals view the text "Connected to SchemaAgent Server" appears, specifying the server to which the connection has been made.
- You now have full access to all schemas and schema constructs available in the server search path. SchemaAgent schema constructs such as global elements, complexTypes, and simpleTypes are visible in **bold blue text**, below the constructs of the active schema (**bold black text**).

Schema constructs can be viewed by Type (Globals), by Namespace, or by Identity Constraints in the respective tabs of the Components entry helper.

### 6.5.2 Opening Schemas Found in the Search Path

This example demonstrates how to open a schema found in a search path defined in SchemaAgent Server. It uses the `DB2schema.xsd` file available in the ..\Tutorial folder as the active schema. The Global tab of the Components entry helper is active.

1. Scroll down to the blue **Company** entry in the Components entry helper, and double-click it. The Goto Definition dialog box is opened.
2. Click the AddressLast.xsd entry, and click OK to confirm. This opens the addressLast.xsd schema and displays the content model of the Company element.

Please note: Double-clicking a SchemaAgent schema construct, such as Element, complexType, or simpleType, opens the associated schema (as well as all other included schemas) in XMLSpy.

6.5.3 Using IIRs
XML schema provides Import, Include, and Redefine (IIR) statements to help modularize schemas. Each method has different namespace requirements. These requirements are automatically checked by SchemaAgent Client and XMLSpy when you try to create IIRs.

Imports, Includes, and Redefines (IIRs)
Schema constructs can be "inserted" by different methods:

- Global elements can be dragged directly from the Components Entry Helper into the content model of a schema component (in Schema View).
- Components, such as complexTypes and simpleTypes, can be selected from the list box that automatically opens when defining new elements/attributes, etc.
- Components, such as complexTypes, can be selected from the Details Entry Helper when creating/updating these type of constructs.

Incorporating schema components
This example uses the DB2Schema.xsd file available in the ..\Tutorial folder as the active schema; the Global tab of the Components Entry Helper is active.

To use schema constructs from SchemaAgent Server schemas:
1. Make sure you are connected to a SchemaAgent server (see [Connecting to SchemaAgent server](#)).

2. Open and rename the DB2Schema.xsd file for this example, for example to Altova-office.

3. Click the icon of the Altova element in the Schema Overview to see its content model.

4. Right-click the Altova sequence compositor and select the menu option Add Child | Element. Note that a list box containing all global elements within the server path opens automatically at this point. Selecting one would incorporate that element.

5. Enter Altova-office as the name for this new element and press Enter.

6. Using the Details Entry Helper, click the type combo box and select the entry OfficeType.
7. Select `Orgchart.xsd` and click OK to confirm.

8. Click OK. The Import command was automatically selected for you. An expand icon appears in the `Altova-office` element.

Please note: The type entry in the Details entry helper has changed; it is now
displayed as `ns1:OfficeType` due to the fact that the `Orgchart.xsd` schema file has been imported and the target namespaces must be different in both schemas. An Import command has also been added to the schema.

9. Click the Expand button to see the `OfficeType` content model.

10. Press **F8** to validate the schema. The "Schema is valid" message should appear at this stage.

Cleaning up the schema:

1. Delete the `Division` element in the content model.
2. Click the Return to globals icon to switch to the Schema Overview.
3. Delete the following global elements: `Division`, `Person` and `VIP`. 
4. Select the menu option **Schema Design | Schema settings** to see how the namespace settings have changed.

The `ns1` prefix has been automatically added to the `www.xmlspy.com/schemas/orgchart` namespace. The Components (see screenshot) and Details Entry Helpers displays all imported constructs with the `ns1:` namespace prefix.

Please note:

- Changes made to schemas under SchemaAgent Server control using XMLSpy automatically update other schemas in the SchemaAgent Server path that referenced the changed schema.
- It is possible to see duplicates of constructs element, simpleTypes etc. in entry helpers (in black and blue), if the schema you are working on is also in the SchemaAgent Server path.
6.5.4 **Viewing Schemas in SchemaAgent**

To work with the active schema and its related schemas in SchemaAgent, select the menu option **Schema Design | Show in SchemaAgent | schema or related schemas** (see screenshot).

You have the option of opening only the active schema in SchemaAgent (**File only** command), or the active schema together with either (i) all directly referenced schemas, or (ii) all directly referencing schemas, or (iii) all directly related schemas.

6.5.5 **SchemaAgent Validation**

XMLSpy, in conjunction with SchemaAgent, allows you to validate not only the currently active schema but also schemas related to the currently active schema. We call this SchemaAgent validation. There are two types of related schemas that SchemaAgent distinguishes for extended validation: (i) directly dependent schemas (directly referenced and directly referencing schemas), and (ii) all dependent schemas (in addition to direct dependencies, these include indirect dependencies, which is the set of schemas that are related to another schema via an intermediary schema.

How to carry out SchemaAgent validation is demonstrated below by means of an example. This example assumes that the schema file `address.xsd` is the active schema in Schema View of XMLSpy. For the **SchemaAgent Validation** command to be enabled, make sure that the search paths on SchemaAgent Server contain the active file and some dependent files. Then do the following:

1. Click the **SchemaAgent Validation** icon in the toolbar or the menu item **Schema Design | SchemaAgent Validation**. This opens the SchemaAgent Validation dialog box (screenshot below), in which you can choose whether to validate the active schema only or one or more related schemas as well.

2. To insert schemas into the list, click the **Show Direct Dependencies** or **Show All**
Dependencies button as required. In this example, we have clicked the Show All Dependencies button, and this inserts all files that are directly referenced or indirectly referenced into the list.

<table>
<thead>
<tr>
<th>Schema files</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\Users\Documents\Altova\XMLSpy2013\Examples\address.xsd</td>
<td></td>
</tr>
<tr>
<td>C:\Users\Documents\Altova\XMLSpy2013\Examples\xiop.xsd</td>
<td></td>
</tr>
<tr>
<td>C:\Users\Documents\Altova\XMLSpy2013\Examples\WannullOrg.xsd</td>
<td></td>
</tr>
</tbody>
</table>

At this point, you can remove a schema from the list (Remove from List) if you wish to.

3. Click the Validate button to validate all the schemas in the list box.

<table>
<thead>
<tr>
<th>Schema files</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\Users\Documents\Altova\XMLSpy2013\Examples\address.xsd</td>
<td>DK</td>
</tr>
<tr>
<td>C:\Users\Documents\Altova\XMLSpy2013\Examples\xiop.xsd</td>
<td>Failed</td>
</tr>
<tr>
<td>C:\Users\Documents\Altova\XMLSpy2013\Examples\WannullOrg.xsd</td>
<td>DK</td>
</tr>
</tbody>
</table>

The Validate column displays whether the validation was successful or whether it failed.

You can now open all the non-valid schemas or a set of selected non-valid schemas in XMSpy.
6.6 Find in Schemas

In Schema View, XML Schemas can be searched intelligently using XMLSpy’s Find & Replace in Schema View feature.

The Find and Replace in Schema View feature is enabled when a schema is active in Schema View. It is accessed in one of two ways:

- Via the Edit | Find and Edit | Replace menu commands.
- Via the Find and Replace buttons in the Find in Schemas window.

Clicking a command or a button pops up the Find or the Replace dialog, according to which command/button was clicked. The Replace dialog (screenshots below) is different from the Find dialog in that it has a text entry field for the Replace term.

The standard Replace dialog looks like this:

![Replace dialog](screenshots)

Clicking the More button expands the dialog to show additional search criteria (screenshot below).
Usage is as follows:

- Enter the search and replace terms in the Search and Replace text fields
- Specify the schema components to be searched in the Components tab
- Specify the properties of the components to be searched; this helps to narrow the search
- Set the scope of the search to the current document or project, or specify a folder to search
- Execute the command
- Use the Find in Schemas window to navigate to a component quickly

The Reset button at the bottom of the dialog resets the original settings, which are as follows:

- No search term, no replace term
- Components: all
- Namespaces: none specified
- Property restrictions: anywhere
- Additional property restrictions: none
- Scope: current file

Note: Regular expressions are not supported in the Replace field.
6.6.1 Search Term

The search term can be entered as a string (select the **String** radio button) or as a number (**Numeric** radio button).

**String search**

In a string search (**screenshot below**), the entry can be: (i) text; (ii) a QName; or (iii) a regular expression. For QName searches, the namespace is determined on the basis of either the prefix used in the document or by the namespace URI, either of which must be entered. In the screenshot below, the `ts:` prefix is the prefix used in the document to identify a certain namespace.

To search using a regular expression, check the Regular Expression check box and then enter the regular expression. Entry helpers for regular expressions are available in a menu that is activated by clicking the right-pointing arrowhead at the right of the Search entry field (**screenshot below**).

![Regular Expression Menu](screenshot below)

You can also select whether a search term must match a whole word in the document and/or whether the casing in the document must match. Use the check boxes below the text entry field to specify these options.

If you wish to search in referenced objects (such as a complexType definition or a global element), then check the Search In Referenced Objects check box. This option is available only in the Find dialog; it is disabled in the Replace dialog.
Numeric search
When the Numeric Search radio button is selected, the search term can be a single operator-and-number search parameter, or a set of two such operator-and-number search parameters joined by the logical connector AND or OR. In the screenshot below, there are two search term parameters which create a search term for all integers between, and including, 1 and 5.

6.6.2 Components
The search can be restricted to one or more component types and to one or more target namespaces. These options are available in the Components tab. Expand the Find or Replace dialog by clicking the More button. This will bring up the tabs for refining the search, one of which is the Components tab (screenshot below).

The Components tab consists of two parts: (i) for selecting the component types to be searched, and (ii) for selecting the target namespaces to be searched.

Component selection
You can enter the component types to be searched by clicking the Add icon located to the right of the text field (see screenshot above). This pops up the Component Restriction dialog (screenshot below), in which you can select the components to be searched by checking them. Checking the Components item at the top of the list selects all components (text entry: all). Unchecking it de-selects all components (text entry: none)—including individually selected
components. Individual components, therefore, can be selected only when the Components item is unchecked. The selected components are entered in the text field as a comma-separated list (see screenshot above).

Note: Each time the Components tab or the Find/Replace dialog is opened, the previous component selection is retained.

**Namespace selection**

To select one or more target namespaces to be searched, click the Add or Insert icons and enter the required namespace/s. If no target namespace is specified, then all target namespaces are searched. To delete a target namespace that has been entered in this pane, select the target namespace and click the Delete icon.
6.6.3 Properties

The search can be restricted to one or more component properties (details and facets) by using options in the Properties tab, as well as to match the contents of properties. Expand the Find or Replace dialog by clicking the More button, and then select the Properties tab (screenshot below).

The Properties tab consists of two parts: (i) for restricting the main search term (entered in the Find text box); and (ii) for adding additional content restrictions (which have their own match term); see the section Additional Restrictions below.

Properties selection

You can enter the property types to be searched by clicking the Add icon which is to the right of the text field (see screenshot above). This pops up the Property Restriction dialog (screenshot below), in which you can select the properties to be searched by checking them. The properties are organized in three groups: (i) Details; (ii) Facets; (iii) Advanced (such as the DerivedFrom property). Checking Details, Facets, or Advanced selects all properties in that group. Unchecking a group de-selects all properties in that group, including individually selected properties. Individual properties, therefore, can be selected only when the group item is unchecked. The selected properties are entered in the text field (see screenshot above).
Note: Each time the Properties tab or the Find/Replace dialog is opened, the previous properties selection is retained.

Additional restrictions
An additional restriction enables you to specify the value of the property to search for. For example, if you are looking for an element called state which has an enumeration MA (for the US state of Massachusetts), you could specify the value MA of the property enumeration with the Additional Restrictions option. You would do this as follows:

1. In the Additional Restrictions pane, click the Add or Insert icon (screenshot below).

2. This adds a row to the pane and pops up the Property Restriction dialog. Deselect all properties and select only the enumeration property (screenshot below).
3. In the text field at the top of the dialog, enter the enumeration value to be searched for, in this case, MA (see screenshot above).

4. Click OK. The additional restriction is entered in the newly created row in the Additional Restrictions pane (screenshot below).
In the screenshot above, notice that the search term is `ipo:state`. In the Properties tab, the `anywhere` specifies that all properties will be searched, but the additional restriction specifies that the search should be restricted to enumerations having a value of `MA`.

Multiple additional restrictions can be added to further narrow the search. To delete an additional restriction, select the additional restriction and click the **Delete** icon.

**Note:** Each time the Properties tab or the Find/Replace dialog is opened, the previous additional restriction/s are retained.

### 6.6.4 Scope

The scope of the search can be set in the Scope tab (screenshot below). You can select either file/s or the currently selected schema component in Schema View.
If the File/s option is selected, you can further specify one from among the following options:

- **Current file**: An additional option to search included, imported and redefined files is also available.
- **Open files**: All XML Schema (XSD) files that are open in XMLSpy. Only the Find All and Replace All commands are enabled; single-step searching is not available.
- **Project**: The currently active project is selected, with the option to skip external folders. Only the Find All and Replace All commands are enabled; single-step searching is not available. If the default view for the .xsd file extension (Tools | Options | File Types | Default View) is not Schema View, then the .xsd files are not searched.
- **Folder**: You can browse for the required folder; an option to search sub-folders is also available. Only the Find All and Replace All commands are enabled; single-step searching is not available. If the default view for the .xsd file extension (Tools | Options | File Types | Default View) is not Schema View, then the .xsd files are not searched.
- **Included, imported, and redefined files** can be included in the scope by checking the option for adding them to the scope.

In the Replace dialog, you can choose whether to copy the replacement to the file on disk or whether to open the file in XMLSpy. Do this by selecting the appropriate button in the dialog.

### 6.6.5 Find and Replace Commands

The **Find** command behaves differently in the Find and Replace dialogs. The behavior of the **Find** command in both dialogs and of the **Replace** command is described below.

#### Find dialog

After you have entered the search term and, optionally, other criteria to refine the search, you can click either the **Find (F3)** or **Find All** command (screenshot below).
Clicking the **Find (Ctrl+F)** command in the dialog closes the Find dialog and finds the next occurrence of the search term within the specified scope and refinement criteria. The next occurrence is found relative to the currently selected component in Schema View. If the search reaches the end of the scope, it will not start automatically from the beginning of the scope. Therefore, you should make sure that the currently selected component in Schema View before starting the search is located before the document part you wish to search.

The result of the **Find** is highlighted in Schema View and the result is also reported in the Find In Schemas window. In the Find In Schemas window, you can click a result to highlight that item in Schema View.

Clicking the **Find All** command closes the Find dialog and lists all the search results in the Find In Schemas window.

**Replace dialog**

In the Replace dialog (**screenshot below**), clicking the **Find** command finds the next occurrence of the search term relative to the current selection in Schema View. You can then click **Replace** to replace this occurrence.

The **Find All** command closes the Replace dialog and lists all the search results in the Find In Schemas window.

The **Replace All** command replaces all occurrences of the found term, closes the Replace dialog, and lists the found terms in the Find In Schemas window.
Note: Regular expressions are not supported in the Replace field.

6.6.6 Results and Information

Each time a Find, Find All, Replace, or Replace All command is executed the results of the command execution are displayed in the Find In Schemas window (screenshot below). The term that was searched for is displayed in green; (in the screenshot below, it can be seen that email was the search term, with no case restriction specified). Notice that the location of the schema file is also given.

The Find All and Replace All commands list all the found occurrences in the document.

Note: The Find and Replace buttons at the top of this window bring up the Find dialog and the Replace dialog, respectively. The Find button can be used to find the next occurrence of the search term.

Features of the Find In Schemas window

Results are displayed in nine separate tabs (numbered 1 to 9). So you can keep the results of one search in one tab, do a new search, and compare results. Clicking on a result in the Find In Schemas window pops up and highlights the relevant component in the Main Window of Schema View. In this way you can search and navigate quickly to the desired component.

The following Find In Schema toolbar commands are available:

- The Next and Previous icons select, respectively, the next and previous messages to the currently selected message.
- The Copy Messages commands copy, respectively, the selected message, the selected message and its children messages, and all messages, to the clipboard.
- The Find commands find text in the Find In Schemas window.
- The Clear command deletes all messages in the currently active tab.
6.6.7 Finding and Renaming Globals

Named global components of XML Schemas can be found and renamed in a selected file and in all schema files related to the selected file. Named global components are all global components except: Include, Import, Redefine, Annotation, Comment, and PI components.

The process works as follows:

1. In Schema Overview, the global component to be found or renamed is selected.
2. In the context menu that pops up on right-clicking the selected component, select the required command (Find All References or Rename with All References).
3. In the dialog that pops up, select the scope of the search (or rename operation). In the case of a Rename operation, enter the new name of the global component.
4. On clicking OK, the search results are displayed in the Find in Schemas window (screenshot below).

The locations of all files in which references to the global component are found are listed (see screenshot above). All renamed components that were found and renamed are also listed.

Find All References

To open the Find All References dialog (screenshot below), do the following: (i) Right-click the global component in Schema Overview, (ii) In the context menu that pops up select the Find All References command.
The global component name is displayed in the **Component Name** field, which is grayed out and cannot be edited. You can choose whether the search should be carried out in the current file or in another file you can browse for (or select from a list of open files). You can also then specify whether related files (included, imported, redefined) should be searched, by checking the Add IIR Files check box at the bottom of the dialog.

### Rename with All References

To rename a global component, right-click it and select **Rename with All References** from the context menu that pops up. This pops up the Rename with All References dialog (**screenshot below**).

The new name you wish to give the selected global component must be entered in the **Component Name** text field. You can choose whether the search and renaming should be carried out in the current file or in another file you can browse for (or select from a list of open files). You can also then specify whether related files (included, imported, redefined) should be searched, by checking the Add IIR Files check box at the bottom of the dialog.
This section on XSLT is organized into the following sections:

- **Editing XSLT documents**: describes the editing support for XSLT documents in XMLSpy
- **XSLT Processing**: shows the various ways in which XSLT transformations can be carried out in the XMLSpy GUI using engines of your choice. This section also explains important XSLT settings in XMLSpy.
- **XSL Outline**: describes the XSL Outline and XSL Info Windows, which together provide a powerful way to view, navigate, and manage a collection of XSLT files.

**XPath Evaluation**

When an XML document is active, you can use the XPath/XQuery Window to evaluate XPath expressions. This is a very useful feature to quickly check how an XPath expression will be evaluated. Type in an XPath expression and specify whether it should be evaluated relative to the document root or to a selected context node in the XML document. The result of the evaluation will be displayed immediately in the XPath/XQuery Window. How to use the XPath/XQuery Window is described in the section Introduction | XPath/XQuery Window.

**XSLT Debugger**

XMLSpy also contains an XSLT Debugger to help you create correct and efficient XSLT stylesheets faster. These two features are described in the section XSLT and XQuery.

**Additional XSLT features**

Additional and more detailed information about the various features described in this section is in the descriptions of the relevant menu commands (in the User Reference section).

**Altova XSLT Engines**

For details about how the Altova XSLT 1.0, 2.0, and 3.0 Engines are implemented, see XSLT and XQuery Engine Information in the Appendices.

**RaptorXML for command line and batch processing**

The XMLSpy GUI enables batch processing via the projects functionality. However, if you are looking for more flexibility, you should try Altova's RaptorXML product, which provides fast XML validation, XSLT transformation, and XQuery execution functionality. RaptorXML is ideal if you wish to perform XSLT transformations from the command line, or batch processing.
7.1 XSLT Documents

XSLT 1.0, 2.0, and 3.0 documents can be edited in Text View and Grid View, and are edited like any other XML document in Text View and Grid View. The default view in which an XSLT document is opened can be set in the File Types section of the Options dialog.

Entry helpers

Entry helpers are available for elements, attributes, and entities. Information about the items displayed in the entry helpers is built into XMLSpy, and is not dependent on references contained in the XSLT document.

The following points should be noted:

1. If a new XSLT document is created via the Create a New Document dialog (File | New), then the appropriate XSLT elements and attributes (XSLT 1.0, XSLT 2.0, or XSLT 3.0, depending upon which document type was created) are loaded into the entry helpers. Additionally, HTML elements and attributes are loaded, as well as the HTML 4.0 entity sets, Latin-1, special characters, and symbols.

2. If an XML document is created via the Create a New Document dialog (File | New) and given XSLT content, no entry helper items are available except for XML character entities.

3. If an XSLT document is opened that was created as an XSLT document via the Create a New Document dialog (File | New), then the entity helpers will be as in Point 1 above.

4. If an XSLT document is opened that was not created as an XSLT document via the Create a New Document dialog (File | New), then the entity helpers will be as in Point 1 above. Additionally, XSL-FO elements and attributes will be listed in the Text View entry helpers.

5. The prefixes of elements in the Elements entry helper are as follows and are invariable: xsl: prefix for XSLT elements; no prefix for HTML elements; fo: prefix for XSL-FO elements. Consequently, in order to use the entry helpers, the namespace declarations in the XSLT document must define prefixes that match the built-in prefixes shown in the entry helpers.

Auto-completion

In Text View, auto-completion is available in a pop-up as you type. The first item in the pop-up list that matches the typed text is highlighted. When an element is being typed, a list of elements pops up with the first nearest match in alphabetical order being highlighted. Similarly, when an
**attribute** is being typed in, a list of applicable attributes pops up. The items in the list are determined according to the rules described in the previous section about entry helpers.

**XPath intelligent editing**
At locations in the XSLT document where XPath expressions can be entered (for example, inside the value of a `select` attribute, inside attribute value templates, and XSLT 3.0 value templates), the following features are available.

- Syntax coloring for the XPath constructs, including matching brackets during typing
- A hover tip if the cursor is placed over an XPath function. This contains information about the function.
- XPath functions and axes are suggested in popups as you type. You can move up or down the list of suggestions with the Up/Down cursors. If the function that is highlighted in the popup is a function, then information about the function (its signature) is displayed in an additional popup.
- If an XML file has been assigned in the Info window, then the elements and attributes of the XML file will also be available in the popup.

**Validating XSLT documents**
The XSLT document can be validated against the XSLT schema built into XMLSpy (click **XML | Validate (F8)**). The correct built-in schema is automatically selected according to whether the XSLT document is XSLT 1.0, XSLT 2.0, or XSLT 3.0 (specified in the `version` attribute of the `xsl:stylesheet` element).
7.2 XSLT Processing

In the XMLSpy GUI, two types of XSLT transformation are available:

- The **XSL/XQuery | XSL Transformation (F10)** command is used for straightforward XML transformations with an XSLT stylesheet to result formats specified and described in the stylesheets.
- The **XSL/XQuery | XSL-FO Transformation** command is used: (i) for transformations of XML to FO to PDF in two steps, and (ii) for one-step transformations of FO to PDF.

Specifying the XSLT processor for the transformation

The XSLT engine that will be used for transformations is specified in the **XSL section of the Options dialog** (screenshot below).

![Screenshot of Options dialog](image)

The available options are explained in the **User Reference** section. The engine specified in the XSL section will be used for all XSLT transformations. Note that for the XSL-FO transformation, an additional XSLT engine option is available: the XSLT engine that is packaged with some FO processors. To select this option, select the corresponding radio button at the bottom of the XSL section (see screenshot above).

Specifying the FO processor

The FO processor that will be used for transformations of FO to PDF is specified in the text box at the bottom of the **XSL section of the Options dialog** (screenshot above).

XSLT 1.0, 2.0, 3.0 and Altova’s XSLT engines

The XSLT version of a stylesheet is specified in the **version** attribute of the **xsl:stylesheet** (or **xsl:transform**) element. XMLSpy contains the built-in Altova XSLT 1.0, Altova XSLT 2.0, and
Altova XSLT 3.0 engines, and the appropriate engine is selected according to the value of the version attribute (1.0 or 2.0 or 3.0).

**XSLT Transformation**

The **XSLT Transformation (F8)** command can be used in the following scenarios:

- To transform an XML document that is active in the GUI and has an XSLT document assigned to it. If no XSLT document is assigned, you are prompted to make an assignment when you click the **XSLT Transformation (F8)** command.
- To transform an XSLT document that is active in the GUI. On clicking the **XSLT Transformation (F8)** command, you are prompted for the XML file you wish to process with the active XSLT stylesheet.
- To transform project folders and files. Right-click the project folder or file and select the command.

**XSL:FO Transformation**

The **XSL:FO Transformation** command can be used in the following scenarios:

- To transform an XML document that is active in the GUI and has an XSLT document assigned to it. The XML document will first be transformed to FO using the specified XSLT engine. The FO document will then be processed with the specified FO processor to produce the PDF output. If no XSLT document is assigned, you are prompted to make an assignment when you click the **XSL:FO Transformation** command.
- To transform an FO document to PDF using the specified FO processor.
- To transform an XSLT document that is active in the GUI. On clicking the **XSL:FO Transformation** command, you are prompted for the XML file you wish to process with the active XSLT stylesheet.
- To transform project folders and files. Right-click the project folder or file and select the command.

For a description of the options in the **XSL:FO output dialog**, see the **User Reference section**.

**Parameters for XSLT**

If you are using the Altova XSLT engines, XSLT parameters can be stored in a convenient GUI dialog. All the stored parameters are passed to the XSLT document each time you transform. For more information, see the description of the **XSLT Parameters / XQuery Variables** command.

**Batch processing with RaptorXML**

RaptorXML is a standalone application that contains Altova’s newest XML validator, XSLT engines, and XQuery engines. It can be used from the command line, via a COM interface, in Java programs, and in .NET applications to validate XML documents, transform XML documents using XSLT stylesheets, and execute XQuery documents.

XSLT transformation tasks can therefore be automated with the use of RaptorXML. For example, you can create a batch file that calls RaptorXML to transform a set of documents. See the **RaptorXML documentation** for details.
7.3 XSL Outline

When an XSLT document is the active document in XMLSpy, information about the structure of the document is displayed in the XSL Outline window and information about the files related to the active XSLT document is displayed in the XSLT tab of the Info Window (which is displayed only when an XSLT document is the active document in XMLSpy). Additionally, via these two windows, a number of commands are available that facilitate editing the XSLT document and managing files related to it.

In the XSL Outline window (screenshot below), you can do the following:

- View the templates and functions in the active XSLT document and in all imported and included XSLT documents.
- Sort the templates and functions on the basis of their names or match expressions, mode, priority, or comments.
- Search for specific templates on the basis of their names/expressions.
- Use the XSL Outline to navigate to the corresponding template in the XSLT document.
- Quickly insert calls to named templates.
- Set a selected named template as the entry point for transformations.

See the section XSL Outline window for details.

In the XSLT tab of the Info Window (screenshot below), you can do the following:
• View information about all the files related to the active XSLT document, such as the locations of imported and included files.
• Set an XML file for transformation with the active XSLT document. Also, the schema (XSD/DTD) file can be set for validating the selected XML file.
• Open a related file from within the Info Window.
• Quickly organize all related files into XMLSpy projects.
• Zip all related files to a user-defined location.

The XSL Outline window and the XSLT tab of the Info Window are described in detail in the subsections of this section.

### 7.3.1 XSL Outline Window

In the XSL Outline Window (*screenshot below*), all templates and functions in the active XSLT document are listed. Templates are indicated with blue icons (.templates without a parameter; and .templates containing parameters). Functions are indicated with a red .icon. In the combo box in the bottom left-hand of the window, you can select whether the templates and functions listed are from: (i) only the active XSLT document (as in the screenshot below), or (ii) the active XSLT document and all included and imported stylesheets.
There are two types of templates: (i) named templates, and (ii) templates that match an XPath expression. Each template is listed with:

- Its name (if the template has a *name* attribute) and/or XPath expression (if the template has a *match* attribute). If the template has both, a *name* and a *match* attribute, then both are listed, with the value of the *name* attribute first: *namevalue*, *matchvalue* (see the template named *bold* in the screenshot above).
- Its mode, if any. Note that a template may have more than one mode (see screenshot above).
- Its priority, if any;
- The comment that directly precedes the template or function, if any.

Functions in the stylesheet are listed by their names. Functions have neither mode nor priority.

**Operations**

The following operations can be performed in the XSL Outline Window:

- **Filtering**: The list displayed in the window can be filtered to show one of the following: (i) all templates and functions (the default setting each time XMLSpy is started); (ii) named templates only; (iii) XPath-expression templates only; (iv) functions only. To select the required filter, click the dropdown arrow to the right of the Search box at bottom right of the window (screenshot below), and select the required filter (the second group of commands in the menu). The selected filter is applied immediately and applies from this moment onwards till it is modified or till XMLSpy is closed.
*Sorting and locating:* Each column can be sorted alphabetically by clicking the column header. Each subsequent click reverses the previous sorting order. After a column has been sorted in this way, if you select any item in the list and then quickly type in a term from the sorted column, the first item in the list that contains that term will be highlighted. In this way, you can quickly go to templates of a particular name/expression, mode, or priority.

*Searching:* Enter in the Search box (at bottom right) the name or XPath expression for which you wish to search. The search results are displayed as you type. The following search options are available in the dropdown list of the Search box (*screenshot above*): (i) whether the name or expression either starts with or contains the search term (the first group of commands in the menu); the starts-with option is the default each time XMLSpy is started; (ii) whether the search results should be displayed as a reduced list or be highlighted (the third group of commands in the menu); the reduced-list option is the default each time XMLSpy is opened. These selections are applied immediately and remain in effect till changed or till XMLSpy is closed.

*Reloading:* After the stylesheet has been modified, click the Synchronize icon in the window's toolbar to update the XSL outline.

*Go to item:* When a template or function is selected in the XSL Outline window, clicking the the Go to Definition icon in the window's toolbar highlights the template or function in the document in Design View. Alternatively, double-click an entry to go to it.

*Named template actions:* Two groups of actions can be carried out involving named templates: (i) Calls to the named template (with `xsl:call-template`) can be inserted in the stylesheet at the cursor insertion point; and (ii) A named template can be set as the entry point for a transformation. The commands for these actions are carried out via icons in the toolbar and are described below.

**Template mode for transformation**

The combo box in the toolbar, called *Set mode for transformation*, lists (i) all the modes in the stylesheet, plus (ii) an empty entry (which selects the default mode) and, in the case of XSLT 3.0 stylesheets, (iii) the `#unnamed` mode. Selecting a mode from the dropdown list, sets the selected mode as the mode for the transformation. The `#unnamed` mode (for all XSLT versions) applies to all templates that have no `mode` attribute.

In the case of XSLT 1.0 and XSLT 2.0 stylesheets, the default mode is the `#unnamed` mode. So selecting the empty entry selects the default mode (which is the `#unnamed` mode and which therefore applies to all templates with no `mode` attribute).

In XSLT 3.0 stylesheets, the top-level `xslt` element can have a `default-mode` attribute, which
holds the default mode for the transformation. If, in the *Set mode for transformation* combo box, the empty entry (default mode) is selected, then the mode specified in the `default-mode` attribute will be used as the transformation mode. If the `#unnamed` mode is selected in the combo box, then the transformation will be applied to all templates with an unnamed mode, that is, to templates with no mode attribute.

**Note:** A template can be given a mode value of `#all` to make it applicable to all modes.

**Named templates**

When a named template is selected, one or more commands in the window's toolbar relating to named templates become enabled (screenshot below).

The commands in the toolbar (screenshot above) are, from left to right:

- **Insert `xsl:call-template`:** This command becomes active when a named template is selected in the XSL Outline window. The command inserts an `xsl:call-template` element at the cursor insertion point in the stylesheet. The `name` attribute of the `xsl:call-template` element that is inserted in the stylesheet is given a value that is the value of the `name` attribute of the selected named template. This makes the `xsl:call-template` a call to the selected named template.

- **Insert `xsl:call-template with param`:** This command becomes active when a named template having one or more `xsl:param` child elements is selected in the XSL Outline window. As with the **Insert `xsl:call-template`** command, the command inserts an `xsl:call-template` element, but in this case with a corresponding `xsl:with-param` child element for every `xsl:param` child element of the selected named template. The names of the inserted `xsl:call-template` and its `xsl:with-param` child elements correspond to the names of the selected named template and its `xsl:param` children.

- **Set the selected named template as entry point for transformation:** When a named template is set as the entry point for a transformation, transformations executed in XMLSpy start at this named template. In the XSL Outline Window, such a named template is indicated in boldface (see screenshot at the start of this section).

- **Clear named template as entry point for transformation:** Becomes active once a named template has been set as the entry point for transformations.

- **Jump to the named template selected as the entry point for transformations:** Becomes active once a named template has been set as the entry point for transformations. When the focus in the XSL Outline window is at some other point than the named template set as the entry point for transformations, clicking this icon highlights the named template in the XSL Outline window, thus making access to it faster.

### 7.3.2 Info Window

The XSLT tab of the Info Window, which is displayed only when an XSLT document is the active document in XMLSpy, displays all the imported and included XSLT files related to the active XSLT document, and enables the selection of an XML file that can be used as the source XML document on which the XSLT document is used for the transformation.
The following files are displayed in the XSLT tab of the Info Window:

- **XSLT files**: All imported and included XSLT files are listed (see screenshot above). The location of each file is displayed in a pop-up when the mouse cursor is placed over the file. Double-clicking an imported or included file, or selecting it and then clicking the **Open** icon in the Info Window toolbar, opens the file in a new window. The **Go to Include/Import Location** icon in the toolbar highlights the include/import declaration in the active XSLT document.

- **XML file**: An XML file can be assigned to the active XSLT stylesheet for transformations. The location of the assigned XML file is displayed in a pop-up when the mouse cursor is placed over the file. If an XML file is specified and the menu command **XSL/XQuery | XSL Transformation (F10)** is clicked, a transformation is executed on the defined XML file using the active XSLT document as the stylesheet. The XML file can be selected by clicking the **XML** icon and browsing; the selected file is displayed in bold face. Alternatively, the XML file can be assigned via the Project Properties dialog (XSL transformation of XSL files) or via a processing instruction in the XSLT document of the form: `<?altova_samplexml "Products.xml"?>`. In each case, the XML file will be shown in the Info Window with the relevant icon:
  - ![XML: Products.xml](assigned via the Project Properties dialog)
  - ![XML: "Products.xml"](assigned via a processing instruction in the XSLT document)
  - ![XML: Products.xml](assigned by clicking the XML icon and browsing for the required file; entry is in bold font face)

In the event that more than one of the above assignments exists, the selection priority is: (i) project; (ii) processing instruction; (iii) browsed by user. The XML file can be opened by double-clicking it or by selecting it and clicking the **Open** toolbar icon.

- **XSD/DTD file**: If the selected XML file has a reference to a schema (XML Schema or DTD), then this schema file is displayed in the XSD/DTD entry. Alternatively, just as with
the XML file, the schema file can be selected via the Project Properties dialog (Validation) or by clicking the XSD/DTD icon and browsing for the required schema file. If the schema file is selected via the Projects Properties dialog, a Projects icon is displayed next to the entry, otherwise the clickable XSD/DTD icon is displayed with the file entry either in a normal font face (when the schema is referenced from the XML file) or bold font face (schema browsed for by the user via the XSD/DTD icon). Should the schema file be assigned via more than one method, then the order of priority is as follows: (i) project; (ii) browsed by user; (iii) reference in XML document. The location of the assigned XSD file is displayed in a pop-up when the mouse cursor is placed over the file. The schema file can be opened by double-clicking it or by selecting it and clicking the Open toolbar icon.

Note: If an XML or XSD/DTD file is selected via the Project Properties dialog, then to clear this selection, you must go to the Project Properties dialog and clear the setting there. If the selection has been made by browsing via the XML or XSD/DTD icons, then to clear this setting, select the file and click the Clear icon in the Info Window toolbar.

Options

XPath intelligent editing: If an XML file has been assigned, the structure of the XML document is known and intelligent XPath editing will extend to elements and attributes. At locations in the XSLT document where an XPath expression can be entered, available elements and attributes will be shown in a popup. This option is switched on by default. To disable XPath intelligent editing, uncheck the check box. The setting is saved for each XSLT file separately when the file is closed, and will be used each time the file is opened.

Toolbar icons

The Info Window toolbar icons (screenshot below) are, from left to right:

- **Reload info**: Updates the Info Window to reflect modifications made in the XSLT document.
- **Clear XML/XSD assignment**: Clears an XML or XSD/DTD assignment made by the user by browsing via the XML or XSD/DTD icons, respectively. Select the file to clear and then click this icon.
- **Open document**: Opens the selected document.
- **Go to import/include location**: When an imported or included file is selected, clicking this icon highlights the relevant import or include declaration in the XSLT document.
- **Zip all local documents**: Zips all the documents listed in the Info Window to a user-defined location. Alternatively, only the selected documents can be zipped; do this by selecting, in the dropdown menu of this icon, the command Zip selected local documents.
- **Add all files to projects**: Adds all files to the current projects. Alternatively, only the selected documents can be added; do this by selecting, in the dropdown menu of this icon, the command Add selected files to project.
XQuery and XQuery Update documents can be edited in Text View. This view (see screenshot) provides entry helpers, syntax coloring, and intelligent editing to make editing easy. In addition, you can validate your XQuery document and run it (with an optional XML file if required) using the built-in Altova XQuery Engine.

**Note:** XQuery and XQuery Update files can be edited only in Text View. No other views of XQuery files are available.

**XQuery and XQuery Update file associations**

In XMLSpy, XQuery and XQuery Update documents are recognized as two different document types. Typically XQuery documents have the `.xq` extension, while XQuery Update documents have the `.xqu` file extension. You can associate additional file extensions with these filetypes, and also change filetype associations, at any time, in the File Type section of the Options dialog (Tools | Options | FileType).
The document type association of a file extension is important because, depending on this association, either an XQuery execution or an XQuery Update will be carried out when the XQuery/Update Execution command is run.

In this section

This section is organized as follows:

- Editing XQuery Documents
- XQuery Evaluation
- XQuery Validation
- XQuery Execution/Update
- XQuery Update Facility
- XQuery and XML Databases

Other related features and information:

- XSLT/XQuery Debugger and Profiler
- XQuery Engine Implementation
- Output Window: XPath/XQuery
- Tools | Options | File Types
- Tools | Options | XQuery

RaptorXML for command line and batch processing

The XMLSpy GUI enables batch processing via the projects functionality. However, if you are looking for more flexibility, you should try Altova's RaptorXML product, which contains Altova's newest XQuery Engine. RaptorXML is ideal if you wish to perform XQuery executions from the command line, or batch processing.
8.1 Editing XQuery Documents

In XMLSpy, XQuery and XQuery Update documents are recognized as two different document types. The document type (XQuery or XQuery Update) is assigned to a file extension in the File Types section of the Options dialog (Tools | Options | File Type, screenshot below). When a file of XQuery or XQuery Update type is opened in XMLSpy, the XQuery editing features of Text View are available for that file.

**File Types**

![File Types screenshot](image)

File extensions currently defined as XQuery and XQuery Update in XMLSpy

- XQuery: .xq, .xql, .xqr, .xquery
- XQuery Update: .xqu

**Note:** The editing features described in this section are identical for XQuery and XQuery Update documents.

**XQuery Execution/Update**

The GUI command XSL/XQuery | XQuery/Update Execution automatically runs either an XQuery execution or XQuery update depending on the filetype of the XQuery file that is selected to be run. See the section XQuery Execution/Update for more details.
8.1.1 XQuery Documents

An XQuery or XQuery Update document is opened automatically in XQuery editing mode of Text View if it is XQuery or XQuery Update conformant. A file is defined as conforming to a certain document type in the *File Types* section of the Options dialog (Tools | Options | FileType, screenshot below).

*File extensions currently defined as XQuery and XQuery Update in XMLSpy*

XQuery .xq .xql .xqr .xquery

XQuery Update .xqu

**Setting additional file extensions to be XQuery conformant**

To set additional file extensions to be XQuery conformant:

1. Select Tools | Options. The Options dialog appears (screenshot below).

![File Types screenshot](image)

2. Select the File Types section.
3. Click Add new file extension to add the new file extension to the list of file types.
4. Under Conformance, select XQuery conformant., and then XQuery or XQuery Update.

You should also make the following settings:

- **Description**: XML Query Language or XQuery Update Facility
- **Content type**: text/xml
• If you wish to use XMLSpy as the default editor for XQuery files, activate the Use XMLSpy as default editor check box.

8.1.2 XQuery Entry Helpers

There are three Entry Helpers in XQuery mode of Text View: XQuery Keywords (blue), XQuery Variables (purple), and XQuery Functions (olive).

Note the following points:

• The color of items in the three Entry Helpers are different and correspond to the syntax coloring used in the text. These colors cannot be changed.
• The listed keywords and functions are those supported by the Altova XQuery Engines.
• The variables are defined in the XQuery document itself. When a $ and a character are entered in Text View, the character is entered in the Variables Entry Helper (unless a variable consisting of exactly that character exists). As soon as a variable name that is being entered matches a variable name that already exists, the newly entered variable name disappears from the Entry Helper.
• To navigate in any Entry Helper, click an item in the Entry Helper, and then use either the scrollbar, mouse wheel, or page-down and page-up to move up and down the list.

To insert any of the items listed in the Entry Helpers into the document, place the cursor at the required insertion point and double-click the item. In XQuery, some character strings represent both a keyword and a function (empty, unordered, and except). These strings are always entered as keywords (in blue)—even if you select the function of that name in the Functions Entry Helper. When a function appears in blue, it can be distinguished by the parentheses that follow the function name.

8.1.3 XQuery Syntax Coloring

An XQuery document can consist of XQuery code as well as XML code. The default syntax coloring for the XQuery code is described in this section. The syntax coloring for XML code in an XQuery document is the same as that used for regular XML documents. All syntax coloring (for both XQuery code and XML code) is set in the Text Fonts section of the Options dialog (Tools | Options). Note that XQuery code can be contained in XML elements by enclosing the XQuery code in curly braces {} (see screenshot for example).
In XQuery code in the XQuery Mode of Text View, the following default syntax coloring is used:

- (: Comments, including 'smiley' delimiters, are in green :)  
- XQuery Keywords are in blue: keyword  
- XQuery Variables, including the dollar sign, are in purple: $start  
- XQuery Functions, but not their parentheses, are in olive: function()  
- Strings are in orange: "Procedure"  
- All other text, such as path expressions, is black (shown underlined below). So:
  
  ```xquery```
  ```
  for $s in doc("report1.xml")/section[section.title = "Procedure"]
  return ($s//incision)[2]/instrument
  ```

You can change these default colors and other font properties in the Text Fonts section of the Options dialog (Tools | Options).

**Note:** In the screenshot above, one pair of colored parentheses for a comment is displayed black and bold. This is because of the bracket-matching feature (see XQuery Intelligent Editing).
8.1.4 XQuery Intelligent Editing

The XQuery mode of Text View provides the following intelligent editing features.

- **Bracket-matching**
- **Keywords**
- **Variables**
- **Functions**
- **Visual guides**

**Bracket-matching**

The bracket-matching feature highlights the opening and closing brackets of a pair of brackets, enabling you to clearly see the contents of a pair of brackets. This is particularly useful when brackets are nested, as in XQuery comments (see screenshot below).

```
1 1 {::(Filename: seqQb.xq :)
2 2 {:: Source: http://www.w3.org/TR/xquery-use-cases ;:})
```

- Bracket-matching is activated when the cursor is placed either immediately before or immediately after a bracket (either opening or closing). That bracket is highlighted (bold black) together with its corresponding bracket. Notice the cursor position in the screenshot above.
- Bracket-matching is enabled for round parentheses `()`, square brackets `[]`, and curly braces `{}`. The exception is angular brackets `<>`, which are used for XML tags.

**Note:** When you place the cursor just inside a start or end bracket, both brackets are highlighted. Pressing Ctrl+E moves the cursor to the other member of the pair. Pressing Ctrl+E repeatedly enables you to switch between the start and end brackets. This is another aid to quickly navigating your document.

**Keywords**

XQuery keywords are instructions used in query expressions, and they are displayed in blue. You select a keyword by placing the cursor inside a keyword, or immediately before or after it. With a keyword selected, pressing Ctrl+Space causes a complete list of keywords to be displayed in a pop-up menu. You can scroll through the list and double-click a keyword you wish to have replace the selected keyword.

In the screenshot above, the cursor was placed in the *let* keyword. Double-clicking a keyword
from the list causes it to replace the `let` keyword.

Variables
Names of variables are prefixed with the `$` sign, and they are displayed in purple. This mechanism of the intelligent editing feature is similar to that for keywords. There are two ways to access the pop-up list of all variables in a document:

- After typing a `$` character, press **Ctrl+Space**
- Select a variable and press **Ctrl+Space**. (A variable is selected when you place the cursor immediately after the `$` character, or within the name of a variable, or immediately after the name of a variable.)

To insert a variable after the `$` character (when typing), or to replace a selected variable, double-click the variable you want in the pop-up menu.

Functions
Just as with keywords and variables, a pop-up menu of built-in functions is displayed when you select a function (displayed in olive) and press **Ctrl+Space**. (A function is selected when you place the cursor within a function name, or immediately before or after a function name. The cursor must not be placed between the parentheses that follow the function's name.) Double-clicking a function name in the pop-up menu replaces the selected function name with the function from the pop-up menu.

To display a tip containing the signature of a function (**screenshot below**), place the cursor immediately after the opening parenthesis and press **Ctrl+Space**. Note that the signature can be displayed only for standard XQuery functions.

The downward-pointing arrowhead indicates that there is more than one function with the same name but with different arguments or return types. Click on the arrowhead to display the signature of the next function (if available); click repeatedly to cycle through all the functions with that name. Alternatively, you can use the **Ctrl+Shift+Up** or **Ctrl+Shift+Down** key-combinations to move through a sequence.

Visual guides
Text folding (or source folding) is enabled on XQuery curly braces, XQuery comments, XML elements, and XML comments, and refers to the ability to expand and collapse these nodes.
Such nodes are indicated in the source folding margin by a +/- sign (see screenshot below). The margin can be toggled on and off in the Text View Settings dialog. When a node is collapsed, this is usually indicated by an ellipsis (see screenshot below). If the mouse cursor is placed over an ellipsis, the content of the collapsed node is displayed in a popup (see screenshot). If the content is too large for a popup, this is indicated by an ellipsis at the bottom of the popup.

```xml
declare function math:EulerSum( $k )
{ (....)
  sum( for $i in ( 1 to $k ) return 1 div ( $i + $i )

declare function math:Pi_Euler( $k )
{
  math:sqrt( 6 * math:EulerSum( $k )
};
```

The **Toggle All Folds** icon in the Text toolbar toggles all nodes to their expanded forms or collapses all nodes to the top-level document element. The following options are available when clicking on the node's +/- icon:

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Click [-]</strong></td>
<td>Collapses the node.</td>
</tr>
<tr>
<td><strong>Click [+]</strong></td>
<td>Expands the node so that descendant nodes are shown expanded or collapsed according to how they were before the node was collapsed.</td>
</tr>
<tr>
<td>Shift+Click [-]</td>
<td>Collapses all descendant nodes, but leaves the node that was clicked in its expanded form.</td>
</tr>
<tr>
<td>Ctrl+Click [+],</td>
<td>Expand the clicked node as well as all its descendant nodes.</td>
</tr>
</tbody>
</table>
8.2 XQuery Evaluation

XQuery expressions can be evaluated against one or more documents in the XPath/XQuery Output Window (screenshot below).

Do this as follows:

1. Enter the XQuery expression in the top pane of the window.
2. In the Where combo box (see screenshot above), select where the XML document to be queried is located. The options are: (i) Current file; (ii) Open files; (iii) Project; (iv) Folder.
3. Click Evaluate XPath/XQuery Expression (F5). The expression is evaluated against the XML file/s. If the specified (Where) location contains more than one XML file, all the XML files are searched for data structures or content matching the expression. Results of all available matches are displayed in the lower pane.

In the screenshot above, a query is made for a section element that has the attribute @id='intro'. The query returns the number of sub-sections of this intro section, and their titles.

For more information, see also Output Window: XPath/XQuery and Previewing and Applying XQuery Updates.
8.3 XQuery Validation

To validate an XQuery or XQuery Update document, do the following:

1. Make the XQuery document the active document.
2. Select XML | Validate, or press the F8 key, or click the Validate toolbar icon.

The document will be validated for correct XQuery syntax.
8.4 XQuery/Update Execution

An XQuery or XQuery Update document can be run in the following ways:

- When the XQuery or XQuery Update document is active.
- When an XML document is active.

**Note:** Whether a document is an XQuery document or XQuery Update document is determined by the document's file extension. XMLSpy recognizes file type associations according to the definitions made in Filetypes section of the Options dialog. ([Tools | Options] Filetypes).

**Note:** For XQuery Update, you can also enter Update expressions in the XPath/XQuery output window and preview updates. If the updates are acceptable, you can apply the updates and then save the updated file. See XQuery Update Facility and Previewing and Applying Updates for more details.

**Execution with XQuery or XQuery Update document active**

To execute an XQuery or XQuery Update document with the XQuery / XQuery Update document active, do the following:

1. Make the XQuery or XQuery Update document the active document.
2. Select XSL/XQuery | XQuery/Update Execution or click the command's toolbar icon. This opens the Define an XML Source for the XQuery dialog (screenshot below).

![XQuery/Update Execution toolbar icon](image)

3. Either browse for an XML file and execute, or skip the selection of an XML source.

Typically, an XQuery document is not associated with any single XML document. This is because XQuery expressions can select any number of XML documents with the `doc()` function. In XMLSpy, however, before executing individual XQuery documents you can select a source XML document for the execution. In such cases, the document node of the selected XML source is the starting context item available at the root level of the XQuery document. Paths that begin with a leading slash are resolved with this document node as its context item.

**Note:** The XQuery/Update Execution command is also available in the context menu of Project window items.
Result of execution / update

- **XQuery execution**: The result document is generated as a temporary file that can be saved to any location with the desired file format and extension.
- **XQuery update**: The update is saved to file, or the updated file is opened, allowing you to preview it, and then either save or close without saving. You can specify which of the two actions to carry out. This is done in the the XQuery section of the Options dialog (Tools | Options | XQuery).

Execution with XML document active

To execute an XQuery or XQuery Update document on an active XML document, do the following

1. Make the XML document the active document.
2. Select XSL/XQuery | XQuery/ Update Execution or click the command's toolbar icon. This opens the Choose XQuery/Update File dialog (screenshot below).

   ![XQuery/ Update Execution toolbar icon](image)

3. Browse for the XQuery or XQuery Update file and click **OK**.

Result of execution / update

- **XQuery execution**: The result document is generated as a temporary file that can be saved to any location with the desired file format and extension.
- **XQuery update**: The update is saved to file, or the updated file is opened, allowing you to preview it, and then either save or close without saving. You can specify which of the two actions to carry out. This is done in the the XQuery section of the Options dialog (Tools | Options | XQuery).

XQuery Variables

If you are using the Altova XQuery engines, XQuery variables can be stored in a convenient GUI dialog. All the stored variables are passed to the XQuery document each time you execute an XQuery document via XMLSpy. For more information, see the description of the XSLT Parameters / XQuery Variable command.

Altova XQuery Engines

For details about how the Altova XQuery Engines are implemented and will process XQuery files, see XQuery Engine Implementation.
8.5 XQuery Update Facility

The XQuery Update Facility is an extension of the XQuery language that enables parts of XML documents to be modified. In normal XQuery execution, the entire document is regenerated, and has to be stored back to its location. This could be inefficient when only small parts of the document need to be modified. With the Update Facility, only those parts of the document that need to be modified are updated.

The XQuery Update Facility is described as extensions to XQuery 1.0 and XQuery 3.1, in the following specifications, respectively:

- XQuery Update Facility 1.0 (W3C Recommendation of 17 March 2011)
- XQuery Update Facility 3.0 (W3C Working Draft of 19 February 2015)

The XQuery Update Facility in XMLSpy

The following points explain how XQuery Update works in XMLSpy:

- An update is carried out by an update expression. For example, an update expression can specify that a node in an XML document is renamed:
  rename node /documents/doc-01 as "document-01"
- In practice, multiple update expressions are entered in a single document—the XQuery Update document.
- As each update expression in the update document executes, the result is not applied immediately, but is added to a Pending Updates List (PUL). As a result, the PUL contains the results of all the update expressions. All updates in the PUL are then applied all together at once.
- In XMLSpy, the PUL updates are applied in one of two ways:
  (i) After being previewed by the user in the GUI. The advantage is that the update can be aborted if the preview shows undesirable results. Previewing is available on running the XQuery/Update Execution command, or on evaluating XQuery Update expressions in the XPath/XQuery output window. How to set the preview option is explained in the respective descriptions.
  (ii) Directly and without any user intervention. The advantage is that the update is carried out silently without requiring user intervention. The direct application of updates (without a preview) is available on running the XQuery/Update Execution command, or on evaluating XQuery Update expressions in the XPath/XQuery output window. How to set the direct-update option is explained in the respective descriptions.

XMLSpy provides a powerful XQuery Update Preview feature, which enables you to preview the effect of update expressions on the active XML document and then apply it. This feature is described in the section Previewing and Applying Updates.

8.5.1 Previewing and Applying Updates

If you wish to modify an XML document using XQuery Update, you can preview updates before applying them to the XML document and saving the modified document.

In the XPath/XQuery output window (screenshot below), you can enter one or more update expressions and then preview updates in the pending update list (PUL) that is displayed in the bottom pane (see screenshot below). If the PUL is as you want it, you can apply the updates to the document and then save the modified document. If you wish not to go ahead with the
modifications in the PUL, you can choose either to not apply modifications or to not save the file.

To create a PUL for an active XML file, do the following:

1. In the toolbar of the XPath/XQuery output window (screenshot above), select either the XQU 1.0 or XQU 3.0 icon (respectively for XQuery Update 1.0 or XQuery Update 3.0).
2. Enter one or more update expressions in the top pane of the window. For a description of update expressions and their syntax, see the section, Update Operations and Syntax.
3. In the toolbar’s scan-location combo box, select the location to be scanned for the updates:
   - Open files: All files that are currently open in XMLSpy will be scanned
   - Current file: Only the currently active file is scanned. If the location selected for scanning is Current file, then the the Evaluate XPath/XQuery Expression on Typing toolbar icon is enabled
   - Project: The currently active project is scanned
   - Folder: You can select a folder to scan
4. To execute the update expression/s and display the PUL, click the Evaluate XPath/XQuery Expression toolbar icon.

**XPath/XQuery output window toolbar**

The following XPath/XQuery output window toolbar commands (framed in red in the screenshot below) are available. They are, from left to right:

- The Show Header toggle specifies whether to show the file name of update locations or not.
- The Show Complete Result toggle specifies whether to show the entire node content or only attributes of the node.
- The Set XPath/XQuery Origin, if selected, sets the cursor location in the active file as the origin of relative XPath expressions. The expression is evaluated relative to this node.
Instead of manually entering the locator path expression of a node, you can let XMLSpy enter it for you. Do this as follows: (i) Place the cursor at the point in the XPath expression where you want to enter the locator path; (ii) Place the cursor in the start tag of the node you want to target; (iii) Click the Copies the XPath of the Current Selection to the Edit Field icon to enter the locator path in the expression. The locator path will be entered as an absolute path starting at the root node of the document.

- The Validate XML File toggles on/off XML file validation.
- The Evaluate XPath/XQuery Expression on Typing command is enabled when the parse-option is Current File. If the command is toggled on, the PUL is generated even as the update expression is being entered.
- The Evaluate XPath/XQuery Expression command evaluates the update expression/s and generates the PUL.
- The Scan Location combo box option is described above.

The Pending Update List (PUL) pane
The PUL pane shows all the updates that will be carried out. If the Show Header option has been toggled on in the window's toolbar, the locations of target files are displayed. The PUL display is divided into three vertical sections (see screenshot below): (i) the update action to carry out; (ii) the content of the target node to be updated; (iii) the update action result.

The following PUL pane toolbar commands are available:

- The Next and Previous icons select, respectively, the next and previous messages to the currently selected message.
- The Copy Line/s commands copy, respectively, the selected line and all lines to the clipboard.
- The Find commands find text in the PUL pane.
- The Clear command deletes all lines in the PUL pane.
- The Collapse Multiline Results command collapses multiline update list items to single lines.
- The Apply Update(s) command applies the pending updates to the target locations. On updating, the updates can be saved to file, or the updated file can be displayed (and subsequently saved manually or not). See the next option.
- The Update Files Directly combo box allows you to select (i) whether files are updated silently on disk, or (ii) whether updated files are opened and made active. If the latter option is selected, non-active target files are made active. You then have the choice of saving the modified document or not.

Note: If one or more files have been updated directly on disk, a list of changed files is displayed. Each item in the list shows the location of the file and is a clickable link to the file.
8.5.2 Update Operations and Syntax

The XQuery Update Facility enables the following operations:

- **Delete** one or several nodes
- **Insert** one or more nodes before, after, or inside a specified node
- **Rename** a node
- **Replace** a node with a sequence of items
- **Replace Value** of a node with the string value of a sequence of items

The keywords and syntax of these operations are described in the sub-sections of this section.

Delete Nodes

*Description and syntax*

Deletes one or more nodes.

```
delete node nodeSequence
delete nodes nodeSequence
```

*Details*

- The expression `nodeSequence` returns a sequence of the node/s to delete. All selected nodes will be marked for deletion.
- It does not matter whether the singular `node` or plural `nodes` is used. No correspondence is needed with the number of items in `nodeSequence`.

*Examples*

```
for $i in /book/section return
delete nodes $i/@id
```

Insert Nodes

*Description and syntax*

Inserts one or more nodes before, after, or inside the specified target node.

```
insert (node|nodes) items into targetNode
insert (node|nodes) items as first into targetNode
insert (node|nodes) items as last into targetNode
insert (node|nodes) items before targetNode
insert (node|nodes) items after targetNode
```

*Details*

- The expression `items` must return a sequence of items. Even though the keyword `node|nodes` is used, `items` can be a sequence of non-node items.
- The expression `targetNode` must point to a single target node.
- If the keyword `into` is used, `targetNode` must be an element node or document-element.
node.

- If the keyphrase `as first` or `as last` is used, the insertion is as first or last children, respectively.
- If the keyword `into` is used alone, then attributes are appended to existing attributes, and elements are inserted as first children.
- If the keyword `before` or `after` is used, `targetNode` can be of any type.
- If an attribute is being inserted, its name must not duplicate that of an already existing attribute.

**Examples**

```xquery
for $i in /book/section return
insert nodes (attribute id { 'somevalue' }, <newelement>some content including the numbers "{ 1 to 3}"</newelement>)
into $i
```

**Rename Node**

**Description and syntax**
Renames an element, attribute, or processing instruction node.

```xquery
rename node targetNode as name
```

**Details**

- The expression `targetNode` must point to a single target node, which can be an element, attribute, or processing instruction.
- The expression `name` must evaluate to a QName or string.
- If a QName is constructed, the mandatory namespace is declared locally.

**Examples**

```xquery
rename node /book/title as 'header-1'
```

```xquery
rename node /book/title as QName("http://www.altova.com/xquf", "header-1")
```

**Replace Node**

**Description and syntax**
Replaces a node with a sequence of any kind of items.

```xquery
replace node targetNode with items
```

**Details**

- The expression `targetNode` must point to a single target node.
- The expression `items` must return a sequence of items. This sequence will replace the target node.
- Except for attribute nodes, a target node can be replaced by any type of sequence.
An attribute node can only be replaced with an attribute node. See example below.

**Examples**

```xquery
replace node //hr with '<line/>
```

```xquery
for $i in //@height return replace node $i with (attribute line-height{'12pt'})
```

**Replace Value of Node**

**Description and syntax**
Replaces the value of a node with the string value of a sequence of items.

```xquery
replace value of node targetNode with items
```

**Details**
- The expression `targetNode` must point to a single target node.
- The expression `items` must return a sequence of items.
- The contents of the target node are replaced by the string value of the sequence returned by the `items` expression. This means that the target node will contain one text node only.

**Examples**

```xquery
for $i in //title return replace value of node $i with ('Draft Title')
```

**The fn:put Function**

The `fn:put` function is provided by XQuery Update Facility 1.0 as an extension to the XQuery built-in function library. (The `fn:` namespace prefix in this section is assumed to be bound to the `namespace: http://www.w3.org/2005/xpath-functions`.)

```xquery
fn:put($node as node(), $uri as xs:string) as empty-sequence()
```

The function stores a document or element to the location specified by `$uri`. It is normally invoked to create a resource on an external storage system such as a file system or a database. The external effects of `fn:put` are implementation-defined, since they occur outside the domain of XQuery. The intent is that, if `fn:put` is invoked on a document node and no error is raised, a subsequent query can access the stored document by invoking `fn:doc` with the same URI.

See the specification for more details.
8.6 **XQuery and XML Databases**

An XQuery document can be used to query an XML database (XML DB). Currently this XQuery functionality is supported only for IBM DB2 databases. The mechanism for querying an XML DB using XQuery essentially involves: (i) indicating to the XQuery engine that XML in a DB is to be queried—as opposed to XML in an XML document; and (ii) accessing the XML data in the DB.

The steps for implementing this mechanism are as follows and are described in detail below:

1. **Set up the XQuery document** to query the XML DB by inserting the XQUERY keyword at the start of the document.
2. For the active XQuery document, **enable DB support** (via the Info window) and **connect to the DB** (using the Quick Connect dialog).
3. In the XQuery document, insert **DB-specific XQuery extensions** so as to access the DB data and make it available for XQuery operations.
4. **Execute the XQuery** document in XMLSpy.

### Setting up the XQuery document to query the XML DB

To set up the XQuery document to query an XML DB, open the XQuery document (or create a new XQuery document) and enter the keyword **XQUERY** (casing is irrelevant) at the start of the document (before the prolog); **see examples below**.

```xml
XQUERY (: Retrieve details of all customers :)
declare default element namespace "http://www.altova.com/xquery/databases/db2";
<a> {db2-fn:xmlcolumn("CUSTOMER.INFO")} </a>
```

If the document uses the optional **xquery version expression**, the **XQUERY** keyword is still required:

```xml
XQUERY xquery version "1.0"; (: Retrieve details of all customers :)
<a> {db2-fn:xmlcolumn("CUSTOMER.INFO")} </a>
```

**Note:** XMLSpy's built-in XQuery Engines read the **XQUERY** keyword as indicating that an XML DB is to be accessed. As a result, attempting to execute an XQuery document containing the **XQUERY** keyword on any XML document other than one contained in an XML DB will result in an error.

### Enable DB support for XQuery and connect to the DB

DB support for an XQuery document is enabled by checking the Enable Database Support check box in the Info window **(screenshot below)**. Note that DB Support must be enabled for each XQuery document separately and each time an XQuery document is opened afresh.
When you enable DB support in the Info window, a Quick Connect dialog pops up, which enables you to connect to a database. Currently, only IBM DB2 databases are supported. How to connect to a DB is described in the section, Connecting to a Database. If connections to data sources already exists, then these are listed in the Data Sources combo box of the Info window (screenshot below), and one of these data sources can be selected as the data source for the active XQuery document. In the Info window, you can also select the root object from among those available in the Root Object combo box.

The Quick Connect dialog (which enables you to connect to a DB) can be accessed at any time by clicking the icon in the Info window.

**Note:** When you close an XQuery document the connection to the DB is closed as well. If you subsequently re-open the XQuery document, you will also have to re-connect to the DB.

**IBM DB2-specific XQuery language extensions**

Two IBM DB2-specific functions can be used in XQuery documents to retrieve data from an IBM DB2 database:

- `db2-fn:xmlcolumn` retrieves an entire XML column without searching or filtering the column.
- `db2-fn:sqlquery` retrieves values based on an SQL `SELECT` statement

The XML data retrieved using these functions can then be operated on using standard XQuery constructs. See examples below.

`db2-fn:xmlcolumn`: The argument of the function is a case-sensitive string literal that identifies an XML column in a table. The string literal argument must be a qualified column name of type XML. The function returns all the XML data in the column as a sequence, without applying a
search condition to it. In the following example, all the data of the INFO (XML) column of the CUSTOMER table is returned within a top-level <newdocelement> element:

XQUERY (: Retrieve details of all customers :)
declare default element namespace "http://www.altova.com/xquery/databases/db2";
<newdocelement> {db2-fn:xmlcolumn("CUSTOMER.INFO")} </newdocelement>

The retrieved data can then be queried with XQuery constructs. In the example below, the XML data retrieved from the the INFO (XML) column of the CUSTOMER table is filtered using an XQuery construct so that only the profiles of customers from Toronto are retrieved.

XQUERY (: Retrieve details of Toronto customers :) declare default element namespace "http://www.altova.com/xquery/databases/db2";
<newdocelement> {db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo[addr/city='Toronto'] } </newdocelement>

Note: In the example above, the document element of the XML files in each cell is customerinfo and the root node of the XML sequence returned by db2-fn:xmlcolumn is considered to be an abstract node above the customerinfo nodes.

db2-fn:sqlquery: The function takes an SQL Select statement as its argument and returns a sequence of XML values. The retrieved sequence is then queried with XQuery constructs. In the following example, the INFO column is filtered for records in the CUSTOMER table that have a CID field with a value between 1000 and 1004. Note that while SQL is not case-sensitive, XQuery is.

XQUERY (: Retrieve details of customers by Cid:) declare default element namespace "http://www.altova.com/xquery/databases/db2";

<persons>
{db2-fn:sqlquery("SELECT info FROM customer WHERE CID>1000 AND CID<1004")/}
<person>
  <id>{data(@Cid)}</id>
  <name>{data(name)}</name>
</person>
</persons>

The XQuery document above returns the following output:

<persons xmlns="http://www.altova.com/xquery/databases/db2">
<person>
  <id>1001</id>
  <name>Kathy Smith</name>
</person>
<person>
  <id>1002</id>
  <name>Jim Jones</name>
</person>
<person>
  <id>1003</id>
  <name>Robert Shoemaker</name>
</person>
</persons>

Note the following points:
- The default element namespace declaration in the prolog applies for the entire XQuery document and is used for navigation of the XML document as well as for construction of new elements. This means that the XQuery selector name is expanded to `<default-element-namespace>:name`, and that constructed elements, such as persons, are in the default element namespace.
- The SQL Select statement is not case-sensitive.
- The WHERE clause of the Select statement should reference another database item—not a node inside the XML file being accessed.
- The "/" after the db2-fn:sqlquery function represents the first item of the returned sequence, and this item is the context node for further navigation.

**Execute the XQuery**

To execute the XQuery document, select the XQuery Execution command (XSL/XQuery menu). Alternatively, press Alt+F10 or click the XQuery Execution icon 

The result of the execution is displayed in a new document.
9 XSLT/XQuery Debugger and Profiler

XMLSpy contains an XSLT/XQuery Debugger to help you create correct XQuery documents faster.

These two features are described in the sub-sections of this section.
9.1 XSLT and XQuery Debugger

The XSLT and XQuery Debugger enables you to test and debug XSLT stylesheets and XQuery documents. The XSLT and XQuery Debugger interface presents simultaneous views of the XSLT/XQuery document, the result document, and the source XML document. You can then go step-by-step through the XSLT/XQuery document. The corresponding output is generated step-by-step, and, if a source XML file is displayed, the corresponding position in the XML file is highlighted for each step. At the same time, windows in the interface provide debugging information.

The XSLT and XQuery Debugger always opens within a debugging session. Debugging sessions can be of the following types:

- XSLT 1.0, which uses the built-in Altova XSLT 1.0 engine
- XSLT 2.0, which uses the built-in Altova XSLT 2.0 engine
- XSLT 3.0, which uses the built-in Altova XSLT 3.0 engine
- XQuery 1.0, which uses the built-in Altova XQuery 1.0 engine
- XQuery 3.1, which uses the built-in Altova XQuery 3.1 engine

Note: XQuery Update Facility 1.0 and XQuery Update Facility 3.0 are currently not supported in XSLT and XQuery Debugger.

Which kind of debugging session is opened is determined automatically by the type of document from which the debugging session is opened (hereafter called the active document or active file). XSLT debugging sessions are opened from XSLT files (which version depends on the value of the version attribute of the xsl:stylesheet (or xsl:transform) element in the XSLT stylesheet (“1.0” for XSLT 1.0, “2.0” for XSLT 2.0, and “3.0” for XSLT 3.0)). XQuery debugging sessions are opened from XQuery files in a similar way. If the active file is an XML file, the selection depends on whether you choose to run an XSLT or XQuery file on the XML file; if the former, the selection further depends on whether the stylesheet is an XSLT 1.0, XSLT 2.0, or XSLT 3.0 stylesheet.

This information is summarized in the table below.

<table>
<thead>
<tr>
<th>Active File</th>
<th>Associated File</th>
<th>Debugging Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLT 1.0</td>
<td>XML; (required)</td>
<td>XSLT 1.0 (using built-in Altova XSLT 1.0 engine)</td>
</tr>
<tr>
<td>XSLT 2.0</td>
<td>XML; (required)</td>
<td>XSLT 2.0 (using built-in Altova XSLT 2.0 engine)</td>
</tr>
<tr>
<td>XSLT 3.0</td>
<td>XML or JSON; (required)</td>
<td>XSLT 3.0 (using built-in Altova XSLT 3.0 engine)</td>
</tr>
<tr>
<td>XQuery 1.0</td>
<td>XML; (optional)</td>
<td>XQuery 1.0 (using built-in Altova XQuery 1.0 engine)</td>
</tr>
<tr>
<td>XQuery 3.1</td>
<td>XML or JSON; (optional)</td>
<td>XQuery 3.1 (using built-in Altova XQuery 3.0 engine)</td>
</tr>
<tr>
<td>XML</td>
<td>XSLT 1.0, 2.0, or 3.0, or XQuery 1.0 or 3.1; (required)</td>
<td>XSLT 1.0, 2.0 or 3.0, or XQuery 1.0 or 3.1. XSLT version depends on value of version attribute of xsl:stylesheet (or xsl:transform) element of XSLT stylesheet.</td>
</tr>
</tbody>
</table>

For details about the three Altova engines, please see the XSLT and XQuery Engine Information section in the Appendices.
Automating XSLT and XQuery tasks with Altova RaptorXML Server 2019

Altova RaptorXML Server contains Altova’s XML Validator, XSLT 1.0, 2.0, and 3.0 engines, and XQuery 1.0 and 3.1 engines. It can be used from the command line, via a COM interface, in Java programs, and in .NET applications to validate XML documents, transform XML documents using XSLT stylesheets, and execute XQuery documents.

XSLT and XQuery tasks can therefore be automated with the use of Altova RaptorXML Server. For example, you can create a batch file that calls Altova RaptorXML to transform a set of XML documents or to execute a set of XQuery documents. See the RaptorXML Server documentation for details.

Altova website: XSLT Debugger, XQuery Debugger

9.1.1 Mechanism and Interface

The broad mechanism used for debugging XSLT and XQuery files using the XSLT and XQuery Debugger is given below. Note that there is a difference between a debugging session and the debugger, even though both are started with the XSL/XQuery | Start Debugger / Go command. You must first start the debugging session, and then step through the XSLT or XQuery document with the debugger.

- Open a debugging session (with the XSL/XQuery | Start Debugger / Go command). The appropriate session (XSLT 1.0, XSLT 2.0, XSLT 3.0, XQuery 1.0 or XQuery 3.1) is selected on the basis of the active file (see XSLT and XQuery Debugger). The XSLT and XQuery Debugger works only in Text View and Grid View. If the view of the active document is not Text View or Grid View when you start the debugging session, you will be prompted for permission to change the view to Text View, which is the default view of the XSLT and XQuery Debugger. You can, in the Debugger Settings dialog, also choose to have this option set permanently.

- Step through the XSLT or XQuery document using the Step Into, Step Out, and Step Over commands in the XSL/XQuery menu. (If you click the Start Debugger / Go command at this point, the debugger will go through the entire transformation or execution, stopping only at breakpoints. If no breakpoint has been set, it will go through the transformation in one step, without showing any debug results.) If an XML file is associated with the session, the corresponding locations in the XML file are highlighted. Simultaneously, output for corresponding steps is generated in the result file and the result document is built up step-by-step. In this way, you can analyse what each statement of the XSLT or XQuery file does.
Alternatively to the view of the three documents (XML, XSLT/XQuery, Output) shown above, a view of two documents (XSLT/XQuery and Output), or a view of any one of the documents can be selected.

- While a debugging session is open, information windows in the interface provide information about various aspects of the transformation/execution (Variables, XPath Watch, Call Stack, Messages, Info, etc).
- While a debugging session is open, you can stop the debugger (not the same as stopping the debugging session) to make changes to any of the documents. All the editing features that are available in your XMLSpy environment are also available while editing a file during a debugging session. When the debugger is stopped, the XSLT and XQuery Debugger interface stays open, and you have access to all the information in the information windows. After stopping the debugger in a debugging session, you can restart the debugger (from the beginning of the XSLT/XQuery document) within the same debugging session.
- Breakpoints can be set in the XSLT file to interrupt the processing at selected points. This speeds up debugging sessions since you do not have to step through each statement in the XSLT or XQuery document manually.
- Tracepoints can be set in the XSLT file. For instructions where a tracepoint is set, the value of that instruction is output when the instruction is reached.
- Stop a debugging session. This closes the XSLT and XQuery Debugger interface and returns you to your previous XMLSpy environment. The information in the information windows is no longer available. Breakpoint and tracepoint information, however, is retained in the respective files till the file is closed. (So if you start another debugging session involving a file containing breakpoints, the breakpoints will apply in the newly opened debugging session.)

**Note:** The Debugger toolbar with Debugger icons appears automatically when a debugging session is started.
9.1.2 Commands and Toolbar Icons

Debugger commands are available in the XSL/XQuery menu and as toolbar icons. The debugger icons are automatically made available in the toolbar when a debugging session is opened. These icons are listed below.

Start Debugger/Go (Alt+F11)
Starts or continues processing the XSLT/XQuery document till the end. If breakpoints have been set, then processing will pause at that point. If the debugger session has not been started, then this button will start the session and stop at the first node to be processed. If the session is running, then the XSLT/XQuery document will be processed to the end, or until the next breakpoint is encountered. If tracepoints have been set, the value of the instruction for which the tracepoint was set is displayed in the Trace window when that instruction is reached.

View the active document only
Maximizes the window of the currently active document in the Debugger interface.

View XSLT/XQuery and Output
Displays the XSLT and Output documents in their windows, while hiding the XML document.

View XML, XSLT/XQuery and Output
Displays the XML, XSLT/XQuery, and Output documents. This is the default view when an XML document is associated for the debugging session.

Stop Debugger
Stops the debugger. This is not the same as stopping the debugger session in which the debugger is running. This is convenient if you wish to edit a document in the middle of a debugging session or to use alternative files within the same debugging session. After stopping the debugger, you must restart the debugger to start from the beginning of the XSLT/XQuery document.

Step into (F11)
Proceeds in single steps through all nodes and XPath expressions in the stylesheet. This command is also used to re-start the debugger after it has been stopped.
Step Over (Ctrl+F11)
Steps over the current node to the next node at the same level, or to the next node at the next higher level from that of the current node. This command is also used to re-start the debugger after it has been stopped.

Step Out (Shift+F11)
Steps out of the current node to the next sibling of the parent node, or to the next node at the next higher level from that of the parent node.

Show current execution node
Displays/selects the current execution node in the XSLT/XQuery document and the corresponding context node in the XML document. This is useful when you have clicked in other tabs which show or mark specific code in the XSLT stylesheet or XML file, and you want to return to where you were before you did this.

Restart Debugger
Cleans the output window and restarts the debugging session with the currently selected files.

Insert/Remove Breakpoint (F9)
Inserts or removes a breakpoint at the current cursor position. Inline breakpoints can be defined for nodes in both the XSLT/XQuery and XML documents, and determine where the processing should pause. A dashed red line appears above the node when you set a breakpoint. Breakpoints cannot be defined on closing nodes, and breakpoints on attributes in XSLT documents will be ignored. This command is also available by right-clicking at the breakpoint location.

Insert/Remove Tracepoint (Shift+F9)
Inserts or removes a tracepoint at the current cursor position. Inline tracepoints can be defined for nodes in XSLT documents. During debugging, when a statement with a tracepoint is reached, the result of that statement is output in the Trace window. A dashed blue line appears above the node when you set a tracepoint. Tracepoints cannot be defined on closing nodes. This command is also available by right-clicking at the tracepoint location.

Enable/Disable Breakpoint (CTRL+F9)
This command (no toolbar icon exists) enables or disables already defined breakpoints. The red breakpoint highlight turns to gray when a breakpoint is disabled. The debugger does not stop at disabled breakpoints. To disable/enable a breakpoint, place the cursor in that node name and
click the Enable/Disable Breakpoint command. This command is also available by right-clicking at the breakpoint location.

Enable/Disable Tracepoint (Shift+CTRL+F9)
This command (no toolbar icon exists) enables or disables already defined tracepoints. The blue tracepoint highlight turns to gray when a tracepoint is disabled. No output is made in the Trace window for disabled tracepoints. To disable/enable a tracepoint, place the cursor in that node name and click the Enable/Disable Tracepoint command. This command is also available by right-clicking at the tracepoint location.

End Debugger Session
Ends the debugging session and returns you to the normal XMLSpy view that was active before you started the debugging session. Whether the output documents that were opened for the debugging session stay open depends on a setting you make in the XSLT/XQuery Debugger Settings dialog.

XSLT Breakpoints / Tracepoints Dialog
This command opens the XSLT/XQuery Breakpoints / Tracepoints dialog, which displays a list of all currently defined breakpoints/tracepoints (including disabled ones) in all files in the current debugging session.

The check boxes indicate whether a breakpoint/tracepoint is enabled (checked) or disabled. You
can remove the highlighted breakpoint/tracepoint by clicking the corresponding Remove button and remove all breakpoints or tracepoints by clicking the corresponding Remove All button. The corresponding Edit Code button takes you directly to the selected breakpoint/tracepoint in the file.

9.1.3 Debugger Settings

The XSLT and XQuery Debugger Settings dialog enables you to set debugging and output options that are applicable to all debugging sessions. To access the Settings dialog, click XSL/XQuery | Debug Settings or click the icon in the toolbar. The different settings are described below.

**Output Window**
Sets the view of the output document window (Default, Text, Grid, or Browser). The Default View is that selected for a file type (identified by its file extension, for example, .xslt or .xq in the File Types section of the Options dialog (Tools | Options). For XSLT transformations, the output file type is defined in the XSLT file. For XQuery executions, the output file type is determined by the serialization format you choose in the XQuery setting of this dialog (see below).

The Close All Output Windows option gives you the opportunity to keep open the output document windows that were opened in the debugging session when the debugging session ends.

**Debugging**
The Debug Built-in Templates setting causes the debugger to step into built-in templates code whenever appropriate. It is not related to the display of built-in templates when clicking this type of template entry in the Templates tab, or if the callstack shows a node from the built-in template file.
The XSLT Debugger works only in Text View or Grid View. The Auto Change to Text View option enables you to automatically switch to the Text View of a document for debugging if a document is not in Text View or Grid View. (The XQuery Debugger works in Text View only.) If the On demand variable execution check box is checked, the definition of a variable will be stepped into when the variable is called. Otherwise, the Debugger will not step into the variable definition when it encounters a call to a variable, but will carry on to the next step.

**Layout of Debugger Documents**

The Debugger Documents are the documents that are open in the Debugger. You can select whether these documents should be tiled vertically, horizontally, or XML/XSLT horizontally with the result document tiled vertically relative to the XML and XSLT.

### 9.1.4 Starting a Debugging Session

The simplest way to start a debugging session is to start one from an XSLT, XQuery, or XML file. If the required associated file (see Table of associated files) has already been assigned to the active file, then the debugging session is started immediately. Otherwise you are prompted to select the required associated file. Since XQuery files neither require nor contain an XML file association, you can choose to be prompted for an optional XML file association each time you start an XQuery debugging session from an XQuery file, or to not be prompted.

**Predefined associations**

Predefined associations are relevant only for XSLT debugging sessions, and refer to cases in which the associated file assignment is already present in the active file. To make an assignment in an XML or XSLT file, do the following:

- In XML/JSON files: Open the file, click XSL/XQuery | Assign XSL and select the XSLT file.
- In XSLT files: Open the file, click XSL/XQuery | Assign sample XML file, and select the XML file.

When you click XSL/XQuery | Start Debugger/Go, the debugging session is started directly, i.e. without you being prompted for any file to associate with the active file.

**Direct assignment**

If no predefined association is present in the active file, you are prompted for an association. When you select XSL/XQuery | Start Debugger/Go, the following happens:

- For XML files: You are prompted to select an XSLT, XQuery, or XQuery Update file.
- For JSON files: You are prompted to select an XSLT, XQuery, or XQuery Update file.
- For XSLT files: You are prompted to select an XML file.
- For XQuery files: You are given the option of selecting an XML file, which you can skip.
After you select the required associated file or skip an optional association, the debugging session is started.

Alternative method of file association

In the Project Properties dialog, you can make predefined associations. Click Project | Project properties, and assign the required files by clicking the Use this XSL / Use this XML check box.

Debugger View

The XSLT and XQuery Debugger works only in Text View and Enhanced Grid View. If either your XML or XSLT file is open in some other view than Text or Grid View, or if an SPS file is associated with an XML file, the following dialog pops up when you start a debugging session involving one of these files.

Clicking OK causes the document to open in Text View. Note that XQuery files are always displayed in Text View.

9.1.5 Information Windows

Information windows that are opened in the XSLT and XQuery Debugger interface during a debugging session contain information about various aspects of the XSLT transformation or
XQuery execution. This information is important in helping you debug your XSLT and XQuery files.

There are eight information windows in XSLT debugging sessions and five in XQuery debugging sessions. These windows are organized into two groups by default, which are located at the bottom of the XSLT and XQuery Debugger interface (see illustration below). These windows and the information they display are described in detail in this section.

The default layout of the XSLT and XQuery Debugger interface.

The first group of information windows displays the following windows as tabs in a single window:

- **Context** (for XSLT debugging sessions only)
- **Variables**
- **XPath Watch**

The second group of information windows displays the following windows as tabs in a single window:

- **Call Stack**
- **Messages**
- **Templates** (for XSLT debugging sessions only)
- **Info**
- **Trace**

In the default layout, therefore, there are two window groups, each having tabs for the different windows in them. One tab is active at a time. So, for example, to display information about Variables in the first information window group, click the **Variables** tab. This causes the Variables information window to be displayed and the Context and XPath-Watch information windows to be hidden. Note that in some tabs, you can use the information display as navigation tools: clicking an item can take you to that item in the XML, XSLT, or XQuery file. See the documentation of the respective information windows (**Context**, **Call Stack**, **Templates**) for details.

The two information window groups can be resized by dragging their borders. Individual windows
can be dragged out of the containing group by clicking the tab name and dragging the window out of the group. A window can be added to a group by dragging its title bar onto the title bar of the group. Note that there is no reset button to return the layout to the default layout.

**Context Window**

The Context Window is available in XSLT debugging sessions only; it is not available in XQuery debugging sessions.

During the processing of the XSLT stylesheet, the processor's context is always within a template that matches some sequence (of nodes or atomic values). The Context Window displays the current processing context, which could be a sequence of nodes, a single node, or an atomic value (such as a string). Depending on the kind of a context item, its value or attribute/s is/are displayed. For example, if the context item is an element, the element's attributes are displayed. If the context item is an attribute or text node, the node's value is displayed.

Clicking an entry in the Context Window, displays that item in the XML document. If the XML document is not currently displayed in the interface, a window for the XML document will be opened.

**Variables Window**

The Variables Window is available in XSLT and XQuery debugging sessions. It displays the variables and parameters that are used in the XSLT/XQuery document when they are in scope, and their values.
Parameters are indicated with `P`, global variables (declared at top-level of a stylesheet) are indicated with `G`, and local variables (declared within an XSLT template) are indicated with `L`. The type of the values of variables and parameters is also indicated by icons in the Value field. The following types are distinguished: Node Set, Node Fragment, String, Number, and Boolean.

**XPath-Watch Window**

The XPath-Watch Window is available in XSLT and XQuery debugging sessions.

It enables you to enter XPath expressions that you wish to evaluate in one or more contexts. As you step through the XSLT document, the XPath expression is evaluated in the current context and the result is displayed in the Value column.

To enter an XPath expression, double-click in the text field under the Name column and enter the XPath. Alternatively, drag an XPath expression from a file and drop it into the XPath-Watch Window. Use expressions that are correct according to the XPath version that corresponds to the XSLT version of the XSLT stylesheet (XPath 1.0 for XSLT 1.0, XPath 2.0 for XSLT 2.0, and XPath 3.0 for XSLT 3.0).

**Note:** If namespaces have been used in the XML file or XSLT file, you must use the correct namespace prefixes in your XPath expressions.

**Call Stack Window**

The Call Stack Window is displayed in XSLT and XQuery debugging sessions.

The Call Stack Window displays a list of previously processed XSLT templates and instructions,
with the current template/instruction appearing at the top of the list.

Clicking an entry in this window, causes the selected XSLT template/instruction to be displayed in the XSLT document window. Clicking a template/instruction that references a built-in template highlights the built-in template in a separate window that displays all built-in templates.

**Messages Window**

The Messages Window is displayed in XSLT and XQuery debugging sessions.

**XSLT 1.0, XSLT 2.0, and XSLT 3.0**

In XSLT debugging sessions, the Messages tab displays error messages, the `xsl:message` instruction(s), or any error messages that may occur during debugging.

**XQuery**

In XQuery debugging sessions, the Messages Window displays error messages.

**Templates Window**

The Templates Window (see screenshot) is available in XSLT debugging sessions only; it is not available in XQuery debugging sessions.

The Templates Window displays the various templates used in the XSLT stylesheet, including built-in templates and named templates. Matched templates are listed by the nodes they match. Named templates are listed by their name. For both types of template, the mode, priority, and location of the template are displayed.
In the screenshot above, there are three matched templates in the XSLT stylesheet: a template which matches the document node /, and templates that match the n1:italic and n1:bold nodes. All the other templates are built-in templates (indicated with an entry to that effect in the Location field).

Clicking an entry in this window, causes the template to be highlighted in the XSLT document window. If you click a built-in template, the template is highlighted in a separate window that displays all the built-in templates.

**Info Window**

The Info Window is available in XSLT and XQuery debugging sessions. It provides meta information about the current debugging session. This information includes what debugger is being used, the names of the source and output documents, and the status of the debugger.

**Trace Window**

The Trace Window is displayed in XSLT and XQuery debugging sessions.
The Trace Window contains the element the tracepoint is set for, its location in the XSLT stylesheet and the result generated when that element is executed. Click on a row in the left side of the window to display the full result on the right.

Arranging the Info Windows

The Information Windows can be arranged inside the XSLT and XQuery Debugger interface. Windows can be docked in the interface, can float in it, and can be arranged as a collection of panes in a window. You can use the following mechanisms to arrange the windows.

Menu

In the XSL/XQuery menu, placing the cursor over the item Debug Windows pops up the list of Info Windows. You can hide or show individual windows by clicking the window.

Context Menu

The context menu can be accessed by right-clicking a window tab or title bar.

Drag-and-drop

You can drag a window by its tab or title bar and place it at a desired location.

Additionally, you can dock the window in another window or in the interface using placement controls that appear when you drag a window:

- When you drag a window over another window, a circular placement control appears (see screenshot below). This control is divided into five placement sectors. Releasing the
mouse key on any of these sectors docks the dragged window into the respective sector of the target window. The four arrow sectors dock the dragged window into the respective sides of the target window. The center button docks the dragged window as a tab of the target window. You can also dock a window as a tab in another window by dragging it to the tab bar and dropping it there.

- When you drag a window, a placement control consisting of four arrows appears. Each arrow corresponds to one side of the Debugger interface. Releasing a dragged window over one of these arrows docks the dragged window into one side of the Debugger interface.

You can also double-click the title bar of a window to toggle it between its docked and floating positions.

9.1.6 Breakpoints

The XSLT and XQuery Debugger enables you to define breakpoints in XSLT, XQuery, and XML documents. Breakpoints are displayed as a dashed red line (shown in the screenshot below).

Please note: It is possible to set a tracepoint and a breakpoint for the same instruction. This appears as a dashed blue and red line (see screenshot).

```xml
  <!-- Local Variable reused -->
  <xsl:variable name="OfficeName" select="n1 Name"/>
  <!-- Display the company name if the variable is true -->
  <xsl:if test="$Show_Company_Name">
    <h3>..........</h3>
    <xsl:value-of select="$OfficeName"/>
  </h3>
  <xsl:message>
    Company Named Displayed</xsl:message>
  </xsl:message>
</xsl:if>
```

When you start the debugger within a debugging session, the debugging will pause at each encountered breakpoint. In this way, you can identify specific areas to debug, and restrict attention to these areas in either the XSLT, XQuery, and/or XML documents. You can set any number of breakpoints.
**Note:** Breakpoints set for a document remain in that document until it is closed. However, if you switch to Schema View (for example, in the case of XSD documents), then the breakpoints are deleted; when you switch back to Text View or Grid View (from Schema View), there will be no breakpoint.

**Breakpoints in XML documents**

You can set breakpoints on any node in an XML document. The break in processing will occur at the start of that node.

**Breakpoints in XSLT documents**

You can set breakpoints at the following points in an XSLT document:

- At the beginning of templates and template instructions (e.g., `xsl:for-each`).
- On an XPath expression (XPath 1.0 or XPath 2.0).
- On any node in a literally constructed XML fragment. The break in processing will occur at the start of that node.

**Breakpoints in XQuery documents**

You can set breakpoints at the following points in an XQuery document:

- At the beginning of XQuery statements.
- In an XQuery expression.
- On any node in a literally constructed XML fragment. The break in processing will occur at the start of that node.

**Inserting/removing breakpoints**

To insert a breakpoint:

1. Place the cursor at the point in the document where you wish to insert the breakpoint (see paragraphs above). In XSLT debugging sessions, you can set breakpoints in both Text View and Grid View. XQuery debugging sessions are available only in Text View.
2. Do one of the following:
   - Select **XSL/XQuery | Insert/Remove Breakpoint**.
   - Press **F9**.
   - Right-click and select **Insert/Remove Breakpoint**.

To remove a breakpoint:

1. Place the cursor at the point in the document containing the breakpoint.
2. Do one of the following:
   - Select **XSL/XQuery | Insert/Remove Breakpoint**.
   - Press **F9**.
   - Right-click and select **Insert/Remove Breakpoint**.

Alternatively, you can use the Breakpoints dialog to remove a breakpoint:
1. Select the menu option **XSL/XQuery | Breakpoints...**
2. Click the breakpoint in the dialog box and click **Remove**.

The **Remove All** button deletes all the breakpoints from the dialog box (and all XSLT stylesheets).

**Disabling/enabling breakpoints:**

After inserting breakpoints, you can disable them if you wish to skip over breakpoints without having to delete them. You can enable them again when necessary.

To disable a breakpoint:

1. Place the cursor in the node or expression containing the breakpoint.
2. Select **XSL/XQuery | Enable/Disable Breakpoint** (or press **Ctrl+F9**). The breakpoint changes from red to gray, indicating that it has been disabled.

Alternatively, you can use the Breakpoints dialog to disable a breakpoint:

1. Select the menu option **XSL/XQuery | Breakpoints/Tracepoint...**. This opens the XSLT Breakpoints / Tracepoints dialog box which displays the currently defined breakpoints in all open XML source and XSLT stylesheet documents.
2. Remove the check mark of the breakpoints you wish to disable, and click **OK** to confirm. The breakpoint changes from red to gray, indicating that it has been disabled.

To enable a breakpoint:

1. Place the cursor in the node or expression containing the breakpoint.
2. Select **XSL/XQuery | Enable/Disable Breakpoint** (or press **Ctrl+F9**). The breakpoint
changes from gray to red, indicating that it has been enabled.

Finding a specific breakpoint
To find a specific breakpoint:

1. Select the menu option XSL/XQuery | Breakpoints/Tracepoints... The XSLT Breakpoints / Tracepoints dialog appears.
2. Click the required breakpoint in the breakpoint list.
3. Click the Edit Code button. The Breakpoints dialog box is closed and the text cursor is placed directly in front of the breakpoint in Text view. In the Enhanced Grid view, the table cell containing the breakpoint is highlighted in red.

Continuing debugging after a breakpoint
To continue debugging after a breakpoint:

- Select the XSL/XQuery | Step into or XSL/XQuery | Start Debugger/Go command.

9.1.7 Tracepoints
The XSLT and XQuery Debugger enables you to define tracepoints in XSLT documents.

Tracepoints allow you to trace content generated by an instruction or view the result of an XPath expression at the point where the tracepoint is set without having to edit the XSLT stylesheet, for example, using the xsl:message element to output debugging messages.

Tracepoints are displayed as a dashed blue line in XSLT stylesheets (shown in the screenshot below).

Note: It is possible to set a tracepoint and a breakpoint for the same instruction. This appears as a dashed blue and red line (see screenshot).
The debugger outputs the content generated by each instruction that has a tracepoint set for it. This output is visible in the Trace window. You can set any number of tracepoints in an XSLT stylesheet.

**Note:** Tracepoints set for a document remain in that document until it is closed.

**Tracepoints in XSLT documents**
You can set tracepoints on XSL instructions and literal results in an XSLT stylesheet.

**Tracepoints in XML and XQuery documents**
You can set tracepoints in XML and XQuery documents.

**Inserting/removing tracepoints**
To insert a tracepoint:

1. Place the cursor at the point in the XSLT document where you wish to insert the tracepoint. During debugging sessions, you can set tracepoints in both Text View and Grid View.
2. Do one of the following:
   - Select **XSL/XQuery | Insert/Remove Tracepoint**.
   - Press **Shift+F9**.
   - Right-click and select **Insert/Remove Tracepoint**.

To remove a tracepoint:
1. Place the cursor at the point in the XSLT document containing the tracepoint.
2. Do one of the following:
   - Select XSL/XQuery | Insert/Remove Tracepoint.
   - Press Shift+F9.
   - Right-click and select Insert/Remove Tracepoint.

Alternatively, you can use the XSLT Breakpoints / Tracepoints dialog to remove a tracepoint:

1. Select the menu option XSL/XQuery | Breakpoints/Tracepoints....
2. Click the tracepoint in the dialog box (see screenshot) and click Remove.

The Remove All button in the Tracepoints pane deletes all the tracepoints from the dialog box (and from all XSLT stylesheets).

**Setting an XPath for a tracepoint**

You can set an XPath for a tracepoint. When you set an XPath for a tracepoint, the result of the evaluation of the XPath is displayed in the Trace window instead of the content generated by the statement for which the tracepoint is set. The XPath is evaluated relatively to the context node at the point where the tracepoint is set.

To set an XPath for a tracepoint:

1. Select the menu option XSL/XQuery | Breakpoints/Tracepoints.... This opens the XSLT Breakpoints / Tracepoints dialog box which displays the currently defined tracepoints in all open XSLT stylesheet documents.
2. Enter the XPath in the XPath column in the row that corresponds to the tracepoint.

**Example**
In the example below, the tracepoint is set such that the context node is Person. The Person element contains a Shares element. We want to display the number of shares that each person has, multiplied by 125 (the value of each share).

Do the following:

1. Set a tracepoint at the line just after the `<xsl:for-each>` instruction that selects the `n1:Person` element (line 555 in the screenshot above).
2. Open the XSLT Breakpoints / Tracepoints dialog and enter the XPath `n1:Shares*125.00` for the tracepoint you just set.
3. Start the Debugger. The results of the XPath you entered for the tracepoint appear in the
The Trace window

Select XSL/XQuery | Start Debugger/Go to start debugging. The output of instructions for which tracepoints are set is displayed in the Trace window (see screenshot). Click a row in the Trace window to display the full result of that statement in the right side of the window (see screenshot).

Please note: Results are displayed in the Trace window only after the traced instruction is completed.

Disabling/enabling tracepoints

After inserting tracepoints, you can disable them if you wish to skip over them without having to delete them. You can enable them again when necessary.

To disable a tracepoint:

1. Place the cursor at the point in the XSLT stylesheet containing the tracepoint.
2. Select XSL/XQuery | Enable/Disable Tracepoint (or press Ctrl+Shift+F9). The tracepoint changes from blue to gray, indicating that it has been disabled.

Alternatively, you can use the XSLT Breakpoints / Tracepoints dialog to disable a tracepoint:

1. Select the menu option XSL/XQuery | Breakpoints/Tracepoints... This opens the XSLT Breakpoints / Tracepoints dialog box which displays the currently defined tracepoints in all open XSLT stylesheet documents.
2. Remove the check mark of each tracepoint you wish to disable, and click OK to confirm. The tracepoints change from blue to gray, indicating that they have been disabled.

To enable a tracepoint:

1. Place the cursor at the point in the XSLT document containing the tracepoint.
2. Select XSL/XQuery | Enable/Disable Tracepoint (or press Ctrl+Shift+F9). The tracepoint changes from gray to blue, indicating that it has been enabled.

Finding a specific tracepoint

To find a specific tracepoint:

1. Select the menu option XSL/XQuery | Breakpoints/Tracepoints.... The XSLT Breakpoints / Tracepoints dialog appears.
2. Click the required tracepoint in the tracepoint list.
3. Click the Edit Code button. The XSLT Breakpoints / Tracepoints dialog box is closed and the text cursor is placed directly in front of the tracepoint in Text view of the XSLT document. In Enhanced Grid view, the table cell containing the tracepoint is highlighted in blue.

9.1.8 Debugger Shortcuts

The default debugger shortcuts are listed below.
<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>F9</td>
<td>Insert/Remove Breakpoint</td>
</tr>
<tr>
<td>F9 + Shift</td>
<td>Insert/Remove Tracepoint</td>
</tr>
<tr>
<td>F9 + CTRL</td>
<td>Enable/Disable Breakpoint</td>
</tr>
<tr>
<td>F9 + Shift + CTRL</td>
<td>Enable/Disable Tracepoint</td>
</tr>
<tr>
<td>F11</td>
<td>Step Into</td>
</tr>
<tr>
<td>F11 + Shift</td>
<td>Step Out</td>
</tr>
<tr>
<td>F11 + CTRL</td>
<td>Step Over</td>
</tr>
<tr>
<td>F11 + Alt</td>
<td>Start Debugger/Go</td>
</tr>
</tbody>
</table>
Authentic View (screenshot below) is a graphical representation of your XML document. It enables XML documents to be displayed without markup and with appropriate formatting and data-entry features such as input fields, combo boxes, and radio buttons. Data that the user enters in Authentic View is entered into the XML file.

Nanonull, Inc.

<table>
<thead>
<tr>
<th>Location:</th>
<th>US</th>
</tr>
</thead>
</table>

| Street:     | 119 Oakstreet, Suite 4876 |
| City:       | Vereno |
| State & Zip: | DC  29213 |
| Phone:      | +1 (321) 555 5155 0 |
| Fax:        | +1 (321) 555 5155 4 |
| E-mail:     | office@nanonull.com |

**Vereno Office Summary:** 4 departments, 15 employees.

The company was established in Vereno in 1995 as a privately held software company. Since 1996, Nanonull has been actively involved in developing nanoelectronic software technologies. It released the first version of its acclaimed NanoSoft Development Suite in February 1999. Also in 1999, Nanonull increased its capital base with investment from a consortium of private investment firms. The company has been expanding rapidly ever since.

To be able to view and edit an XML document in Authentic View, the XML document must be associated with a **StyleVision Power Stylesheet (SPS)**, which is created in Altova’s StyleVision product. An SPS (.sps file) is, in essence, an XSLT stylesheet. It specifies an output presentation for an XML file that can include data-entry mechanisms. Authentic View users can, therefore, write data back to the XML file or DB. An SPS is based on a schema and is specific to it. If you wish to use an SPS to edit an XML file in Authentic View, you must use one that is based on the same schema as that on which the XML file is based.

**Using Authentic View**

- If an XML file is open, you can switch to Authentic View by clicking the **Authentic** button at the bottom of the Main Window. If an SPS is not already assigned to the XML file, you will be prompted to assign one to it. You must use an SPS that is based on the same schema as the XML file.
- A new XML file is created and displayed in Authentic View by selecting the **File | New** command and then clicking the "Select a StyleVision Stylesheet" button. This new file is a template file associated with the SPS you open. It can have a variable amount of starting data already present in it. This starting data is contained in an XML file (a
Template XML File) that may optionally be associated with the SPS. After the Authentic View of an XML file is displayed, you can enter data in it and save the file.

- You can also open an SPS via the Authentic | New Document command. If a Template XML File has been assigned to the SPS, then the data in the Template XML File is used as the starting data of the XML document template created in Authentic View.

In this section
This section contains an Authentic View tutorial, which shows you how to use Authentic View. It is followed by the section, Editing in Authentic View, which explains individual editing features in detail.

More information about Authentic View
For more information about Authentic View, see (i) the section Authentic | Authentic View Interface, which describes the Authentic View editing window, and (ii) the Authentic menu section of the User Reference part of this documentation.
10.1 Authentic View Tutorial

In Authentic View, you can edit XML documents in a graphical WYSIWYG interface (screenshot below), just like in word-processor applications such as Microsoft Word. In fact, all you need to do is enter data. You do not have to concern yourself with the formatting of the document, since the formatting is already defined in the stylesheet that controls the Authentic View of the XML document. The stylesheet (StyleVision Power Stylesheet, shortened to SPS in this tutorial) is created by a stylesheet designer using Altova's StyleVision product.

Nanonull, Inc.

Location: US

<table>
<thead>
<tr>
<th>Street</th>
<th>Phone: +1 (321) 555 5155 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>119 Oakstreet, Suite 4876</td>
<td>Fax: +1 (321) 555 5155 4</td>
</tr>
<tr>
<td>City: Vereno</td>
<td>E-mail: <a href="mailto:office@nanonull.com">office@nanonull.com</a></td>
</tr>
<tr>
<td>State &amp; Zip: DC 29213</td>
<td></td>
</tr>
</tbody>
</table>

**Vereno Office Summary:** 4 departments, 15 employees.

The company was established in **Vereno in 1995** as a privately held software company. Since 1996, Nanonull has been actively involved in developing nanoelectronic software technologies. It released the first version of its acclaimed **NanoSoft Development Suite** in February 1999. Also in 1999, Nanonull increased its capital base with investment from a consortium of private investment firms. The company has been expanding rapidly ever since.

Editing an XML document in Authentic View involves two user actions: (i) editing the structure of the document (for example, adding or deleting document parts, such as paragraphs and headlines); and (ii) entering data (the content of document parts).

This tutorial takes you through the following steps:

- Opening an XML document in Authentic View. The key requirement for Authentic View editing is that the XML document be associated with an SPS file.
- A look at the Authentic View interface and a broad description of the central editing mechanisms.
- Editing document structure by inserting and deleting nodes.
- Entering data in the XML document.
- Entering (i) attribute values via the Attributes entry helper, and (ii) entity values.
- Printing the document.

Remember that this tutorial is intended to get you started, and has intentionally been kept simple. You will find additional reference material and feature descriptions in the Authentic View interface section.
Tutorial requirements

All the files you need for the tutorial are in the Examples folder of your Altova application folder. These files are:

- NanonullOrg.xml (the XML document you will open)
- NanonullOrg.sps (the StyleVision Power Stylesheet to which the XML document is linked)
- NanonullOrg.xsd (the XML Schema on which the XML document and StyleVision Power Stylesheet are based, and to which they are linked)
- nanonull.gif and Altova_right_300.gif (two image files used in the tutorial)

Please note: At some points in the tutorial, we ask you to look at the XML text of the XML document (as opposed to the Authentic View of the document). If the Altova product edition you are using does not include a Text View (as with Authentic Desktop and Authentic Browser), then use a plain text editor like Wordpad or Notepad to view the text of the XML document.

Caution: We recommend that you use a copy of NanonullOrg.xml for the tutorial, so that you can always retrieve the original should the need arise.

10.1.1 Opening an XML Document in Authentic View

In Authentic View, you can edit an existing XML document or create and edit a new XML document. In this tutorial, you will open an existing XML document in Authentic View (described in this section) and learn how you can edit it (subsequent sections). Additionally, in this section is a description of how a new XML document can be created for editing in Authentic View.

Opening an existing XML document

The file you will open is NanonullOrg.xml. It is in the Examples folder of your Altova application. You can open NanonullOrg.xml in one of two ways:

- Click File | Open in your Altova product, then browse for NanonullOrg.xml in the dialog that appears, and click Open.
- Use Windows Explorer to locate the file, right-click, and select your Altova product as the application with which to open the file.

The file NanonullOrg.xml opens directly in Authentic View (screenshot below). This is because

1. The file already has a StyleVision Power Stylesheet (SPS) assigned to it, and
2. In the Options dialog (Tools | Options), in the View tab, the option to open XML files in Authentic View if an SPS file is assigned has been checked. (Otherwise the file would open in Text View.)
Remember: It is the SPS that defines and controls how an XML document is displayed in Authentic View. Without an SPS, there can be no Authentic View of the document.

Creating a new XML document based on an SPS

You can also create a new XML document that is based on an SPS. You can do this in two ways: via the File | New menu command and via the Authentic | New Document menu command. In both cases an SPS is selected.

Via File | New

1. Select File | New, and, in the Create a New Document dialog, select XML as the new file type to create.
2. Click Select a STYLEVISION Stylesheet, and browse for the desired SPS.

Via Authentic | New Document

2. In the Create a New Document dialog, browse for the desired SPS.

If a Template XML File has been assigned to the SPS, then the data in the Template XML File is used as the starting data of the XML document template created in Authentic View.

10.1.2 The Authentic View Interface

The Authentic View editing interface consists of a main window in which you enter and edit the document data, and three entry helpers. Editing a document is simple. If you wish to see the markup of the document, switch on the markup tags. Then start typing in the content of your document. To modify the document structure, you can use either the context menu or the Elements entry helper.

Displaying XML node tags (document markup)

An XML document is essentially a hierarchy of nodes. For example:
By default, the node tags are not displayed in Authentic View. You can switch on the node tags by selecting the menu item **Authentic | Show Large Markup** (or the toolbar icon). Large markup tags contain the names of the respective nodes. Alternatively, you can select small markup (no node names in tags) and mixed markup (a mixture of large, small, and no markup tags, which is defined by the designer of the stylesheet; the default mixed markup for the document is no markup).

You can view the text of the XML document in the Text View of your Altova product or in a text editor.

**Entry helpers**

There are three entry helpers in the interface (*screenshot below*), located by default along the right edge of the application window. These are the Elements, Attributes, and Entity entry helpers.
Elements entry helper: The Elements entry helper displays elements that can be inserted and removed with reference to the current location of the cursor or selection in the Main Window. Note that the entry helper is context-sensitive; its content changes according to the location of the cursor or selection. The content of the entry helper can be changed in one other way: when another node is selected in the XML tree of the Elements entry helper, the elements relevant to that node are displayed in the entry helper. The Elements entry helper can be expanded to show the XML tree by checking the Show XML Tree check box at the top of the entry helper (see screenshot above). The XML tree shows the hierarchy of nodes from the top-level element node all the way down to the node selected in the Main Window.

Attributes entry helper: The Attributes entry helper displays the attributes of the element selected in the Main Window, and the values of these attributes. Attribute values can be entered or edited in the Attributes entry helper. Element nodes from the top-level element down to the selected element are available for selection in the combo box of the Attributes entry helper. Selecting an element from the dropdown list of the combo box causes that element's attributes to be displayed in the entry helper, where they can then be edited.

Entities entry helper: The Entities entry helper is not context-sensitive, and displays all the
entities declared for the document. Double-clicking an entity inserts it at the cursor location. How to add entities for a document is described in the section Authentic View interface.

Context menu
Right-clicking at a location in the Authentic View document pops up a context menu relevant to that (node) location. The context menu provides commands that enable you to:

- Insert nodes at that location or before or after the selected node. Submenus display lists of nodes that are allowed at the respective insert locations.
- Remove the selected node (if this allowed by the schema) or any removable ancestor element. The nodes that may be removed (according to the schema) are listed in a submenu.
- Insert entities and CDATA sections. The entities declared for the document are listed in a submenu. CDATA sections can only be inserted within text.
- Cut, copy, paste (including pasting as XML or text), and delete document content.

Note: For more details about the interface, see Authentic View interface

10.1.3 Node Operations
There are two major types of nodes you will encounter in an Authentic View XML document: element nodes and attribute nodes. These nodes are marked up with tags, which you can switch on. There are also other nodes in the document, such as text nodes (which are not marked up) and CDATA section nodes (which are marked up, in order to delimit them from surrounding text).

The node operations described in this section refer only to element nodes and attribute nodes. When trying out the operations described in this section, it is best to have large markup switched on.

Note: It is important to remember that only same-or higher-level elements can be inserted before or after the selected element. Same-level elements are siblings. Siblings of a paragraph element would be other paragraph elements, but could also be lists, a table, an image, etc. Siblings could occur before or after an element. Higher-level elements are ancestor elements and siblings of ancestors. For a paragraph element, ancestor elements could be a section, chapter, article, etc. A paragraph in a valid XML file would already have ancestors. Therefore, adding a higher-level element in Authentic View, creates the new element as a sibling of the relevant ancestor. For example, if a section element is inserted after a paragraph, it is created as a sibling of the section that contains the current paragraph element.

Carrying out node operations
Node operations can be carried out by selecting a command in the context menu or by clicking the node operation entry in the Elements entry helper. In some cases, an element or attribute can be added by clicking the Add Node link in the Authentic View of the document. In the special cases of elements defined as paragraphs or list items, pressing the Enter key when within such an element creates a new sibling element of that kind. This section also describes how nodes can be created and deleted by using the Apply Element, Remove Node, and Clear Element mechanisms.
Inserting elements

Elements can be inserted at the following locations:

- The cursor location within an element node. The elements available for insertion at that location are listed in a submenu of the context menu's Insert command. In the Elements entry helper, elements that can be inserted at a location are indicated with the icon. In the NanonullOrg.xml document, place the cursor inside the para element, and create bold and italic elements using both the context menu and Elements entry helper.

- Before or after the selected element or any of its ancestors, if allowed by the schema. Select the required element from the submenu/s that roll out. In the Elements entry helper, elements that can be inserted before or after the selected element are indicated with the and icons, respectively. Note that in the Elements entry helper, you can insert elements before/after the selected element only; you cannot insert before/after an ancestor element. Try out this command, by first placing the cursor inside the para element and then inside the table listing the employees.

Add Node link

If an element or attribute is included in the document design, and is not present in the XML document, an Add Node link is displayed at the location in the document where that node is specified. To see this link, in the line with the text, Location of logo, select the @href node within the CompanyLogo element and delete it (by pressing the Delete key). The add @href link appears within the CompanyLogo element that was edited (screenshot below). Clicking the link adds the @href node to the XML document. The text box within the @href tags appears because the design specifies that the @href node be added like this. You still have to enter the value (or content) of the @href node. Enter the text nanonull.gif.

If the content model of an element is ambiguous, for example, if it specifies that a sequence of child elements may appear in any order, then the add... link appears. Note that no node name is specified. Clicking the link will pop up a list of elements that may validly be inserted.

Note: The Add Node link appears directly in the document template; there is no corresponding entry in the context menu or Elements entry helper.

Creating new elements with the Enter key

In cases where an element has been formatted as a paragraph or list item (by the stylesheet designer), pressing the Enter key when inside such a node causes a new node of that kind to be
inserted after the current node. You can try this mechanism in the NanonullOrg.xml document by going to the end of a para node (just before its end tag) and pressing Enter.

Applying elements
In elements of mixed content (those which contain both text and child elements), some text content can be selected and an allowed child element be applied to it. The selected text becomes the content of the applied element. To apply elements, in the context menu, select Apply and then select from among the applicable elements. (If no elements can be applied to the selected text, then the Apply command does not appear in the context menu.) In the Elements entry helper, elements that can be applied for a selection are indicated with the icon. In the NanonullOrg.xml document, select text inside the mixed content para element and experiment with applying the bold and italic elements.

The stylesheet designer might also have created a toolbar icon to apply an element. In the NanonullOrg.xml document, the bold and italic elements can be applied by clicking the bold and italic icons in the application's Authentic toolbar.

Removing nodes
A node can be removed if its removal does not render the document invalid. Removing a node causes a node and all its contents to be deleted. A node can be removed using the Remove command in the context menu. When the Remove command is highlighted, a submenu pops up which contains all nodes that may be removed, starting from the selected node and going up to the document's top-level node. To select a node for removal, the cursor can be placed within the node, or the node (or part of it) can be highlighted. In the Elements entry helper, nodes that can be removed are indicated with the icon. A removable node can also be removed by selecting it and pressing the Delete key. In the NanonullOrg.xml document, experiment with removing a few nodes using the mechanisms described. You can undo your changes with Ctrl+Z.

Clearing elements
Element nodes that are children of elements with mixed content (both text and element children) can be cleared. The entire element can be cleared when the node is selected or when the cursor is placed inside the node as an insertion point. A text fragment within the element can be cleared of the element markup by highlighting the text fragment. With the selection made, select Clear in the context menu and then the element to clear. In the Elements entry helper, elements that can be cleared for a particular selection are indicated with the icon (insertion point selection) and icon (range selection). In the NanonullOrg.xml document, try the clearing mechanism with the bold and italic child elements of para (which has mixed content).

Tables and table structure
There are two types of Authentic View table:

- **SPS tables (static and dynamic).** The broad structure of SPS table is determined by the stylesheet designer. Within this broad structure, the only structural changes you are allowed are content-driven. For example, you could add new rows to a dynamic SPS table.
- **XML tables,** in which you decide to present the contents of a particular node (say, one for person-specific details) as a table. If the stylesheet designer has enabled the creation of this node as an XML table, then you can determine the structure of the table and edit its
contents. XML tables are discussed in detail in the Tables in Authentic View section.

10.1.4 Entering Data in Authentic View

Data is entered into the XML document directly in the main window of Authentic View. Additionally for attributes, data (the value of the attribute) can be entered in the Attributes entry helper. Data is entered (i) directly as text, or (ii) by selecting an option in a data-entry device, which is then mapped to a predefined text entry.

Adding text content

You can enter element content and attribute values directly as text in the main window of Authentic View. To insert content, place the cursor at the location where you want to insert the text, and type. You can also copy text from the clipboard into the document. Content can also be edited using standard editing mechanisms, such as the Caps and Delete keys. For example, you can highlight the text to be edited and type in the replacement text with the Caps key on.

For example, to change the name of the company, in the Name field of Office, place the cursor after Nanonull, and type in USA to change the name from Nanonull, Inc. to Nanonull USA, Inc.

If text is editable, you will be able to place your cursor in it and highlight it, otherwise you will not be able to. Try changing any of the field names (not the field values), such as "Street", "City", or "State/Zip," in the address block. You are not able to place the cursor in this text because such text is not XML content; it is derived from the StyleVision Power Stylesheet.

Inserting special characters and entities

When entering data, the following type of content is handled in a special way:

- **Special characters that are used for XML markup** (ampersand, apostrophe, greater than, less than, and quotes). These characters are available as built-in entities and can be entered in the document by double-clicking the respective entity in the Entities entry helper. If these characters occur frequently (for example, in program code listings), then they can be entered within CDATA sections. To insert a CDATA section, right-click at the location where you wish to enter the CDATA section, and select Insert CDATA Section from the context menu. The XML processor ignores all markup characters within CDATA sections. This also means that if you want a special character inside a CDATA section, you should enter that character and not its entity reference.

- **Special characters that cannot be entered via the keyboard** should be entered by copying them from the character map of your system to the required location in the document.

- **A frequently used text string** can be defined as an entity, which appears in the Entities entry helper. The entity is inserted at the required locations by placing the cursor at each required location and double-clicking the entity in the entry helper. This is useful for maintenance because the value of the text string is held in one location; if the value needs to be changed, then all that needs to be done is to change the entity definition.
Note: When markup is hidden in Authentic View, an empty element can easily be overlooked. To make sure that you are not overlooking an empty element, switch large or small markup on.

Try using each type of text content described above.

Adding content via a data-entry device

In the content editing you have learned above, content is added by directly typing in text as content. There is one other way that element content (or attribute values) can be entered in Authentic View: via data-entry devices.

Given below is a list of data-entry devices in Authentic View, together with an explanation of how data is entered in the XML file for each device.

<table>
<thead>
<tr>
<th>Data-Entry Device</th>
<th>Data in XML File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Field (Text Box)</td>
<td>Text entered by user</td>
</tr>
<tr>
<td>Multiline Input Field</td>
<td>Text entered by user</td>
</tr>
<tr>
<td>Combo box</td>
<td>User selection mapped to value</td>
</tr>
<tr>
<td>Check box</td>
<td>User selection mapped to value</td>
</tr>
<tr>
<td>Radio button</td>
<td>User selection mapped to value</td>
</tr>
<tr>
<td>Button</td>
<td>User selection mapped to value</td>
</tr>
</tbody>
</table>

In the static table containing the address fields (shown below), there are two data-entry devices: an input field for the Zip field and a combo-box for the State field. The values that you enter in the text fields are entered directly as the XML content of the respective elements. For other data-entry devices, your selection is mapped to a value.

Vereno Office Summary

- Apartments, 15 employees.

For the Authentic View shown above, here is the corresponding XML text:

```
<Address>
  <ipo:street>119 Oakstreet, Suite 4876</ipo:street>
  <ipo:city>Vereno</ipo:city>
  <ipo:state>DC</ipo:state>
  <ipo:zip>29213</ipo:zip>
</Address>
```

Notice that the combo-box selection DC is mapped to a value of DC. The value of the Zip field is
entered directly as content of the `ipo:zip` element.

### 10.1.5 Entering Attribute Values

An attribute is a property of an element, and an element can have any number of attributes. Attributes have values. You may sometimes be required to enter XML data as an attribute value. In Authentic View, you enter attribute values in two ways:

- As content in the main window if the attribute has been created to accept its value in this way
- In the Attributes entry helper

#### Attribute values in the main window

Attribute values can be entered as normal text or as text in an input field, or as a user selection that will be mapped to an XML value. They are entered in the same way that element content is entered: see Entering Data in Authentic View. In such cases, the distinction between element content and attribute value is made by the StyleVision Power Stylesheet and the data is handled appropriately.

#### Attribute values in the Attributes Entry Helper

If you wish to enter or change an attribute value, you can also do this in the Attributes Entry Helper. First, the attribute node is selected in Authentic View, then the value of the attribute is entered or edited in the Attributes entry helper. In the `NanonullOrg.xml` document, the location of the logo is stored as the value of the `href` attribute of the `CompanyLogo` element. To change the logo to be used:

1. Select the `CompanyLogo` element by clicking a `CompanyLogo` tag. The attributes of the `CompanyLogo` element are displayed in the Attributes entry helper.
2. In the Attributes Entry Helper, change the value of the `href` attribute from `nanonull.gif` to `Altova_right_300.gif` (an image in the Examples folder).

   ![Attributes Entry Helper](image)

This causes the Nanonull logo to be replaced by the Altova logo.

**Note:** Entities cannot be entered in the Attributes entry helper.

### 10.1.6 Adding Entities

An entity in Authentic View is typically XML data (but not necessarily), such as a single character; a text string; and even a fragment of an XML document. An entity can also be a binary file, such as an image file. All the entities available for a particular document are displayed in the Entities Entry Helper (screenshot below). To insert an entity, place the cursor at the location in
the document where you want to insert it, and then double-click the entity in the Entities entry helper. Note that you cannot enter entities in the Attributes entry helper.

The ampersand character (&) has special significance in XML (as have the apostrophe, less than and greater than symbols, and the double quote). To insert these characters, entities are used so that they are not confused with XML-significant characters. These characters are available as entities in Authentic View.

In NanonullOrg.xml, change the title of Joe Martin (in Marketing) to Marketing Manager Europe & Asia. Do this as follows:

1. Place the cursor where the ampersand is to be inserted.
2. Double-click the entity listed as "amp". This inserts an ampersand (screenshot below).

![Entities Entry Helper](image)

Note: The Entities Entry Helper is not context-sensitive. All available entities are displayed no matter where the cursor is positioned. This does not mean that an entity can be inserted at all locations in the document. If you are not sure, then validate the document after inserting the entity: XML | Validate XML (F8).

Defining your own entities

As a document editor, you can define your own document entities. How to do this is described in the section Defining Entities in Authentic View.

10.1.7 Printing the Document

A printout from Authentic View of an XML document preserves the formatting seen in Authentic View.

To print NanonullOrg.xml, do the following:

1. Switch to Hide Markup mode if you are not already in it. You must do this if you do not want markup to be printed.
2. Select File | Print Preview to see a preview of all pages. Shown below is part of a print
Notice that the formatting of the page is the same as that in Authentic View.

3. To print the file, click **File | Print**.

Note that you can also print a version of the document that displays markup. To do this, switch Authentic View to Show small markup mode or Show large markup mode, and then print.
10.2 Authentic View Interface

Authentic View is enabled by clicking the Authentic tab of the active document. If no SPS has been assigned to the XML document, you are prompted to assign one. You can assign an SPS at any time via the Authentic | Assign a Stylevision Stylesheet command.

This section provides:

- An overview of the interface
- A description of the toolbar icons specific to Authentic View
- A description of viewing modes available in the main Authentic View window
- A description of the Entry Helpers and how they are to be used
- A description of the context menus available at various points in the Authentic View of the XML document

Additional sources of Authentic View information are:

- An Authentic View Tutorial, which shows you how to use the Authentic View interface. This tutorial is available in the documentation of the Altova XMLSpy and Altova Authentic Desktop products (see the Tutorials section), as well as online.
- For a detailed description of Authentic View menu commands, see the User Reference section of your product documentation.

Altova website: 🌐 XML content editing, XML authoring

10.2.1 Overview of the GUI

Authentic View has a menu bar and toolbar running across the top of the window, and three areas that cover the rest of the interface: the Project Window, Main Window, and Entry Helpers Window. These areas are shown below.
Menu bar
The menus available in the menu bar are described in detail in the User Reference section of your product documentation.

Toolbar
The symbols and icons displayed in the toolbar are described in the section, Authentic View toolbar icons.

Project window
You can group XML, XSL, XML schema, and Entity files together in a project. To create and modify the list of project files, use the commands in the Project menu (described in the User Reference section of your product documentation). The list of project files is displayed in the Project window. A file in the Project window can be accessed by double-clicking it.

Main window
This is the window in which the XML document is displayed and edited. It is described in the section, Authentic View main window.

Entry helpers
There are three entry helper windows in this area: Elements, Attributes, and Entities. What entries appear in these windows (Elements and Attributes Entry Helpers) are context-sensitive, i.e. it depends on where in the document the cursor is. You can enter an element or entity into the document by double-clicking its entry helper. The value of an attribute is entered into the value field of that attribute in the Attributes Entry Helper. See the section Authentic View Entry Helpers for details.

Status Bar
The Status Bar displays the XPath to the currently selected node.

Context menus
These are the menus that appear when you right-click in the Main Window. The available commands are context-sensitive editing commands, i.e. they allow you to manipulate structure and content relevant to the selected node. Such manipulations include inserting, appending, or deleting a node, adding entities, or cutting and pasting content.

10.2.2 Authentic View Toolbar Icons
Icons in the Authentic View toolbar are command shortcuts. Some icons will already be familiar to you from other Windows applications or Altova products, others might be new to you. This section describes icons unique to Authentic View. In the description below, related icons are grouped together.

Show/hide XML markup
In Authentic View, the tags for all, some, or none of the XML elements or attributes can be displayed, either with their names (large markup) or without names (small markup). The four
markup icons appear in the toolbar, and the corresponding commands are available in the 

**Authentic** menu.

Hide markup. All XML tags are hidden except those which have been collapsed. Double-clicking on a collapsed tag (which is the usual way to expand it) in Hide markup mode will cause the node's content to be displayed and the tags to be hidden.

Show small markup. XML element/attribute tags are shown without names.

Show large markup. XML element/attribute tags are shown with names.

Show mixed markup. In the StyleVision Power Stylesheet, each XML element or attribute can be specified to display (as either large or small markup), or not to display at all. This is called mixed markup mode since some elements can be specified to be displayed with markup and some without markup. In mixed markup mode, therefore, the Authentic View user sees a customized markup. Note, however, that this customization is created by the person who has designed the StyleVision Power Stylesheet. It cannot be defined by the Authentic View user.

**Editing dynamic table structures**

Rows in a **dynamic SPS table** are repetitions of a data structure. Each row represents an occurrence of a single element. Each row, therefore, has the same XML substructure as the next.

The dynamic table editing commands manipulate the rows of a dynamic SPS table. That is, you can modify the number and order of the element occurrences. You cannot, however, edit the columns of a dynamic SPS table, since this would entail changing the substructure of individual element occurrences.

The icons for dynamic table editing commands appear in the toolbar, and are also available in the 

**Authentic** menu.

Append row to table

Insert row in table

Duplicate current table row (i.e. cell contents are duplicated)

Move current row up by one row

Move current row down by one row
Delete the current row

**Please note:** These commands apply only to **dynamic SPS tables**. They should not be used inside static SPS tables. The various types of tables used in Authentic View are described in the [Using Tables in Authentic View](#) section of this documentation.

Creating and editing XML tables

You can insert your own tables should you want to present your data as a table. Such tables are inserted as XML tables. You can modify the structure of an XML table, and format the table. The icons for creating and editing XML tables are available in the toolbar, and are shown below. They are described in the section [XML table editing icons](#).

![XML table editing icons](image)

The commands corresponding to these icons are **not available as menu items**. Note also that for you to be able to use XML tables, this function must be enabled and suitably configured in the StyleVision Power Stylesheet.

A detailed description of the types of tables used in Authentic View and of how XML tables are to be created and edited is given in [Using Tables in Authentic View](#).

Text formatting icons

Text in Authentic View is formatted by applying to it an XML element or attribute that has the required formatting. If such formatting has been defined, the designer of the StyleVision Power Stylesheet can provide icons in the Authentic View toolbar to apply the formatting. To apply text formatting using a text formatting icon, highlight the text you want to format, and click the appropriate icon.

DB Row Navigation icons

The arrow icons are, from left to right, Go to First Record in the DB; Go to Previous Record; Open Go to Record # dialog; Go to Next Record; and Go to Last Record.

This icon opens the Edit Database Query dialog in which you can enter a query. Authentic View displays the queried record/s.

XML database editing

The [Select New Row with XML Data for Editing](#) command enables you to select a new row from the relevant table in an XML DB, such as IBM DB2. This row appears in Authentic View, can be edited there, and then saved back to the DB.

Portable XML Form (PXF) toolbar buttons

The following PXF toolbar buttons are available in the Authentic View of XMLSpy and Authentic...
Desktop:

Clicking the individual buttons generates HTML, RTF, PDF, and/or DocX output.

These buttons are enabled when a PXF file is opened in Authentic View. Individual buttons are enabled if the PXF file was configured to contain the XSLT stylesheet for that specific output format. For example, if the PXF file was configured to contain the XSLT stylesheets for HTML and RTF, then only the toolbar buttons for HTML and RTF output will be enabled while those for PDF and DocX (Word 2007+) output will be disabled.

10.2.3 Authentic View Main Window

There are four viewing modes in Authentic View: Large Markup; Small Markup; Mixed Markup; and Hide All Markup. These modes enable you to view the document with varying levels of markup information. To switch between modes, use the commands in the Authentic menu or the icons in the toolbar (see the previous section, Authentic View toolbar icons).

Large markup

This shows the start and end tags of elements and attributes with the element/attribute names in the tags:

The element Name in the figure above is expanded, i.e. the start and end tags, as well as the content of the element, are shown. An element/attribute can be contracted by double-clicking either its start or end tag. To expand the contracted element/attribute, double-click the contracted tag.

In large markup, attributes are recognized by the equals-to symbol in the start and end tags of the attribute:

Small markup

This shows the start and end tags of elements/attributes without names:
Notice that start tags have a symbol inside it while end tags are empty. Also, element tags have an angular-brackets symbol while attribute tags have and equals sign as its symbol (see screenshot below).

To collapse or expand an element/attribute, double-click the appropriate tag. The example below shows a collapsed element (highlighted in blue). Notice the shape of the tag of the collapsed element and that of the start tag of the expanded element to its left.

Mixed markup
Mixed markup shows a customized level of markup. The person who has designed the StyleVision Power Stylesheet can specify either large markup, small markup, or no markup for individual elements/attributes in the document. The Authentic View user sees this customized markup in mixed markup viewing mode.

Hide all markup
All XML markup is hidden. Since the formatting seen in Authentic View is the formatting of the printed document, this viewing mode is a WYSIWYG view of the document.
Content display

In Authentic View, content is displayed in two ways:

- Plain text. You type in the text, and this text becomes the content of the element or the value of the attribute.

- Data-entry devices. The display contains either an input field (text box), a multiline input field, combo box, check box, or radio button. In the case of input fields and multiline input fields, the text you enter in the field becomes the XML content of the element or the value of the attribute.

In the case of the other data-entry devices, your selection produces a corresponding XML value, which is specified in the StyleVision Power Stylesheet. Thus, in a combo box, a selection of, say, "approved" (which would be available in the dropdown list of the combo box) could map to an XML value of "1", or to "approved", or anything else; while "not approved" could map to "0", or "not approved", or anything else.

Optional nodes

When an element or attribute is optional (according to the referenced schema), a prompt of type `add [element/attribute]` is displayed:

Clicking the prompt adds the element, and places the cursor for data entry. If there are multiple optional nodes, the prompt `add...` is displayed. Clicking the prompt displays a menu of the optional nodes.

10.2.4 Authentic View Entry Helpers

There are three entry helpers in Authentic View: for Elements, Attributes, and Entities. They are displayed as windows down the right side of the Authentic View interface (see screenshot below).
The Elements and Attributes Entry Helpers are context-sensitive, i.e. what appears in the entry helper depends on where the cursor is in the document. The entities displayed in the Entities Entry Helper are not context-sensitive; all entities allowed for the document are displayed no matter where the cursor is.

Each of the entry helpers is described separately below.

**Elements Entry Helper**

The Elements Entry Helper consists of two parts:

- The upper part, containing an XML tree that can be toggled on and off using the Show XML tree check box. The XML tree shows the ancestors up to the document's root element for the current element. When you click on an element in the XML tree, elements corresponding to that element (as described in the next item in this list) appear in the lower part of the Elements Entry Helper.
- The lower part, containing a list of the nodes that can be inserted within, before, and after; removed; applied to or cleared from the selected element or text range in Authentic View. What you can do with an element listed in the Entry Helper is indicated by the icon to the
left of the element name in the Entry Helper. The icons that occur in the Elements Entry Helper are listed below, together with an explanation of what they mean.

To use node from the Entry Helper, click its icon.

**Insert After Element**
The element in the Entry Helper is inserted after the selected element. Note that it is appended at the correct hierarchical level. For example, if your cursor is inside a \texttt{//sect1/para} element, and you append a \texttt{sect1} element, then the new \texttt{sect1} element will be appended not as a following sibling of \texttt{//sect1/para} but as a following sibling of the \texttt{sect1} element that is the parent of that \texttt{para} element.

**Insert Before Element**
The element in the Entry Helper is inserted before the selected element. Note that, just as with the Insert After Element command, the element is inserted at the correct hierarchical level.

**Remove Element**
Removes the element and its content.

**Insert Element**
An element from the Entry Helper can also be inserted within an element. When the cursor is placed within an element, then the allowed child elements of that element can be inserted. Note that allowed child elements can be part of an elements-only content model as well as a mixed content model (text plus child elements).

An allowed child element can be inserted either when a text range is selected or when the cursor is placed as an insertion point within the text.

- When a text range is selected and an element inserted, the text range becomes the content of the inserted element.
- When an element is inserted at an insertion point, the element is inserted at that point.

After an element has been inserted, it can be cleared by clicking either of the two Clear Element icons that appear (in the Elements Entry Helper) for these inline elements. Which of the two icons appears depends on whether you select a text range or place the cursor in the text as an insertion point (see below).

**Apply Element**
If you select an element in your document (by clicking either its start or end tag in the Show large markup view) and that element can be replaced by another element (for example, in a mixed content element such as \texttt{para}, an \texttt{italic} element can be replaced by the \texttt{bold} element), this icon indicates that the element in the Entry Helper can be applied to the selected (original) element. The Apply Element command can also be applied to a text range within an element of mixed content; the text range will be created as content of the applied element.
• If the applied element has a child element with the same name as a child of the original element and an instance of this child element exists in the original element, then the child element of the original is retained in the new element's content.

• If the applied element has no child element with the same name as that of an instantiated child of the original element, then the instantiated child of the original element is appended as a sibling of any child element or elements that the new element may have.

• If the applied element has a child element for which no equivalent exists in the original element's content model, then this child element is not created directly but Authentic View offers you the option of inserting it.

If a text range is selected rather than an element, applying an element to the selection will create the applied element at that location with the selected text range as its content. Applying an element when the cursor is an insertion point is not allowed.

**Clear Element (when range selected)**
This icon appears when text within an element of mixed content is selected. Clicking the icon clears the element from around the selected text range.

**Clear Element (when insertion point selected)**
This icon appears when the cursor is placed within an element that is a child of a mixed-content element. Clicking the icon clears the inline element.

**Attributes Entry Helper**
The Attributes Entry Helper consists of a drop-down combo box and a list of attributes. The element that you have selected (you can click the start or end tag, or place the cursor anywhere in the element content to select it) appears in the combo box.

The Attributes Entry Helper shown in the figures below has a `para` element in the combo box. Clicking the arrow in the combo box drops down a list of all the `para` element's ancestors up to the document's root element, which in this case is OrgChart.

Below the combo box, a list of valid attributes for that element is displayed, in this case for `para`. If an attribute is mandatory on a given element, then it appears in bold. (In the example below, there are no mandatory attributes except the built-in attribute `xsi:type`.)
To enter a value for an attribute, click in the value field of the attribute and enter the value. This creates the attribute and its value in the XML document.

In the case of the xsi:nil attribute, which appears in the Attributes Entry Helper when a nillable element has been selected, the value of the xsi:nil attribute can only be entered by selecting one of the allowed values (true or false) from the dropdown list for the attribute's value.

The xsi:type attribute can be changed by clicking in the value field of the attribute and then selecting, from the dropdown list that appears, one of the listed values. The listed values are the available abstract types defined in the XML Schema on which the Authentic View document is based.

### Entities Entry Helper

The Entities Entry Helper allows you to insert an entity in your document. Entities can be used to insert special characters or text fragments that occur often in a document (such as the name of a company). To insert an entity, place the cursor at the point in the text where you want to have the entity inserted, then double-click the entity in the Entities Entry Helper.

Note: An internal entity is one that has its value defined within the DTD. An external entity is one that has its value contained in an external source, e.g. another XML file. Both internal and external entities are listed in the Entities Entry Helper. When you insert an entity, whether internal or external, the entity—not its value—is inserted into the XML text. If the entity is an internal entity, Authentic View displays the value of the entity. If the entity is an external entity, Authentic View displays the entity—and not its value. This means, for example, that an XML file that is an external entity will be shown in the Authentic View display as an entity; its content does not replace the entity in the Authentic View display.

You can also define your own entities in Authentic View and these will also be displayed in the entry helper: see Define Entities in the Editing in Authentic View section.
10.2.5 Authentic View Context Menus

Right-clicking on some selected document content or node pops up a context menu with commands relevant to the selection or cursor location.

Inserting elements

The figure below shows the **Insert** submenu, which is a list of all elements that can be inserted at that current cursor location. The **Insert Before** submenu lists all elements that can be inserted before the current element. The **Insert After** submenu lists all elements that can be inserted after the current element. In the figure below, the current element is the `para` element. The **bold** and **italic** elements can be inserted within the current `para` element.

As can be seen below, the `para` and `Office` elements can be inserted before the current `para` element.

The node insertion, replacement (**Apply**), and markup removal (**Clear**) commands that are available in the context menu are also available in the Authentic View entry helpers and are fully described in that section.

Insert entity

Positioning the cursor over the Insert Entity command rolls out a submenu containing a list of all declared entities. Clicking an entity inserts it at the selection. See Define Entities for a description of how to define entities for the document.

Insert CDATA Section

This command is enabled when the cursor is placed within text. Clicking it inserts a CDATA section at the cursor insertion point. The CDATA section is delimited by start and end tags; to see these tags you should switch on large or small markup. Within CDATA sections, XML markup and parsing is ignored. XML markup characters (the ampersand, apostrophe, greater than, less than, and quote characters) are not treated as markup, but as literals. So CDATA sections are useful for text such as program code listings, which have XML markup characters.

Remove node

Positioning the mouse cursor over the **Remove** command pops up a menu list consisting of the selected node and all its removable ancestors (those that would not invalidate the document) up to the document element. Click the element to be removed. This is a quick way to delete an element or any removable ancestor. Note that clicking an ancestor element will remove all its descendants, including the selected element.
Clear
The **Clear** command clears the element markup from around the selection. If the entire node is selected, then the element markup is cleared for the entire node. If a text segment is selected, then the element markup is cleared from around that text segment only.

Apply
The **Apply** command applies a selected element to your selection in the main Window. For more details, see [Authentic View entry helpers](#).

Copy, Cut, Paste
These are the standard Windows commands. Note, however, that the **Paste** command pastes copied text either as XML or as Text, depending on what the designer of the stylesheet has specified for the SPS as a whole. For information about how the **Copy as XML** and **Copy as Text** commands work, see the description of the **Paste As** command immediately below.

Paste As
The **Paste As** command offers the option of pasting as XML or as text an Authentic View XML fragment (which was copied to the clipboard). If the copied fragment is pasted as XML it is pasted together with its XML markup. If it is pasted as text, then only the text content of the copied fragment is pasted (not the XML markup, if any). The following situations are possible:

- An **entire node together with its markup tags** is highlighted in Authentic View and copied to the clipboard. (i) The node can be pasted as XML to any location where this node may validly be placed. It will not be pasted to an invalid location. (ii) If the node is pasted as text, then only the node's **text content** will be pasted (not the markup); the text content can be pasted to any location in the XML document where text may be pasted.
- A **text fragment** is highlighted in Authentic View and copied to the clipboard. (i) If this fragment is pasted as XML, then the XML markup tags of the text—even though these were not explicitly copied with the text fragment—will be pasted along with the text, but only if the XML node is valid at the location where the fragment is pasted. (ii) If the fragment is pasted as text, then it can be pasted to any location in the XML document where text may be pasted.

**Note:** Text will be copied to nodes where text is allowed, so it is up to you to ensure that the copied text does not invalidate the document. The copied text should therefore be:

- (i) lexically valid in the new location (for example, non-numeric characters in a numeric node would be invalid), and
- (ii) not otherwise invalidate the node (for example, four digits in a node that accepts only three-digit numbers would invalidate the node).

If the pasted text does in any way invalidate the document, this will be indicated by the text being displayed in red.

Delete
The **Delete** command removes the selected node and its contents. A node is considered to be selected for this purpose by placing the cursor within the the node or by clicking either the start or
end tag of the node.
10.3 Editing in Authentic View

This section describes important features of Authentic View in detail. Features have been included in this section either because they are frequently used or because the mechanisms or concepts involved require explanation.

The section explains the following:

- There are three distinct types of tables used in Authentic View. The section Using tables in Authentic View explains the three types of tables (static SPS, dynamic SPS, and XML), and when and how to use them. It starts with the broad, conceptual picture and moves to the details of usage.
- The Date Picker is a graphical calendar that enters dates in the correct XML format when you click a date. See Date Picker.
- An entity is shorthand for a special character or text string. You can define your own entities, which allows you to insert these special characters or text strings by inserting the corresponding entities. See Defining Entities for details.
- In the Enterprise and Professional editions of Altova products, Authentic View users can sign XML documents with digital XML signatures and verify these signatures.
- What image formats can be displayed in Authentic View.

Altova website: XML content editing, XML authoring

10.3.1 Basic Editing

When you edit in Authentic View, you are editing an XML document. Authentic View, however, can hide the structural XML markup of the document, thus displaying only the content of the document (first screenshot below). You are therefore not exposed to the technicalities of XML, and can edit the document as you would a normal text document. If you wish, you could switch on the markup at any time while editing (second screenshot below).

Vereno Office Summary: 4 departments, 16 employees.

The company was established in Vereno in 1995 as a privately held software company. Since 1996, Nanonull has been actively involved in developing nanoelectronic software technologies. It released the first version of its acclaimed NanoSoft Development Suite in February 1999. Also in 1999, Nanonull increased its capital base with investment from a consortium of private investment firms. The company has been expanding rapidly ever since.

An editable Authentic View document with no XML markup.
Inserting nodes

Very often you will need to add a new node to the Authentic XML document. For example, a new `Person` element might need to be added to an address book type of document. In such cases the XML Schema would allow the addition of the new element. All you need to do is right-click the node in the Authentic View document before which or after which you wish to add the new node. In the context menu that appears, select **Insert Before** or **Insert After** as required. The nodes available for insertion at that point in the document are listed in a submenu. Click the required node to insert it. The node will be inserted. All mandatory descendant nodes are also inserted. If a descendant node is optional, a clickable link, **Add NodeName**, appears to enable you to add the optional node if you wish to.

If the node being added is an element with an abstract type, then a dialog (*something like in the screenshot below*) appears containing a list of derived types that are available in the XML Schema.
The screenshot above pops up when a Publication element is added. The Publication element is of type PublicationType, which is an abstract complex type. The two complex types BookType and MagazineType are derived from the abstract PublicationType. Therefore, when a Publication element is added to the XML document, one of these two concrete types derived from Publication's abstract type must be specified. The new Publication element will be added with an xsi:type attribute:

```xml
<Publication xsi:type="BookType"> ... </Publication>
<Publication xsi:type="MagazineType"> ... </Publication>
... 
<Publication xsi:type="MagazineType"> ... </Publication>
```

Selecting one of the available derived types and clicking OK does the following:

- Sets the selected derived type as the value of the xsi:type attribute of the element
- Inserts the element together with the descendant nodes defined in the content model of the selected derived type.

The selected derived type can be changed subsequently by changing the value of the element's xsi:type attribute in the Attributes Entry Helper. When the element's type is changed in this way, all nodes of the previous type's content model are removed and nodes of the new type's content model are inserted.

**Text editing**

An Authentic View document will essentially consist of text and images. To edit the text in the document, place the cursor at the location where you wish to insert text, and type. You can copy, move, and delete text using familiar keystrokes (such as the Delete key) and drag-and-drop mechanisms. One exception is the Enter key. Since the Authentic View document is pre-formatted, you do not—and cannot—add extra lines or space between items. The Enter key in Authentic View therefore serves to append another instance of the element currently being edited,
and should be used exclusively for this purpose.

Copy as XML or as text
Text can be copied and pasted as XML or as text.

- If text is pasted as XML, then the XML markup is pasted together with the text content of nodes. The XML markup is pasted even if only part of a node's contents has been copied. For the markup to be pasted it must be allowed, according to the schema, at the location where it is pasted.
- If text is pasted as text, XML markup is not pasted.

To paste as XML or text, first copy the text (Ctrl+C), right-click at the location where the text is to be pasted, and select the context menu command Paste As | XML or Paste As | Text. If the shortcut Ctrl+V is used, the text will be pasted in the default Paste Mode of the SPS. The default Paste Mode will have been specified by the designer of the SPS. For more details, see the section Context Menus.

Alternatively, highlighted text can be dragged to the location where it is to be pasted. When the text is dropped, a pop-up appears asking whether the text is to be pasted as text or XML. Select the desired option.

Text formatting
A fundamental principle of XML document systems is that content be kept separate from presentation. The XML document contains the content, while the stylesheet contains the presentation (formatting). In Authentic View, the XML document is presented via the stylesheet. This means that all the formatting you see in Authentic View is produced by the stylesheet. If you see bold text, that bold formatting has been provided by the stylesheet. If you see a list or a table, that list format or table format has been provided by the stylesheet. The XML document, which you edit in Authentic View contains only the content; it contains no formatting whatsoever. The formatting is contained in the stylesheet. What this means for you, the Authentic View user, is that you do not have to—nor can you—format any of the text you edit. You are editing content. The formatting that is automatically applied to the content you edit is linked to the semantic and/or structural value of the data you are editing. For example, an email address (which could be considered a semantic unit) will be formatted automatically in a certain way because it is an email. In the same way, a headline must occur at a particular location in the document (both a structural and semantic unit) and will be formatted automatically in the way the stylesheet designer has specified that headlines be formatted. You cannot change the formatting of either email address or headline. All that you do is edit the content of the email address or headline.

In some cases, content might need to be specially presented; for example, a text string that must be presented in boldface. In all such cases, the presentation must be tied in with a structural element of the document. For example, a text string that must be presented in boldface, will be structurally separated from surrounding content by markup that the stylesheet designer will format in boldface. If you, as the Authentic View user, need to use such a text string, you would need to enclose the text string within the appropriate element markup. For information about how to do this, see the Insert Element command in the Elements Entry Helper section of the documentation.

Using RichEdit in Authentic View
In Authentic View, when the cursor is placed inside an element that has been created as a
RichEdit component, the buttons and controls in the RichEdit toolbar (screenshot below) become enabled. Otherwise they are grayed out.

![RichEdit](image)

Select the text you wish to style and specify the styling you wish to apply via the buttons and controls of the RichEdit toolbar. RichEdit enables the Authentic View user to specify the font, font-weight, font-style, font-decoration, font-size, color, background color and alignment of text. The text that has been styled will be enclosed in the tags of the styling element.

**Inserting entities**

In XML documents, some characters are reserved for markup and cannot be used in normal text. These are the ampersand (&), apostrophe (‘), less than (<), greater than (>), and quote (”) characters. If you wish to use these characters in your data, you must insert them as entity references, via the **Entities Entry Helper** (screenshot below).

![Entities](image)

XML also offers the opportunity to create custom entities. These could be: (i) special characters that are not available on your keyboard, (ii) text strings that you wish to re-use in your document content, (iii) XML data fragments, or (iv) other resources, such as images. You can define your own entities within the Authentic View application. Once defined, these entities appear in the **Entities Entry Helper** and can then be inserted as in the document.

**Inserting CDATA sections**

CDATA sections are sections of text in an XML document that the XML parser does not process as XML data. They can be used to escape large sections of text if replacing special characters by entity references is undesirable; this could be the case, for example, with program code or an XML fragment that is to be reproduced with its markup tags. CDATA sections can occur within element content and are delimited by `<![CDATA[ and ]]>` at the start and end, respectively. Consequently the text string `]]>` should not occur within a CDATA section as it would prematurely signify the end of the section. In this case, the greater than character should be escaped by its entity reference (`&gt;`). To insert a CDATA section within an element, place the cursor at the desired location, right-click, and select **Insert CDATA Section** from the context menu. To see the CDATA section tags in Authentic View, **switch on the markup display**. Alternatively, you could highlight the text that is to be enclosed in a CDATA section, and then select the **Insert CDATA section** command.

**Note:** CDATA sections cannot be inserted into input fields (that is, in text boxes and multiline text boxes). CDATA sections can only be entered within elements that are displayed in Authentic View as text content components.
Editing and following links

A hyperlink consists of two parts: the link text and the target of the link. You can edit the link text by clicking in the text and editing. But you cannot edit the target of the link. (The target of the link is set by the designer of the stylesheet (either by typing in a static target address or by deriving the target address from data contained in the XML document.) From Authentic View, you can go to the target of the link by pressing Ctrl and clicking the link text. (Remember: merely clicking the link will set you up for editing the link text.)

10.3.2 Tables in Authentic View

The three table types fall into two categories: SPS tables (static and dynamic) and CALS/HTML Tables.

SPS tables are of two types: static and dynamic. SPS tables are designed by the designer of the StyleVision Power Stylesheet to which your XML document is linked. You yourself cannot insert an SPS table into the XML document, but you can enter data into SPS table fields and add and delete the rows of dynamic SPS tables. The section on SPS tables below explains the features of these tables.

CALS/HTML tables are inserted by you, the user of Authentic View. Their purpose is to enable you to insert tables at any allowed location in the document hierarchy should you wish to do so. The editing features of CALS/HTML Tables and the CALS/HTML Table editing icons are described below.

SPS Tables

Two types of SPS tables are used in Authentic View: static tables and dynamic tables.

Static tables are fixed in their structure and in the content-type of cells. You, as the user of Authentic View, can enter data into the table cells but you cannot change the structure of these tables (i.e. add rows or columns, etc) or change the content-type of a cell. You enter data either by typing in text, or by selecting from options presented in the form of check-box or radio button alternatives or as a list in a combo-box. After you enter data, you can edit it.

Please note: The icons or commands for editing dynamic tables must not be used to edit static tables.

<table>
<thead>
<tr>
<th>Street:</th>
<th>119 Oakstreet, Suite 4876</th>
<th>Phone:</th>
<th>+1 (321) 555 5155</th>
</tr>
</thead>
<tbody>
<tr>
<td>City:</td>
<td>Vereno</td>
<td>Fax:</td>
<td>+1 (321) 555 5155 - 9</td>
</tr>
<tr>
<td>State &amp; Zip:</td>
<td>DC 29213</td>
<td>E-mail:</td>
<td><a href="mailto:office@nanonull.com">office@nanonull.com</a></td>
</tr>
</tbody>
</table>

Nanonull, Inc.
**Dynamic tables** have rows that represent a repeating data structure, i.e. each row has an identical data structure (not the case with static tables). Therefore, you can perform row operations: append row, insert row, move row up, move row down, and delete row. These commands are available under the **Authentic** menu and as icons in the toolbar (shown below).

To use these commands, place the cursor anywhere in the appropriate row, and then select the required command.

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>Title</th>
<th>Ext</th>
<th>EMail</th>
<th>Shares</th>
<th>Leave</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vernon</td>
<td>Callaby</td>
<td>Office Manager</td>
<td>581</td>
<td><a href="mailto:v.callaby@nanonull.com">v.callaby@nanonull.com</a></td>
<td>1500</td>
<td>25</td>
</tr>
<tr>
<td>Frank</td>
<td>Further</td>
<td>Accounts Receivable</td>
<td>471</td>
<td><a href="mailto:f.further@nanonull.com">f.further@nanonull.com</a></td>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>Loby</td>
<td>Matias</td>
<td>Accounting Manager</td>
<td>963</td>
<td><a href="mailto:l.matias@nanonull.com">l.matias@nanonull.com</a></td>
<td>[add Shares]</td>
<td>23</td>
</tr>
</tbody>
</table>

**Employees: 3 (20% of Office, 9% of Company)**

Shares: 1500 (13% of Office, 6% of Company)

**Non-Shareholders: Frank Further, Loby Matias.**

To move among cells in the table, use the Up, Down, Left, and Right arrow keys. To move forward from one cell to the next, use the **Tab** key. Pressing the **Tab** key in the last cell of the last row creates a new row.

**CALS/HTML Tables**

CALS/HTML tables can be inserted by you, the user of Authentic View, for certain XML data structures that have been specified to show a table format. There are three steps involved when working with CALS/HTML tables: inserting the table; formatting it; and entering data. The commands for working with CALS/HTML tables are available as icons in the toolbar (see [CALS/HTML table editing icons](#)).

**Inserting tables**

To insert a CALS/HTML table do the following:

1. Place your cursor where you wish to insert the table, and click the **icon.** (Note that where you can insert tables is determined by the schema.) The Insert Table dialog (screenshot below) appears. This dialog lists all the XML element data-structures for which a table structure has been defined. For example, in the screenshot below, the *informaltable* element and *table* element have each been defined as both a CALS table as well as an HTML table.
2. Select the entry containing the element and table model you wish to insert, and click **OK**.
3. In the next dialog (*screenshot below*), select the number of columns and rows, and specify whether a header and/or footer is to be added to the table and whether the table is to extend over the entire available width. Click **OK** when done.

For the specifications given in the dialog box shown above, the following table is created.

![Insert Table dialog](image)

By using the **Table** menu commands, you can add and delete columns, and create row and column joins and splits. But to start with, you must create the broad structure.

### Formatting tables and entering data

The table formatting will already have been assigned in the document design. However, you might, under certain circumstances, be able to modify the table formatting. These circumstances are as follows:
The elements corresponding to the various table structure elements must have the relevant CALS or HTML table properties defined as attributes (in the underlying XML Schema). Only those attributes that are defined will be available for formatting. If, in the design, values have been set for these attributes, then you can override these values in Authentic View.

In the design, no style attribute containing CSS styles must have been set. If a style attribute containing CSS styles has been specified for an element, the style attribute has precedence over any other formatting attribute set on that element. As a result, any formatting specified in Authentic View will be overridden.

To format a table, row, column, or cell, do the following:

1. Place the cursor anywhere in the table and click the (Table Properties) icon. This opens the Table Properties dialog (see screenshot), where you specify formatting for the table, or for a row, column, or cell.

![Table Properties dialog screenshot](image)

2. Set the cellspacing and cellpadding properties to "0". Your table will now look like this:

   ![Table formatting](image)

3. Place the cursor in the first row to format it, and click the (Table Properties) icon. Click the Row tab.
Since the first row will be the header row, set a background color to differentiate this row from the other rows. Note the Row properties that have been set in the figure above. Then enter the column header text. Your table will now look like this:

<table>
<thead>
<tr>
<th>Name</th>
<th>Telephone</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notice that the alignment is centered as specified.

4. Now, say you want to divide the "Telephone" column into the sub-columns "Office" and "Home", in which case you would need to split the horizontal width of the Telephone column into two columns. First, however, we will split the vertical extent of the header cell to make a sub-header row. Place the cursor in the "Telephone" cell, and click the (Split vertically) icon. Your table will look like this:

<table>
<thead>
<tr>
<th>Name</th>
<th>Telephone</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5. Now place the cursor in the cell below the cell containing "Telephone", and click the (Split horizontally) icon. Then type in the column headers "Office" and "Home". Your table will now look like this:
Now you will have to split the horizontal width of each cell in the "Telephone" column.

You can also add and delete columns and rows, and vertically align cell content, using the table-editing icons. The CALS/HTML table editing icons are described in the section titled, CALS/HTML Table Editing Icons.

Moving among cells in the table
To move among cells in the CALS/HTML table, use the Up, Down, Right, and Left arrow keys.

Entering data in a cell
To enter data in a cell, place the cursor in the cell, and type in the data.

Formatting text
Text in a CALS/HTML table, as with other text in the XML document, must be formatted using XML elements or attributes. To add an element, highlight the text and double-click the required element in the Elements Entry Helper. To specify an attribute value, place the cursor within the text fragment and enter the required attribute value in the Attributes Entry Helper. After formatting the header text bold, your table will look like this.

<table>
<thead>
<tr>
<th>Name</th>
<th>Telephone</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The text above was formatted by highlighting the text, and double-clicking the element strong, for which a global template exists that specifies bold as the font-weight. The text formatting becomes immediately visible.

Please note: For text formatting to be displayed in Authentic View, a global template with the required text formatting must have been created in StyleVision for the element in question.

CALS/HTML Table Editing Icons
The commands required to edit CALS/HTML tables are available as icons in the toolbar, and are listed below. Note that no corresponding menu commands exist for these icons.

For a full description of when and how CALS/HTML Tables are to be used, see CALS/HTML Tables.
Insert table

The "Insert Table" command inserts a **CALS/HTML table** at the current cursor position.

Delete table

The "Delete table" command deletes the currently active table.

Append row

The "Append row" command appends a row to the end of the currently active table.

Append column

The "Append column" command appends a column to the end of the currently active table.

Insert row

The "Insert row" command inserts a row above the current cursor position in the currently active table.

Insert column

The "Insert column" command inserts a column to the left of the current cursor position in the currently active table.

Join cell left

The "Join cell left" command joins the current cell (current cursor position) with the cell to the left. The tags of both cells remain in the new cell, the column headers remain unchanged and are concatenated.

Join cell right

The "Join cell right" command joins the current cell (current cursor position) with the cell to the right. The contents of both cells are concatenated in the new cell.
Join cell below

The "Join cell below" command joins the current cell (current cursor position) with the cell below. The contents of both cells are concatenated in the new cell.

Join cell above

The "Join cell above" command joins the current cell (current cursor position) with the cell above. The contents of both cells are concatenated in the new cell.

Split cell horizontally

The "Split cell Horizontally" command creates a new cell to the right of the currently active cell. The size of both cells, is now the same as the original cell.

Split cell vertically

The "Split cell Vertically" command creates a new cell below the currently active cell.

Align top

This command aligns the cell contents to the top of the cell.

Center vertically

This command centers the cell contents.

Align bottom

This command aligns the cell contents to the bottom of the cell.

Table properties

The "Table properties" command opens the Table Properties dialog box. This icon is only made active for HTML tables, it cannot be clicked for CALS tables.
10.3.3 Editing a DB

In Authentic View, you can edit database (DB) tables and save data back to a DB. This section contains a full description of interface features available to you when editing a DB table. The following general points need to be noted:

- The number of records in a DB table that are displayed in Authentic View may have been deliberately restricted by the designer of the StyleVision Power Stylesheet in order to make the design more compact. In such cases, only that limited number of records is initially loaded into Authentic View. Using the DB table row navigation icons (see Navigating a DB Table), you can load and display the other records in the DB table.
- You can query the DB to display certain records.
- You can add, modify, and delete DB records, and save your changes back to the DB. See Modifying a DB Table.

To open a DB-based StyleVision Power Stylesheet in Authentic View:
- Click Authentic | Edit Database Data, and browse for the required StyleVision Power Stylesheet.

Note: In Authentic view, data coming from a SQLite database is not editable. When you attempt to save SQLite data from the Authentic view, a message box will inform you of this known limitation.

Navigating a DB Table

The commands to navigate DB table rows are available as buttons in the Authentic View document. Typically, one navigation panel with either four or five buttons accompanies each DB table.
The arrow icons are, from left to right, Go to First Record in the DB Table; Go to Previous Record; Open the Go to Record dialog (see screenshot); Go to Next Record; and Go to Last Record.

```
Go To Record

Go to record #: 1

OK
Cancel
```

To navigate a DB table, click the required button.

**XML Databases**

In the case of XML DBs, such as IBM DB2, one cell (or row) contains a single XML document, and therefore a single row is loaded into Authentic View at a time. To load an XML document that is in another row, use the Authentic | Select New Row with XML Data for Editing menu command.

**DB Queries**

A DB query enables you to query the records of a table displayed in Authentic View. A query is made for an individual table, and only one query can be made for each table. You can make a query at any time while editing. If you have unsaved changes in your Authentic View document at the time you submit the query, you will be prompted about whether you wish to save all changes made in the document or discard all changes. Note that even changes made in other tables will be saved/discarded. After you submit the query, the table is reloaded using the query conditions.

**Please note:** If you get a message saying that too many tables are open, then you can reduce the number of tables that are open by using a query to filter out some tables.

To create and submit a query:

1. Click the Query button for the required table in order to open the Edit Database Query dialog (see screenshot). This button typically appears at the top of each DB table or below it. If a Query button is not present for any table, the designer of the StyleVision Power Stylesheet has not enabled the DB Query feature for that table.
2. Click the **Append AND** or **Append OR** button. This appends an empty criterion for the query (shown below).

4. Enter the expression for the criterion. An expression consists of: (i) a field name (available from the associated combo-box); (ii) an operator (available from the associated combo-box); and (iii) a value (to be entered directly). For details of how to construct expressions see the **Expressions in criteria** section.

5. If you wish to add another criterion, click the **Append AND** or **Append OR** button according to which logical operator (AND or OR) you wish to use to join the two criteria. Then add the new criterion. For details about the logical operators, see the section **Re-ordering criteria in DB Queries**.
Expressions in criteria

Expressions in DB Query criteria consist of a field name, an operator, and a value. The available field names are the child elements of the selected top-level data table; the names of these fields are listed in a combo-box (see screenshot above). The operators you can use are listed below:

- `=` Equal to
- `<>` Not equal to
- `<` Less than
- `<=` Less than or equal to
- `>` Greater than
- `>=` Greater than or equal to
- `LIKE` Phonetically alike
- `NOT LIKE` Phonetically not alike
- `IS NULL` Is empty
- `NOT NULL` Is not empty

If IS NULL or NOT NULL is selected, the Value field is disabled. Values must be entered without quotes (or any other delimiter). Values must also have the same formatting as that of the corresponding DB field; otherwise the expression will evaluate to FALSE. For example, if a criterion for a field of the date datatype in an MS Access DB has an expression `StartDate=25/05/2004`, the expression will evaluate to FALSE because the date datatype in an MS Access DB has a format of `YYYY-MM-DD`.

Using parameters with DB Queries

You can enter the name of a parameter as the value of an expression when creating queries. Parameters are variables that can be used instead of literal values in queries. When you enter it in an expression, its value is used in the expression. Parameters that are available have been defined by the SPS designer in the SPS and can be viewed in the View Parameters dialog (see screenshot below). Parameters have been assigned a default value in the SPS, which can be overridden by passing a value to the parameter via the command line (if and when the output document is compiled via the command line).

To view the parameters defined for the SPS, click the Parameters button in the Edit Database Query dialog. This opens the View Parameters dialog (see screenshot).
The View Parameters dialog contains all the parameters that have been defined for the stylesheet in the SPS and parameters must be edited in the stylesheet design.

Re-ordering criteria in DB Queries
The logical structure of the DB Query and the relationship between any two criteria or sets of criteria is indicated graphically. Each level of the logical structure is indicated by a square bracket. Two adjacent criteria or sets of criteria indicate the AND operator, whereas if two criteria are separated by the word OR then the OR operator is indicated. The criteria are also appropriately indented to provide a clear overview of the logical structure of the DB Query.

The DB Query shown in the screenshot above may be represented in text as:

State=CA AND (City=Los Angeles OR City=San Diego OR (City=San Francisco AND CustomerNr=25))
You can re-order the DB Query by moving a criterion or set of criteria up or down relative to the other criteria in the DB Query. To move a criterion or set of criteria, do the following:

1. Select the criterion by clicking on it, or select an entire level by clicking on the bracket that represents that level.
2. Click the Up or Down arrow button in the dialog.

The following points should be noted:

- If the adjacent criterion in the direction of movement is at the same level, the two criteria exchange places.
- A set of criteria (i.e. criterion within a bracket) changes position within the same level; it does not change levels.
- An individual criterion changes position within the same level. If the adjacent criterion is further outward/inward (i.e. not on the same level), then the selected criterion will move outward/inward, **one level at a time**.

To delete a criterion in a DB Query, select the criterion and click **Delete**.

**Modifying a DB Query**

To modify a DB Query:

1. Click the **Query** button. The Edit Database Query dialog box opens. You can now edit the expressions in any of the listed criteria, add new criteria, re-order criteria, or delete criteria in the DB Query.
2. Click **OK**. The data from the DB is automatically re-loaded into Authentic View so as to reflect the modifications to the DB Query.

**Modifying a DB Table**

**Adding a record**

To add a record to a DB table:

1. Place the cursor in the DB table row and click the **icon (to append a row) or the **icon (to insert a row). This creates a new record in the temporary XML file.
2. Click the **File | Save** command to add the new record in the DB. In Authentic View a row for the new record is appended to the DB table display. The **AltovaRowStatus** for this record is set to **A** (for Added).

When you enter data for the new record it is entered in bold and is underlined. This enables you to differentiate added records from existing records—if existing records have not been formatted with these text formatting properties. Datatype errors are flagged by being displayed in red.

The new record is added to the DB when you click **File | Save**. After a new record is saved to the DB, its **AltovaRowStatus** field is initialized (indicated with **---**) and the record is displayed in Authentic View as a regular record.

**Modifying a record**

To modify a record, place the cursor at the required point in the DB table and edit the record as
When you modify a record, entries in all fields of the record are underlined and the AltovaRowStatus of all primary instances of this record is set to U (for Updated). All secondary instances of this record have their AltovaRowStatus set to u (lowercase). Primary and secondary instances of a record are defined by the structure of the DB—and correspondingly of the XML Schema generated from it. For example, if an Address table is included in a Customer table, then the Address table can occur in the Design Document in two types of instantiations: as the Address table itself and within instantiations of the Customer table. Whichever of these two types is modified is the type that has been primarily modified. Other types—there may be more than one other type—are secondary types. Datatype errors are flagged by being displayed in red.

The modifications are saved to the DB by clicking **File | Save**. After a modified record is saved to the DB, its AltovaRowStatus field is initialized (indicated with ---) and the record is displayed in Authentic View as a regular record.

**Please note:**

- If even a single field of a record is modified in Authentic View, the entire record is updated when the data is saved to the DB.
- The date value 0001-01-01 is defined as a NULL value for some DBs, and could result in an error message.

**Deleting a record**

To delete a record:

1. Place the cursor in the row representing the record to be deleted and click the strike-through icon. The record to be deleted is marked with a strikethrough. The AltovaRowStatus is set as follows: primary instances of the record are set to D; secondary instances to d; and records indirectly deleted to X. Indirectly deleted records are fields in the deleted record that are held in a separate table. For example, an Address table might be included in a Customer table. If a Customer record were to be deleted, then its corresponding Address record would be indirectly deleted. If an Address record in the Customer table were deleted, then the Address record in the Customer table would be primarily deleted, but the same record would be secondarily deleted in an independent Address table if this were instantiated.

2. Click **File | Save** to save the modifications to the DB.

**Please note:** Saving data to the DB resets the Undo command, so you cannot undo actions that were carried out prior to the save.

### 10.3.4 Working with Dates

There are two ways in which dates can be edited in Authentic View:

- Dates are entered or modified using the **Date Picker**.
- Dates are entered or modified by **typing in the value**.

The method the Authentic View user will use is defined in the SPS. Both methods are described
in the two sub-sections of this section.

Note on date formats
In the XML document, dates can be stored in one of several date datatypes. Each of these datatypes requires that the date be stored in a particular lexical format in order for the XML document to be valid. For example, the \texttt{xs:date} datatype requires a lexical format of \texttt{YYYY-MM-DD}. If the date in an \texttt{xs:date} node is entered in anything other than this format, then the XML document will be invalid.

In order to ensure that the date is entered in the correct format, the SPS designer can include the graphical Date Picker in the design. This would ensure that the date selected in the Date Picker is entered in the correct lexical format. If there is no Date Picker, the Authentic View should take care to enter the date in the correct lexical format. Validating the XML document could provide useful tips about the required lexical format.

Date Picker
The Date Picker is a graphical calendar used to enter dates in a standard format into the XML document. Having a standard format is important for the processing of data in the document. The Date Picker icon appears near the date field it modifies (see screenshot).

![Organization Chart]

To display the Date Picker (see screenshot), click the Date Picker icon.

![Date Picker]

To select a date, click on the desired date, month, or year. The date is entered in the XML document, and the date in the display is modified accordingly. You can also enter a time zone if this is required.
Text Entry

For date fields that do not have a Date Picker (see screenshot), you can edit the date directly by typing in the new value.

Please note: When editing a date, you must not change its format.

If you edit a date and change it such that it is out of the valid range for dates, the date turns red to alert you to the error. If you place the mouse cursor over the invalid date, an error message appears (see screenshot).

If you try to change the format of the date, the date turns red to alert you to the error (see screenshot).

10.3.5 Defining Entities

You can define entities for use in Authentic View, whether your document is based on a DTD or an XML Schema. Once defined, these entities are displayed in the Entities Entry Helper and in the Insert Entity submenu of the context menu. When you double-click on an entity in the Entities Entry Helper, that entity is inserted at the cursor insertion point.

An entity is useful if you will be using a text string, XML fragment, or some other external resource in multiple locations in your document. You define the entity, which is basically a short name that stands in for the required data, in the Define Entities dialog. After defining an entity you can use it at multiple locations in your document. This helps you save time and greatly enhances maintenance.

There are two broad types of entities you can use in your document: a parsed entity, which is XML data (either a text string or a fragment of an XML document), or an unparsed entity, which is non-XML data such as a binary file (usually a graphic, sound, or multimedia object). Each entity has a name and a value. In the case of parsed entities the entity is a placeholder for the XML data. The value of the entity is either the XML data itself or a URI that points to a .xml file that contains the XML data. In the case of unparsed entities, the value of the entity is a URI that points to the non-XML data file.
To define an entity:

1. Click **Authentic | Define XML Entities...** This opens the Define Entities dialog *(screenshot below).*

2. Enter the name of your entity in the Name field. This is the name that will appear in the Entities Entry Helper.

3. Enter the type of entity from the drop-down list in the Type field. The following types are possible: An **Internal** entity is one for which the text to be used is stored in the XML document itself. Selecting **PUBLIC** or **SYSTEM** specifies that the resource is located outside the XML file, and will be located with the use of a public identifier or a system identifier, respectively. A system identifier is a URI that gives the location of the resource. A public identifier is a location-independent identifier, which enables some processors to identify the resource. If you specify both a public and system identifier, the public identifier resolves to the system identifier, and the system identifier is used.

4. If you have selected PUBLIC as the Type, enter the public identifier of your resource in the PUBLIC field. If you have selected Internal or SYSTEM as your Type, the PUBLIC field is disabled.

5. In the Value/Path field, you can enter any one of the following:
   - If the entity type is Internal, enter the text string you want as the value of your entity. Do not enter quotes to delimit the entry. Any quotes that you enter will be treated as part of the text string.
   - If the entity type is SYSTEM, enter the URI of the resource or select a resource on your local network by using the Browse button. If the resource contains parsed data, it must be an XML file (i.e., it must have a `.xml` extension). Alternatively, the resource can be a binary file, such as a GIF file.
   - If the entity type is PUBLIC, you must additionally enter a system identifier in this field.

6. The NDATA entry tells the processor that this entity is not to be parsed but to be sent to the appropriate processor. The NDATA field must therefore contain some value to indicate that the entity is an unparsed entity.

**Dialog features**

You can do the following in the Define Entities dialog:
Append entities
Insert entities
Delete entities
Sort entities by the alphabetical value of any column by clicking the column header; clicking once sorts in ascending order, twice in descending order.
Resize the dialog box and the width of columns.
Locking. Once an entity is used in the XML document, it is locked and cannot be edited in the Define Entities dialog. Locked entities are indicated by a lock symbol in the first column. Locking an entity ensures that the XML document valid with respect to entities. (The document would be invalid if an entity is referenced but not defined.)
Duplicate entities are flagged.

Limitations of entities

- An entity contained within another entity is not resolved, either in the dialog, Authentic View, or XSLT output, and the ampersand character of such an entity is displayed in its escaped form, i.e. `&amp;`.
- External unparsed entities that are not image files are not resolved in Authentic View. If an image in the design is defined to read an external unparsed entity and has its URI set to be an entity name (for example: `logo`), then this entity name can be defined in the Define Entities dialog (see screenshot above) as an external unparsed entity with a value that resolves to the URI of the image file (as has been done for the logo entity in the screenshot above).

### 10.3.6 XML Signatures

An SPS can be designed with an XML signature configured for Authentic View. When XML signatures are enabled in the SPS, the Authentic View user can digitally sign the Authentic XML file with the enabled signature. After the document has been signed, any modification to it will cause the verification of the signature to fail. Whenever a signed Authentic XML document is opened in the Authentic View of any Altova product, the verification process will be run on the document and the result of the verification will be displayed in a window.

**Note:** XML signatures can be used, and will be verified, in the Authentic View of Enterprise and Professional editions of the following Altova products: Authentic Desktop, Authentic Browser, XMLSpy, and StyleVision.

**XML signature actions**

The following Authentic View user actions for signatures are possible:

- **Choosing the certificate/password:** Signatures are authenticated with either a certificate or a password. The authentication object (certificate or password) is required when the signature is created and again when it is verified. If an Authentic XML document has a signature-enabled SPS assigned to it, the SPS might specify a default certificate or password for the signature. Whether a default certificate or password has been specified or not, the signature can be configured to allow the Authentic View user to select an own certificate/password. The Authentic View user can do this at any time in the XML Signature dialog (shown below). Selecting an own certificate/password overrides the default certificate/password. The own certificate/password is stored in memory and is used for the current session. If, after an own certificate/password has been selected, the
Authentic View user closes the file or the application, the SPS reverts to its default setting for the certificate/password.

- **Signing the document:** The Authentic XML document can be signed either automatically or manually. Automatic signing will have been specified in the signature configuration by the SPS designer and causes the Authentic XML document to be signed automatically when it is saved. If the automatic-signing option has not been activated, the document can be signed manually. This is done by clicking the XML Signature toolbar icon or the Authentic | XML Signature command, and, in the XML Signature dialog that then pops up (screenshot above), clicking the Sign Document button. Note that signing the document with an embedded signature would require the schema to allow the Signature element as the last child element of the root (document) element. Otherwise the document will be invalid against the schema. When signing the document, the authentication object and the placement of the signature are determined according to the signature configuration. You must ensure that you have access to the authentication information. For more information about this, consult your SPS designer.

- **Verifying the Authentic XML document:** If an SPS has XML Signatures enabled, the verification process will be run on the signature each time the Authentic View XML document is loaded. If the password or certificate key information is not saved with the SPS and signature, respectively, the Authentic View user will be prompted to enter the password or select a certificate for verification. Note that if an embedded signature is generated, it will be saved with the XML file when the XML file is saved. The generated signature must be explicitly removed (via the Remove Signature button of the XML Signature dialog; see screenshot above) if you do not wish to save it with the XML file. Similarly, if a detached signature is generated, it too must be explicitly removed if it is not required.

### 10.3.7 Images in Authentic View

Authentic View allows you to specify images that will be used in the final output document (HTML, RTF, PDF and Word 2007). You should note that some image formats might not be supported in some formats or by some applications. For example, the SVG format is supported in PDF, but not in RTF and would require a browser add-on for it to be viewed in HTML. So, when selecting an image format, be sure to select a format that is supported in the output formats of your document. Most image formats are supported across all the output formats (see list below).

Authentic View is based on Internet Explorer, and is able to display most of the image formats that your version of Internet Explorer can display. The following commonly used image formats are supported:

- GIF
- JPG
Relative paths
Relative paths are resolved relative to the SPS file.

10.3.8 Keystrokes in Authentic View

The Enter key
In Authentic View the Enter key is used to append additional elements when it is in certain cursor locations. For example, if the chapter of a book may (according to the schema) contain several paragraphs, then pressing Enter inside the text of the paragraph causes a new paragraph to be appended immediately after the current paragraph. If a chapter can contain one title and several paragraphs, pressing Enter inside the chapter but outside any paragraph element (including within the title element) causes a new chapter to be appended after the current chapter (assuming that multiple chapters are allowed by the schema).

Please note: The Enter key does not insert a new line. This is the case even when the cursor is inside a text node, such as paragraph.

Using the keyboard
The keyboard can be used in the standard way, for typing and navigating. Note the following special points:

- The Tab key moves the cursor forward, stopping before and after nodes, and highlighting node contents; it steps over static content.
- The add... and add Node hyperlinks are considered node contents and are highlighted when tabbed. They can be activated by pressing either the spacebar or the Enter key.
10.4 Authentic Scripting

The **Authentic Scripting** feature provides more flexibility and interactivity to SPS designs. These designs can be created or edited in StyleVision Enterprise and Professional editions, and can be viewed in the Authentic View of the Enterprise and Professional editions of Altova products.

A complete listing of support for this feature in Altova products is given in the table below. Note, however, that in the trusted version of Authentic Browser plug-in, internal scripting is turned off because of security concerns.

<table>
<thead>
<tr>
<th>Altova Product</th>
<th>Authentic Scripts Creation</th>
<th>Authentic Scripts Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>StyleVision Enterprise</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>StyleVision Professional</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>StyleVision Standard *</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>XMLSpy Enterprise</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>XMLSpy Professional</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>XMLSpy Standard</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>AuthenticDesktop Enterprise</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Authentic Browser Plug-in</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Enterprise Trusted **</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Authentic Browser Plug-in</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Enterprise Untrusted</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* No AuthenticView
** Scripted designs displayed. No internal macro execution or event handling. External events fired.

Authentic Scripts behave in the same way in all Altova products, so no product-specific code or settings are required.

**Authentic Script Warning Dialog**

If a PXF file, or an XML file linked to an SPS, contains a script and the file is opened or switched to Authentic View, then a warning dialog (screenshot below) pops up.

![Authentic Script Warning](image)

You can choose one of the following options:

- Click **Yes**, to add the folder containing the file to the Trusted Locations list for Authentic scripts. Subsequently, all files in the trusted folder will be opened in Authentic View.
without this warning dialog being displayed first. The Trusted Locations list can be accessed via the menu command Authentic | Trusted Locations, and modified.

- Click **No** to not add the folder containing the file to the Trusted Locations list. The file will be displayed in Authentic View with scripts disabled. The Authentic Script Warning dialog will appear each time this file is opened in Authentic View. To add the file's folder to the Trusted Locations list subsequently, open the Trusted locations dialog via the menu command Authentic | Trusted Locations, and add the folder or modify as required.

For a description of the Trusted Locations dialog, see the description of the Authentic | Trusted Locations menu command in the User Reference.

**Note:** When XMLSpy is accessed via its COM interface (see Programmers' Reference to see how this can be done), the security check is not done and the Authentic Script Warning dialog is not displayed.

### How Authentic Scripting works

The designer of the SPS design can use Authentic Scripting in two ways to make Authentic documents interactive:

- By assigning scripts for user-defined actions (macros) to design elements, toolbar buttons, and context menu items.
- By adding to the design event handlers that react to Authentic View events.

All the scripting that is required for making Authentic documents interactive is done within the StyleVision GUI (Enterprise and Professional editions). Forms, macros and event handlers are created within the Scripting Editor interface of StyleVision and these scripts are saved with the SPS. Then, in the Design View of StyleVision, the saved scripts are assigned to design elements, toolbar buttons, and context menus. When an XML document based on the SPS is opened in an Altova product that supports Authentic Scripting (see table above), the document will have the additional flexibility and interactivity that has been created for it.

### Documentation for Authentic Scripting

The documentation for Authentic Scripting is available in the documentation of StyleVision. It can be viewed online via the Product Documentation page of the Altova website.
11 HTML and CSS

XMLSpy provides intelligent editing features for HTML and CSS documents. Both types of documents can be edited in Text View, and the active HTML document can be previewed in Browser View.

The intelligent editing features of each type of document is described separately in the sub-sections of this section: HTML and CSS.
11.1 **HTML**

HTML documents can be edited in Text View, and the edited page can then be viewed immediately in Browser View. Text View provides a number of useful HTML editing features. These are described in detail in Text View, but the main features, as well as HTML-specific options, are listed below.

**Support level**

XMLSpy supports HTML 4.0 and HTML 5.0. Entry-helper and intelligent editing are available for the respective HTML versions. These features are described below.

**Entry helpers**

Elements, Attributes and Entities entry helpers are available when an HTML document is active. The entry helpers are context-sensitive; the items displayed in the entry helpers are those available at the current cursor location. Use the HTML entry helpers as described in Text View.

**Auto-completion**

As you type markup text into your HTML document, XMLSpy provides Auto-completion help. A pop-up containing a list of all nodes available at the cursor insertion point is displayed. As you type, the selection jumps to the first closest match in the list (see screenshot below). Click the selected item to insert it at the cursor insertion point.

Auto-completion for elements appears when the left bracket of node tags is entered. When the start tag of an element node is entered in the document, the end tag is automatically inserted as well. This ensures well-formedness.

Auto-completion for attributes appears when a space is entered after the element name in a start tag. When you click an attribute name in the Auto-completion pop-up, the attribute is entered with quotes characters and the cursor positioned between the quotes.

The Entities entry helper contains character entities from the HTML 4.0 and HTML 5.0 entity sets, Latin-1, special characters, and symbols.
HTML Info window

The HTML Info window (screenshot below) lists applications that can be used to quickly access the active HTML file. For example, if an HTML file is active in XMLSpy, double-clicking the Mozilla Firefox item in the HTML Info Window starts an instance of Mozilla Firefox and loads the active HTML document in it.

Note the following usage points:

- The icon to the right of the Open HTML With item enables applications to be added to the Open HTML With list. All the browsers installed on the system, or any other application (such as a text editor), can be added via the menu commands accessed via the Open HTML With icon. The associated applications would typically be browser or editor applications.
- After an application has been added to the Open HTML With list (except when added with the Add Installed Browsers command), its name in the Open HTML With list can be changed by selecting it, pressing F2, and editing the name.
- The icons to the right of each application listed in the Open HTML With list each opens a menu containing commands to: (i) open the application; (ii) open the application and load the linked HTML file; (iii) remove the application from the list. Double-clicking an application name opens the linked HTML file in that application.
- Applications added to or removed from the Open HTML With list are also added to or removed from the CSS Info window.

Assigning a DTD

For XHTML documents, a DTD or XML Schema can be assigned via the DTD/Schema menu, which enables you to browse for the required DTD or XML Schema file. An XHTML document can be edited exactly like an XML document.

Browser View commands

Browser View commands are available in the Browser menu.
11.2 CSS

CSS documents can be edited using Text View's intelligent editing features. These features, as they apply to the editing of CSS documents, are listed below.

**Syntax coloring**

A CSS rule consists of a selector, one or more properties, and the values of those properties. These three components may be further sub-divided into more specific categories; for example, a selector may be a class, pseudo-class, ID, element, or attribute. Additionally, a CSS document can contain other items than rules: for example, comments. In Text View, each such category of items can be displayed in a different color (screenshot below) according to settings you make in the Options dialog (see below).

You can set the colors of the various CSS components in the Text Fonts section of the Options dialog (screenshot below). In the combo box at top left, select CSS, and then select the required color (in the Styles pane) for each CSS item.
**Source folding**

Source folding refers to the ability to expand and collapse each CSS rule, which is indicated in the source folding margin by a +/- sign. The margin can be toggled on and off in the Text View Settings dialog. When a rule is collapsed, this is visually indicated by an ellipsis. If the mouse cursor is placed over an ellipsis, the content of the collapsed rule is displayed in a popup. If the content is too large for a popup, this is indicated by an ellipsis at the bottom of the popup.

The **Toggle All Folds** icon in the Text toolbar toggles all rules to their expanded forms or collapses all rules to the top-level document element.

**Note:** that the pair of curly braces that delimit a rule (screenshot above) turns bold when the cursor is placed either before or after one of the curly braces. This indicates clearly where the definition of a particular rule starts and ends.

**CSS outline**

The CSS Outline entry helper (screenshots below) provides an outline of the document in terms of its selectors. Clicking a selector in the CSS Outline highlights it in the document. In the screenshot at left below, the selectors are unsorted and are listed in the order in which they appear in the document. In the screenshot at right, the Alphabetical Sorting feature has been toggled on (using the toolbar icon), and the selectors are sorted alphabetically.

You should note the following points: (i) For evaluating the alphabetical order of selectors, all parts of the selector are considered, including the period, hash, and colon characters; (ii) If the CSS document contains several selectors grouped together to define a single rule (e.g. h4, h5, h6 {...}), then each selector in the group is listed separately.

The icons in the toolbar of the CSS Outline entry helper, from left to right, do the following:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image" /></td>
<td>Toggles automatic synchronization (with the document) on and off. When auto-synchronization is switched on, selectors are entered in the entry helper even as you type them into the document.</td>
</tr>
<tr>
<td><img src="image2.png" alt="Image" /></td>
<td>Synchronizes the entry helper with the current state of the document.</td>
</tr>
</tbody>
</table>
Toggles alphabetical sorting on and off. When off, the selectors are listed in the order in which they appear in the document. When sorted alphabetically, ID selectors appear first because they are prefaced by a hash (e.g. #intro).

Properties entry helper

The Properties entry helper (screenshot below) provides a list of all CSS properties, arranged alphabetically. A property can be inserted at the cursor insertion point by double-clicking the property.

Auto-completion of properties and tooltips for properties

As you start to type the name of a property, XMLSpy prompts you with a list of properties that begin with the letters you have typed (screenshot below). Alternatively, you can place the cursor anywhere inside a property name and then press Ctrl+Space to pop up the list of CSS properties.

You can view a tooltip containing the definition of a property and its possible values by scrolling down the list or navigating the list with the Up and Down keys of your keyboard. The tooltip for the highlighted property is displayed. To insert a property, either press Enter when it is selected, or click it.
CSS Info window

When a CSS file is active, the CSS Info window (screenshot below) is enabled. The CSS Info window provides the following functionality:

- It enables you to switch between CSS 2.1 and CSS 3.0. The entry helpers and intelligent editing features of the GUI will be switched according to the CSS version selected in the toolbar of the Info window.
- It enables the CSS file to be linked to an HTML file. This functionality enables you to modify the CSS document and view the effect of changes immediately. Additionally, the linked HTML file can be opened in multiple browsers via the CSS Info window, thus enabling changes in the CSS document to be viewed in multiple browsers.
- The CSS Info window lists the imported CSS stylesheets, thus giving you an overview of the import structure of the active CSS stylesheet.

![CSS Info window screenshot](image)

Note the following usage points:

- The toolbar of the Info window contains icons for CSS 2.1 and CSS 3.0. Select the version you want in order to switch entry helpers and intelligent editing features to the selected CSS version.
- Only one HTML file can be linked to the active CSS document. Do this by clicking the icon to the right of the Linked HTML item, then selecting the command Set Link to HTML and browsing for the required HTML file. The linked HTML file will be listed under the Linked HTML item in the Info window (see screenshot above). Creating this link does not modify the CSS document or the HTML document in any way. The link serves to set up an HTML file to which the active CSS document can be applied for testing.
- Double-clicking the Linked HTML file listing opens the HTML file in XMLSpy.
- The toolbar icons enable you to horizontally and vertically tile the CSS document and the HTML file.
- When changes to the CSS document are saved, the HTML file that is open in XMLSpy can be automatically updated. To enable these automatic updates, check the Update Linked HTML Browser View upon save check box. Note that these updates will only occur if the HTML file contains a reference to the CSS document being edited.
- To change the linked HTML file, select another HTML file via the Set Link to HTML command.
- To remove the link to the HTML file, click the icon to the right of the Linked HTML item.
and select the command **Remove Link**.

- The icon to the right of the *Open HTML With* item enables applications to be added to the *Open HTML With* list. All the browsers installed on the system, or any other application (such as a text editor), can be added via the menu commands accessed via the *Open HTML With* icon. The associated applications would typically be browser or editor applications.

- After an application has been added to the *Open HTML With* list (except when added with the **Add Installed Browsers** command), its name in the *Open HTML With* list can be changed by selecting it, pressing **F2**, and editing the name.

- The icons to the right of each application listed in the *Open HTML With* list each opens a menu containing commands to: (i) open the application; (ii) open the application and load the linked HTML file; (iii) remove the application from the list. Double-clicking an application name opens the linked HTML file in that application.

- Applications added to or removed from the *Open HTML With* list are also added to or removed from the HTML Info window.

- The *Imported* item displays a list of the CSS files imported by the active CSS document.
12 JSON, JSON Schema

JSON (JavaScript Object Notation) is a lightweight data storage and interchange format that uses
JavaScript syntax, and, like XML, is a human-readable, text-only format. Since JSON text can be
read and used by any programming language, it has come to be used widely as a data exchange
format, especially on the web.

As part of its IDE functionality, XMLSpy provides support for the editing and validation of JSON
data documents (instance documents) and for the creation of syntactically and semantically
correct JSON Schema documents.

JSON5

JSON5 is an extension of JSON that adds some ECMAScript 5 extensions (see json5.org for
more information). JSON5 is a strict subset of JavaScript, adds no new data types to existing
JSON types, and works with all existing JSON content.

All XMLSpy functionality that is available for JSON instance documents is also available for
JSON5 instance documents. However, note the following major differences between JSON5 and
JSON, and in the way XMLSpy handles the two formats:

- JSON5 is not an official successor to JSON. It therefore uses its own file extension: .json5.
- By default, XMLSpy recognizes files with the .json file extension as JSON instance
documents, and those with the .json5 file extension as JSON5 instance documents.
- JSON5 instance documents can be validated against JSON schemas.

In this documentation, the term JSON instances refers to both JSON and JSON5 instance
documents unless otherwise indicated. Also see the section Differences between JSON5 and
JSON.

JSON and JSON Schema in XMLSpy

Both document types—JSON instance and JSON schema—are written in JSON format, and must
adhere to JSON rules of well-formedness and validity. Both types of document (instance and
schema) typically have the .json file extension. JSON instances can be edited in Text View and
Grid View, and JSON schema documents can be edited in those two views as well as in JSON
Schema View, which is a graphical schema editor.

XMLSpy provides the following support for working with JSON instance and JSON schema
documents:

- In Text View, syntax coloring and syntax checks; auto-completion in JSON schemas and
  in instance documents if these have schema associations, folding margins; and structural
  markings. All of these features ease and speed up the editing of valid JSON instance and
  JSON schema documents. Text View provides validation of both instance and schema
documents.
- In Grid View, a tabular grid structure that helps to better visualize document structure.
  You can edit directly in Grid View. You can also switch between Text View and Grid View
to suit your editing needs. Grid View provides validation of both instance and schema
documents.
- JSON instance document validation in Text View and Grid View. The validation is carried
out against a JSON schema that is assigned in the Info Window.

- **JSON Schema View** displays JSON schemas in a graphical layout. This enables the use of drag-and-drop functionality (in addition to text entry) for the quick creation of JSON schemas. Entry helpers within the view provide editing input. Additionally, the schema is continuously checked for validity, and errors are flagged.

**JSON instances: opening existing instance documents and creating new instance documents**

- In the **Options | File types** section, you can set the default view (Text View or Grid View) for opening JSON/JSON5 instance documents. Existing JSON/JSON5 documents will be opened in the default starting view you select. You can switch between Text View and Grid View at any time.

- To create a new JSON or JSON5 instance document, click **File | New**, and select, respectively, json: JavaScript Object Notation or json5: JSON with ECMAScript 5 extensions. You will be prompted to optionally choose a JSON schema file for the new instance file. If you assign a schema, the assignment will be entered in the Info Window. The new instance document will be opened in Text View or Grid View, depending on the settings in the **Options | File types** tab.

**JSON schemas: opening existing schemas and creating new schemas**

- An existing JSON schema document opens in **JSON Schema View**. You can switch to Text View or Grid View at any time.

- To create a new JSON schema document, click **File | New**, and select json: JSON Schema. The new JSON schema document will be opened in **JSON Schema View**, with the $schema keyword at the start of the document. You can switch to Text View or Grid View at any time.

All these views (Text, Grid, and JSON Schema) are described in the sub-sections of this section.

**In this section**

This section is organized into the following topics:

- **JSON Data** explains the basics of JSON documents
- **JSON Schema** describes what a JSON schema is and how it works
- **JSON Documents in Text View** shows you how to work with the JSON-relevant features of Text View
- **JSON Documents in Grid View** describes how to edit JSON documents in Grid View
- **Validating JSON Data/Documents** describes how to assign a JSON schema to a JSON document and how to validate JSON documents
- **JSON Transformations with XSLT/XQuery** describes how JSON documents can be queried with XPath/XQuery 3.1
- **Generating JSON Schema from a JSON Instance** describes the functionality to generate a schema from an instance
- **JSON Schema View** explains the JSON-schema-editing features of the view and how you can use it when creating your JSON projects
12.1 JSON Data

This section contains a brief description of how JSON data is structured. JSON data is typically stored in a JSON (instance) document but can also be stored as a JSON data fragment in a document of another type. A JSON data fragment or document is a JSON data structure, which is broadly defined as set out below.

XMLSpy additionally supports JSON5, which is an extension of JSON that adds some minimal ECMAScript 5 extensions. See json5.org for more information.

JSON objects and arrays

A JSON document (saved typically with the file extension .json) is built on the following core data structures:

Object

An object is delimited by curly braces, and is an unordered collection of zero or more key:value pairs. These key:value pairs are the properties of the object. The key must always be a string and must therefore always be enclosed in quotes. The key (also called the name of the property) is separated from its value by a colon. A property value can be of any JSON datatype (see list below). A property is separated from the next by a comma. The listing below is an example of an object with three properties (all of which have atomic-type values):

```json
{
    "emailtype": "home",
    "emailaddress": "contact01.home@altova.com",
    "citycode": 22
}
```

Array

An array is delimited by square brackets, and is a comma-separated ordered list of zero or more items. These items can be of any JSON datatype (see list below).

- An array containing two objects

The array below consists of two objects (each enclosed in curly braces). The array itself is indicated with square brackets.

```json
[
    {
        "emailtype": "home",
        "emailaddress": "contact01.office@altova.com",
        "citycode": 22
    },
    {
        "emailtype": "office",
        "emailaddress": "contact01.office@altova.com",
        "citycode": 22
    }
]
```
Arrays that are the values of an object’s properties

The listing below is of an object with three key:value pairs. Each value is an array that contains a tuple (sequence). (A tuple can be considered to be a one-dimensional array.) The three items in each tuple are atomic types.

```json
{
    "x": [1, 2, "abc"],
    "y": [3, 4, "def"],
    "z": [5, 6, "ghi"]
}
```

JSON data types

Object property values and array items can be of the following types:

- **string** (must be enclosed in quotes). A string can additionally be specified to have a format, such as a *date-time* or *email* format
- **number**: A number with a fractional part; it includes integers
- **integer**: A number with no fractional part; a subset of the number type
- **boolean** (true/false, not enclosed in quotes)
- **object**: When used within another object, allows data to be nested
- **array**: Provides the ability to build more complex structures than allowed by objects
- **null** (null, not enclosed in quotes)

Example of JSON data

Here is an example of a JSON data fragment. Note how the document is structured into objects and arrays. Also note the data type of key values; string values are in quotes, other types are colored green.

```json
{
    "first": "Jason",
    "last": "Jones",
    "isManager": true,
    "age": 35,
    "address": {
        "street": "Jason Avenue",
        "city": "Jasonville",
        "state": "JS",
        "postcode": "JS12 ON34"
    },
    "phone": [
        {
            "type": "home",
            "number": "12 3456-7890"
        },
        {
            "type": "office",
            "number": "789 012-34567"
        }
    ]
}
```
Some differences between JSON5 and JSON

JSON5 is a strict subset of JavaScript, adds no new JSON data types, and works with all existing JSON content. Some notable differences are listed below:

- JSON5 supports comments. Comments are delimited like this: `// comment //` or `/* comment */`.
- In JSON5, the keys of `key:value` pairs do not need to be enclosed in quotes.
- In JSON5, strings can be written across multiple lines.
- JSON5 documents can be validated against JSON schemas.
12.2 JSON Schema

In the same way that an XML Schema specifies the structure and content of an XML document, a JSON schema specifies how the JSON data in a JSON document is organized. It specifies what data fields are expected and how the values are represented. The JSON Schema specification and more information about JSON Schema is available [here](#).

A JSON schema is itself a JSON object. Lexically, the entire schema is contained within curly braces (see listing below), which are the delimiters of JSON objects. The schema is written in JSON syntax and will be saved typically in a file with a `.json` extension. It is indicated as a JSON schema, by the `$schema` keyword, which should be the first keyword of the top-level object, and should have the value: `"http://json-schema.org/draft-N/schema#"`, where `N` is the number of the version. Here is an example of how the `$schema` keyword is used.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  ...
}
```

**Note:** Although the `$schema` keyword can have the value “`http://json-schema.org/schema#`”—which specifies the latest version of the schema—it is best to use a URL that identifies the specific version. For more information, see [JSON Schema Version](#).

In XMLSpy, you can create JSON schemas graphically in JSON Schema View. How to do this is described in the section [JSON Schema View](#). Besides the schema editing features available in JSON Schema View, the following schema-related features are available:

- **Validation with the JSON Validator of XMLSpy:** Assign a JSON schema to a JSON instance document, and validate the instance document from within XMLSpy. See [Validating JSON Documents](#) for information.
- **Generating JSON Schema from a JSON Instance:** If a JSON instance document already exists, you can generate a JSON schema from it. You can subsequently edit the schema if you need to.
- **Converting between JSON and XML:** You can convert between documents of the two formats.

**Terminology**

Given below are definitions of common JSON schema terms used in the GUI and this documentation.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
<td>The top-level schema object in a JSON schema document; the schema file.</td>
</tr>
<tr>
<td>Object</td>
<td>A JSON type containing zero or more properties.</td>
</tr>
<tr>
<td>Property</td>
<td>A <code>key:value</code> pair of an object. Its value can be any JSON datatype.</td>
</tr>
<tr>
<td>Keyword</td>
<td>The <code>key</code> part of an object's <code>key:value</code> pair. It is always a string.</td>
</tr>
<tr>
<td>Sub-schema</td>
<td>An object that is a child of an operator or a dependency.</td>
</tr>
</tbody>
</table>
### Definition
The complete description of any JSON type. Definitions can be global or local.

### Array
A comma-separated ordered list of zero or more items of any JSON datatype.

### Atomic types
The string, number, integer, boolean, and null JSON datatypes.

### Type selectors
The any and multiple types, which select any and multiple types, respectively

### Operators
Occurrence selectors that can be added as children of definitions. See the section Operators.

### JSON data types
Object property values and array items can be of the following types:

- **string** (must be enclosed in quotes). A string can additionally be specified to have a format, such as a date-time or email format
- **number**: A number with a fractional part; it includes integers
- **integer**: A number with no fractional part; a subset of the number type
- **boolean** (true/false, not enclosed in quotes)
- **object**: When used within another object, allows data to be nested
- **array**: Provides the ability to build more complex structures than allowed by objects
- **null** (null, not enclosed in quotes)
12.3 JSON Documents in Text View

Altova website: [JSON Editor]

JSON schemas and JSON/JSON5 instance documents can be edited using the intelligent editing features of Text View. These features include: folding margins, structural marking, syntax coloring, syntax checking, and auto-completion. XMLSpy also provides conversion between JSON/JSON5 and XML in both directions, and enables you to generate a JSON schema from a JSON/JSON5 instance.

Folding margins

Source folding is enabled on JSON keywords and definitions, and refers to the ability to expand and collapse these nodes. Such nodes are indicated in the source folding margin by a +/- sign (see screenshot below). The margin can be toggled on and off in the Text View Settings dialog. When a node is collapsed, this is visually indicated by an ellipsis (see screenshot below). If the mouse cursor is placed over an ellipsis, the content of the collapsed node is displayed in a popup (see screenshot). If the content is too large for a popup, this is indicated by an ellipsis at the bottom of the popup.

The Toggle All Folds icon in the Text toolbar toggles all nodes to their expanded forms or collapses all nodes to the top-level document element.

The following options are available when clicking on the node's +/- icon:

<table>
<thead>
<tr>
<th>Click [+]</th>
<th>Expands the node so that descendant nodes are shown expanded or collapsed according to how they were before the node was collapsed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click [-]</td>
<td>Collapses the node.</td>
</tr>
<tr>
<td>Shift+Click [-]</td>
<td>Collapses all descendant nodes, but leaves the node that was clicked in its expanded form.</td>
</tr>
</tbody>
</table>
Structural marking

The pair of curly braces or square brackets that delimit a JSON object or array, respectively, (see screenshot below) turns bold when the cursor is placed either before or after one of the braces or brackets. This indicates where the definition of a particular element starts and ends.

```
"Department": [ {
  "Name": "Administration",
  "Person": [ {
    "First": "Vernon",
    "Last": "Callaby",
    "Title": "Office Manager",
    "PhoneExt": 582,
    "EMail": "v.callaby@nanonull.com",
    "Shares": 1500,
    "LeaveTotal": 25,
    "LeaveUsed": 4,
    "LeaveLeft": 21
  }
],
```

Syntax coloring

A JSON document (JSON instance/schema), as well as a JSON5 document, is each made up of object strings, value strings, operators, numbers and keywords. In Text View, each category of items can be displayed in a different color (see screenshot above) according to settings you make in the Options dialog (screenshot below). You can set the colors of the various JSON components in the Text Fonts section of the Options dialog (screenshot below). In the combo box at top left, select JSON, and then select the required color (in the Styles pane) for each JSON item.
Note: JSON5 syntax—but not JSON syntax—allows for comments. Comments in JSON5 are delimited like this: `// comment //` or `/* comment */`.

Syntax checking
The syntax of a JSON document (JSON instance/schema) can be checked by selecting the command `XML | Check Well-Formedness (F7)`. The results of the well-formed check are displayed in the Messages window (screenshot below).

The error message in the screenshot above points out an error in the document: An opening curly brace occurs at a location where a colon is expected.
Auto-completion

Auto-completion is enabled when the JSON document (JSON instance/schema) being edited is associated with a schema.

- If the document is a JSON schema, then auto-completion is based on the schema version indicated by the `$schema` keyword. For more information, see also JSON Schema Version.
- If the document is a JSON/JSON5 instance, then a JSON schema must be assigned to the instance in order for auto-completion to be enabled.

Auto-completion provides you with the available entry options at the cursor location. It does this (i) via pop-ups in the main window, and (ii) via the entry helpers (see screenshot below). The pop-ups and entry helpers each display a list of entries that are valid at that cursor location. To move through the entries in the pop-up list, use the arrow keys. If the schema contains a description of the entry (in the entry's `description` keyword in the schema), then the description is displayed next to the highlighted pop-up entry. Select an entry from the pop-up window or double-click an entry in the entry helper to insert it.

In the instance document shown in the screenshot above, the pop-up and JSON Properties entry helper are shown when the cursor is located after the quotes that indicate the start of a property's name. The entry helper displays all the properties allowed at that point; the properties that have already been entered are shown grayed out and disabled. The pop-up displays only the properties that are allowed at that point.

There are two other entry helpers: JSON Values and JSON Entities (screenshot below). These show, respectively, the allowed values of `key:value` pairs and entities for escaping characters in JSON strings. The JSON Values entry helper in the screenshot below shows the values allowed for the `type` keyword while editing a JSON schema. The last entry in the JSON Entities entry helper, `\u00FF`, is a placeholder that stands for a Unicode character. Replace the part highlighted in blue with the code of the Unicode character you want.
Other context-sensitive auto-completion entries or hints include the following, when these are specified in the schema: enumerations, descriptions, required occurrences, and default values.
12.4 JSON Documents in Grid View

Grid View enables you to see the structure of the JSON document (JSON instance/schema) in a grid and to restructure blocks of structured text. This provides you with an overview and editing capability that is not always present in Text View, especially in the case of long and complexly structured documents. In the case of JSON documents, such complexity can occur in the form of arrays and objects being nested within other arrays and objects at multiple levels. For example, compare the JSON text listed below (as it would appear in Text View) and its representation in Grid View (as shown in the screenshot further below).

Note: Avro support is available in the Enterprise Edition only.

JSON code listing in Text View

```json
{
  "web-app": {
    "servlet": {
      "servlet-name": "altovaCDS",
      "servlet-class": "org.altova.cds.CDSServlet",
      "init-param": {
        "configGlossary:installationAt": "Philadelphia, PA",
        "configGlossary:adminEmail": "ksm@pobox.com",
        "configGlossary:poweredBy": "Altova",
        "configGlossary:poweredByIcon": "/images/altova.gif",
        "configGlossary:staticPath": "/content/static",
        "templateProcessorClass": "org.altova.WysiwygTemplate",
        "templateLoaderClass": "org.altova.FilesTemplateLoader",
        "templatePath": "templates",
        "templateOverridePath": ",",
        "defaultListTemplate": "listTemplate.htm",
        "defaultFileTemplate": "articleTemplate.htm",
        "useJSP": false,
        "jspListTemplate": "listTemplate.jsp",
        "jspFileTemplate": "articleTemplate.jsp",
        "cachePackageTagsTrack": 200,
        "cachePackageTagsStore": 200,
        "cachePackageTagsRefresh": 60,
        "cacheTemplatesTrack": 100,
        "cacheTemplatesStore": 50,
        "cacheTemplatesRefresh": 15,
        "cachePagesTrack": 200,
        "cachePagesStore": 100,
        "cachePagesRefresh": 10,
        "cachePagesDirtyRead": 10,
        "searchEngineListTemplate": "forSearchEnginesList.htm",
        "searchEngineFileTemplate": "forSearchEngines.htm",
        "searchEngineRobotsDb": "WEB-INF/robots.db",
        "useDataStore": true,
        "dataStoreClass": "org.altova.SqlDataStore",
        "redirectionClass": "org.altova.SqlRedirection",
        "dataStoreName": "altova",
        "dataStoreDriver": "com.microsoft.jdbc.sqlserver.SQLServerDriver",
        "dataStoreUrl": "jdbc:sqlserver://LOCALHOST:1433;DatabaseName=goon",
        "dataStoreUser": "sa",
    }
  }
}
```
"dataStorePassword": "dataStoreTestQuery",
"dataStoreTestQuery": "SET NOCOUNT ON; select test='test';",
"dataStoreLogFile": "/usr/local/tomcat/logs/datastore.log",
"dataStoreInitConns": 10,
"dataStoreMaxConns": 100,
"dataStoreConnUsageLimit": 100,
"dataStoreLogLevel": "debug",
"maxUrlLength": 500
},
},
"servlet-name": "altovaEmail",
"servlet-class": "org.altova.cds.EmailServlet",
"init-param": {
  "mailHost": "mail1",
  "mailHostOverride": "mail2"
}
},
"servlet-name": "altovaAdmin",
"servlet-class": "org.altova.cds.AdminServlet"
},
"servlet-name": "fileServlet",
"servlet-class": "org.altova.cds.FileServlet"
},
"servlet-name": "altovaTools",
"servlet-class": "org.altova.cms.AltovaToolsServlet",
"init-param": {
  "templatePath": "toolstemplates/",
  "log": 1,
  "logLocation": "/usr/local/tomcat/logs/AltovaTools.log",
  "logMaxSize": "",
  "dataLog": 1,
  "dataLogLocation": "/usr/local/tomcat/logs/dataLog.log",
  "dataLogMaxSize": "",
  "removePageCache": "/content/admin/remove?cache=pages&id=",
  "removeTemplateCache": "/content/admin/remove?cache=templates&id=",
  "fileTransferFolder": "/usr/local/tomcat/webapps/content/fileTransferFolder",
  "lookInContext": 1,
  "adminGroupId": 4,
  "betaServer": true
}
},
"servlet-mapping": {
  "altovaCDS": "/",
  "altovaEmail": "/altovaultil/aemail/*",
  "altovaAdmin": "/admin/*",
  "fileServlet": "/static/*",
  "altovaTools": "/tools/*"
}
},
"taglib": {
  "taglib-uri": "altova.tld",
  "taglib-location": "/WEB-INF/tlds/altova.tld"
}
While the document structure in Text View (listing above) is difficult to make out without a longer, more careful reading, the structure in Grid View (screenshot below) is more readily seen at a glance.

Additionally, the structure can be easily modified by adding, deleting, or moving objects in the grid. Entire blocks of text can be reorganized (for example, by sorting them or moving them). Content, too, can be edited in Grid View. For a detailed explanation of how to work with structured text in Grid View, see the section Grid View.

Auto-completion

Auto-completion is enabled when the JSON document (JSON instance/schema) being edited is associated with a schema.

- If the document is a JSON schema, then auto-completion is based on the schema version indicated by the $schema keyword. For more information, see also JSON Schema Version.
- If the document is a JSON/JSON5 instance, then a JSON schema must be assigned to the instance in order for auto-completion to be enabled.

Auto-completion provides you with the available entry options at the cursor location. It does this (i) via pop-ups in the main window, and (ii) via the entry helpers (see screenshot below). The pop-ups and entry helpers each display a list of entries that are valid at that cursor location. To move through the entries in the pop-up list, use the arrow keys. Select an entry from the pop-up window or double-click an entry in the entry helper to insert it.
In the instance document shown in the screenshot above, the pop-up window and JSON Properties entry helper are shown when the cursor is located in the third property's name field. The entry helper displays all the properties allowed at that point; the properties that have already been entered are shown grayed out and disabled. The pop-up displays only the properties that are allowed at that point.

There are two other entry helpers: JSON Values and JSON Entities (screenshot below). These show, respectively, the allowed values of key:value pairs and entities for escaping characters in JSON strings. The JSON Values entry helper in the screenshot below shows the values allowed for the type keyword while editing a JSON schema. The last entry in the JSON Entities entry helper, \u00FF, is a placeholder that stands for a Unicode character. Replace the part highlighted in blue with the code of the Unicode character you want.

Other context-sensitive auto-completion entries or hints include the following (if these are specified in the schema): enumerations, descriptions, required occurrences, and default values.
12.5 Validating JSON Documents

XMLSpy contains a JSON validation engine that can be invoked to do the following:

- **If a JSON schema is the active document:** Validates the JSON schema against the appropriate JSON Schema specification (for which no additional schema assignment is needed); the schema version is indicated by the `$schema` keyword; the validation can be carried out in any of the three views (Text, Grid, and JSON Schema).

- **If a JSON instance is the active document:** Validates the JSON instance against a JSON schema. The schema is assigned to the JSON instance as described below. JSON instance validation can be carried out in Text View and Grid View.

- **If a JSON5 instance is the active document:** Validates the JSON instance against a JSON schema. The schema is assigned to the JSON5 instance as described below. JSON5 instance validation can be carried out in Text View and Grid View.

**Avro validation (Enterprise Edition only)**

Avro data and Avro schema documents, as JSON documents, can be validated in Text View and Grid View:

- **If an Avro data instance in JSON format is the active document:** Validates the Avro instance against an Avro schema. The schema is assigned to the instance as described below.

- **If an Avro schema is the active document:** Validates the Avro schema against the Avro schema specification (no schema assignment is needed); the validation can be carried out in Text View or Grid View.

Assigning a JSON schema to a JSON instance

In order to validate a JSON instance against a JSON schema, the schema must be assigned to the active instance document. The assignment is entered in the Info window (screenshot below; Avro support is available in the Enterprise edition only) of the active JSON instance document, or via the Project Properties dialog (the Validate With option). Note that JSON5 instance documents are validated against JSON schemas.

In the JSON tab of the Info Window, click the arrow icon next to Validation against schema, and, in the menu that appears, click Select JSON Schema (see screenshot above; Avro support is available in the Enterprise edition only). For JSON5 instance documents only the JSON schema option is enabled. Browse for the schema, and click OK. The schema will be assigned to the active JSON instance document, and the schema's filename will be entered in the Info window. If the JSON instance document is empty, the assignment of a JSON schema to the instance will...
automatically fill the JSON instance with sample data based on the schema.

To remove the assignment, select the command **Remove Schema** from the same menu (see screenshot above).

For information about generating JSON schema from the JSON instance, see the section **Generating JSON Schema from a JSON Instance**.

**Validating instance and schema documents**

Select the command **XML | Validate XML (F8)** or click the **Validate (F8)** icon in the toolbar to validate the active JSON document (instance or schema). If an instance document is being validated, a schema document must be assigned to the instance (see above). Validation results are displayed in the **Messages window**.

Errors are also flagged in the line-numbering margin. If a smart fix is available for an error, then a light bulb icon is shown on the line that generates the error. When you place the mouse over the icon, a popup appears that lists available smart fixes. Select a fix to apply it immediately.

**Note:** The validation error indicators and smart fixes described above are refreshed only when the **XML | Validate (F8)** command is executed; they are not updated in the background. So, after correcting an error, you must run the **Validate (F8)** command again to make sure that the error has indeed been fixed.

To go to the schema document from the instance document, double-click the schema in the Info window (see screenshot above), or select the command **DTD/Schema | Go to Schema**. To go directly to the schema definition of a JSON keyword or object, select the keyword or object in the instance document and select **DTD/Schema | Go to Definition**.

You can also validate a [project folder] containing JSON files by using the **Validate** command.
12.6 JSON Transformations with XSLT/XQuery

JSON maps, arrays, and objects can be targeted with XPath/XQuery 3.1 expressions. As a result, JSON documents can be transformed with XSLT 3.0, XQuery 3.1, and XQuery Update 3.0 documents by using the built-in engines of XMLSpy.

The following functionality is available:

- An active JSON document can be queried with XPath/XQuery 3.1 expressions from the XPath/XQuery output window.
- An active JSON document can be transformed with a user-selected XSLT or XQuery file.
- An active XSLT or XQuery document can be executed on a user-selected JSON source file.

These features are described below in more detail.

*Note:* You can try out JSON transformations by using the JSON, XSLT, and XQuery files in the JSON Examples folder of the Examples project located in your application folder: `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2019\Examples`.

Querying a JSON document via the XPath/XQuery output window

JSON documents can be queried by entering an XPath/XQuery 3.1 query expression in the XPath/XQuery output window (see screenshot below). Select either the XPath 3.1 icon or XQuery 3.1 icon, and ensure that the window is in JSON evaluation mode (explained below).

![XPath/XQuery output window](image)

The information given below pertains to evaluations of JSON documents in JSON evaluation mode. (For an overview of the XPath/XQuery window and detailed information about its usage, see the section [Output Window: XPath/XQuery](#).)

**JSON evaluation mode**

JSON evaluation mode is described through these points:

- The XPath/XQuery window will be in either XML evaluation mode or JSON evaluation mode. Which mode is currently active is indicated by the active mode's button being highlighted. See the XML/JSON evaluation mode buttons in the screenshot above. In the screenshot, the window is in JSON evaluation mode.
- In the screenshot above, notice that the XML and JSON buttons are grayed out, indicating that they are disabled. When the buttons are disabled, their status—whether activated or deactivated—cannot be changed. Conversely, if the buttons are enabled (not grayed...
out), then the evaluation mode of the window can be changed.

- The enabled/disabled state of the XML/JSON evaluation mode buttons depends on the evaluation scope (the value of the Where field; see screenshot above). Evaluation-scope values are divided into two groups for the determination of the enabled/disabled state: (i) Single file (Current file), and (ii) Multiple files (Open files, Project, Folder).
  - If, for the evaluation scope, a single file (Current file) is selected (as in the screenshot above), then the window's mode (JSON or XML) is determined on the basis of the file's extension. Either the file is JSON conformant, in which case JSON evaluation mode is activated; or the file is not JSON conformant, and XML evaluation mode is switched on. Since the file type of the single file is known, the appropriate evaluation mode is activated, and both buttons are disabled so that the mode cannot be changed.
  - If a multiple-files option (Open files, Project, Folder) is selected, then both evaluation mode buttons are enabled, and the user can select what mode to activate (JSON or XML). The default evaluation mode for a multiple-file scope is XML.
- In XML evaluation mode, XML conformant files will be processed and JSON files will be skipped.
- In JSON evaluation mode, JSON conformant files will be processed and XML files will be skipped.
- JSON expressions can also be queried in Debug Mode.

The XPath/XQuery expression

In XPath/XQuery expressions, use features that are valid for querying JSON documents, such as the lookup operator and array functions. Note the following points:

- The Set current selection in document as origin feature is disabled. As a result, expressions must be absolute, and cannot be relative.
- As you enter the expression, auto-completion provides context-sensitive completion options (see screenshot below).

- The Copy XPath of current selection to edit field icon creates an XQuery expression that targets the selection in the active JSON document and copies the expression to the window's edit field. To make a selection in the JSON document, place the cursor within an object or select a part of an object.

Given below are two XPath/XQuery expressions that have been used to query the JSON document MusicLibrary.json (see screenshot below), which you can find in the Examples project located in your application folder: C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2019\Examples.

XPath/XQuery expression: `?*`
Results
Results show the JSON component that is selected by the XPath/XQuery expression in the left side of the Results pane, and the component's value in the right side of the pane. Maps and arrays are displayed in short or verbose format according to whether the Show complete result icon in the toolbar is toggled off or on. Maps and arrays in the left side of the pane can be expanded by clicking the plus icon. The values of the expanded components will then be displayed in the right side of the pane.

Transforming a JSON document with XSLT/XQuery
To transform an active JSON document with an XSLT 3.0, XQuery 3.1, or XQuery Update 3.0 document, do the following:

- **XSLT 3.0 transformation**: Click XSL/XQuery | XSL Transformation, browse for the XSLT 3.0 file, and click OK.
- **XQuery 3.1 or XQuery Update 3.0 transformation**: Click XSL/XQuery | XQuery/Update Execution, browse for the XQuery 3.1 or XQuery Update 3.0 file, and click OK.

The transformed document/s will be generated, and can be viewed directly in XMLSpy.

Note: XSLT/XQuery Debugger can be started from a JSON document, but breakpoints and tracepoints can be set in the XSLT or XQuery document only.

Providing a JSON source for an XSLT/XQuery document
To execute an active XSLT or XQuery document on a JSON source file, do the following:

- **Active XSLT 3.0 document**: Click XSL/XQuery | XSL Transformation, browse for the JSON file, and click OK.
- **Active XQuery 3.1 or XQuery Update 3.0 document**: Click XSL/XQuery | XQuery/Update Execution, browse for the JSON file, and click OK.

The transformed document/s will be generated, and can be viewed directly in XMLSpy.

Note: XSLT/XQuery Debugger can be started from an XSLT or XQuery document and a JSON
document can be assigned as input for the debugging session. However, breakpoints and tracepoints can be set in the XSLT or XQuery document only.
12.7 Generating JSON Schema from a JSON Instance

XMLSpy can generate a JSON schema from a JSON instance document (including from JSON5 instances). This feature is very useful since it quickly provides you with a schema based on an already existing JSON instance, and saves you the trouble of manually creating a schema from scratch. You can then modify or extend the generated schema according to your requirements.

Generating the schema

To generate a JSON schema from a JSON instance, do the following:

1. Make the JSON instance document the active document.
2. In the JSON tab of the Info Window, click the arrow icon next to Validation against schema, and, in the menu that appears, click Generate JSON Schema (see screenshot below; Avro support is available in the Enterprise edition only). Alternatively, select the menu command DTD/Schema | Generate DTD/Schema.
3. In the Generate JSON Schema dialog that now appears (see next section below), modify the settings as you want (see below for details). Click OK when done.
4. You will be prompted to provide a path and filename for the generated JSON schema. On clicking Save, the schema is generated and becomes the active document. In the instance document, the generated schema file will be assigned as the schema to use for validation; any previous assignment will be overwritten. To change the assignment, use the Select JSON Schema command of the context menu (see screenshot above).

Note: The version of the JSON schema will be the default version selected in the JSON tab of the Options dialog.

Settings for generating the schema

You can specify options for schema generation in the Generate JSON Schema dialog (screenshot below). See the previous section for information about how to access this dialog.
Detect array tuples

An array tuple is the sequence of items in an array. For example, the following array has a tuple with three items: [1, 2, "abc"]. For the validation of arrays, the schema can specify whether the order and datatype of array (tuple) items are to be considered or not. If the Detect Array Tuples option is checked (see screenshot above), then the order and datatype of items will be detected. Based on what is detected, a corresponding definition will be created in the schema.

The options for this setting are as follows:

- **Number of tuple items**: A minimum and maximum number of tuple items can be specified. If a tuple in the instance has an item-count within this range, then this array will detected and defined.
- **Simple types only**: Only tuples that have simple-type items (the atomic types string, number, integer, boolean, and null) are to be considered for detection.
- **Identically named arrays**: Only arrays that are defined as values of properties that have the same name are considered for detection. For example, in the following JSON data fragment, the arrays marked with red-shaded brackets are all values of properties named a1 (shaded in blue): 
  
  ```json
  { "object1": [ { "a1": [1, 2, "abc" ] }, { "a1": [3, 4, "def" ] }, { "a1": [5, 6, "ghi" ] } ] }
  ```

- **Minimum number of arrays**: A minimum number of arrays for enabling array detection can be specified.

Other settings

- **Ignore order of object property names**: If selected, the order of an object’s properties is checked and recreated as closely as possible. Otherwise, the order is not checked.
- **Try to match the string format**: The schema can specify that string datatypes must have a particular format. If this option is selected, then XMLSpy will try to detect the string format and add a format definition for strings wherever possible.
- **Make simple arrays local**: A simple array is one in which all items are of the same simple
datatype. If selected, all simple arrays will be defined locally in the schema, instead of using global definitions that are referenced locally.

- **Make simple objects local:** A simple object is one in which all property values are of the same simple datatype. If selected, all simple objects will be defined locally in the schema, instead of using global definitions that are referenced locally.

**Note:** After the JSON schema has been generated, you can make local definitions of individual objects and arrays global, and vice versa. For more information, see the section Global and Local Definitions.
12.8 Generating a JSON Instance from a JSON Schema

You can generate a JSON instance from a JSON schema when the JSON schema is the active file in Text View, Grid View, or Schema View. Click the DTD/Schema | Generate Sample XML/JSON File. Note that this command generates a JSON document, not a JSON5 document.
12.9 Converting between JSON and XML

The following conversion options are available:

- **Convert XML Instance to JSON**: When an XML instance document is the active document, you can select whether to generate a JSON or JSON5 instance document. Use the command **Convert | Convert XML Instance to/from JSON**.

- **Convert JSON Instance to XML**: When a JSON/JSON5 instance document is the active document, an XML instance document is generated from the JSON instance by clicking **Convert | Convert XML Instance to/from JSON**.

- **Convert XML Schema to JSON Schema**: When an XML Schema document is the active document, a JSON schema document is generated from the XML Schema by clicking **Convert | Convert XML Schema to/from JSON Schema**.

- **Convert JSON Schema to XML Schema**: When a JSON schema document is the active document, an XML Schema document is generated from the JSON schema by clicking **Convert | Convert XML Schema to/from JSON Schema**.

All these conversions are enabled in both Text View and Grid View. Click the links above to see descriptions of the respective functionality.
12.10 JSON Schema View

JSON Schema View can be used to view and edit JSON schema documents. The main parts of the JSON Schema View window are:

- A main window that switches between a **Definitions Overview Grid** and a **Design View**
- Three **entry helper windows** (located by default on the right-hand side of the main window): Overview, Details, and Constraints
- A Messages window (located by default below the main window)
- An **Info window** (located by default at bottom left of the application window)

The screenshot below shows the main window and the Overview entry helper.

![JSON Schema View](image)

The main window

The main window switches between a **Definitions Overview Grid** (shown in screenshot above) and a **Design View** (screenshot below). Definitions Overview Grid shows the current document's main schema (listed as "Document Schema"), plus any definitions that you add to the schema. (A definition is a description of a JSON data structure. In the screenshot above, object_01 and array_01 are definitions, of an object and an array, respectively.) Definitions are also listed in the Overview entry helper (see screenshot above).

While Definitions Overview Grid provides a high level view of the JSON schema, it does not show what is within any definition listed in the overview. To view and edit a definition in Design View (screenshot below), click the definition's icon (see screenshot above) or double-click the definition in the Overview entry helper (see screenshot above).
To switch back to Definitions Overview Grid from Design View, click the **Switch to Definitions Grid** icon at the top left of Design View (see screenshot above). To configure Design View, click the menu command **Schema Design | Configure View**.

**The entry helpers**

Both modes of Schema View (Definitions Overview Grid and Design View) have three entry helpers: Overview, Details, and Constraints. These entry helpers provide mechanisms for: (i) displaying information about the schema and its definitions, and (ii) entering information and values related to definitions. They are described in detail in the section **Entry Helpers: Overview, Details, Constraints**.

12.10.1 **JSON Schema Version**

A JSON schema is written in JSON syntax and will be saved typically in a file with a .json extension. It is indicated as a JSON schema by the **$schema** keyword, which should be the first keyword of the top-level object and have a value of: "http://json-schema.org/draft-N/schema#" or "http://json-schema.org/schema#".

```json
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  ...
}
```

or
In the first example above, the schema version is explicitly named, with version names having the form draft-04, draft-06, etc. If the version name is omitted, as in the second example above, then this indicates that the schema version to be used is the latest version (currently draft-07).

In JSON Schema View, you can change the version in the combo box in the bar at the top of the main window (see screenshot below).

![Screenshot of combo box with version options]

Features of new schema versions that are not defined in an older version

If you use a feature of a newer schema version and then switch to an older version that does not support this feature, then the following happens:

- A message appears asking whether you wish to remove/convert the feature or keep the feature
- If kept, the new feature's corresponding component or detail is shown in an orange text color. For example, if a value has been set for the const keyword (new in draft-06) and you switch the schema version to draft-04, then the value of the const keyword is displayed in orange.

JSON Schema versions

For information about the JSON Schema specifications, especially about additional features with each version, see the links below:


12.10.2 Adding Global Definitions

The Definitions Overview Grid in the main window (screenshot below) displays a list of the schema's global definitions. These global definitions are: (i) the main document schema definition, (ii) definitions of global JSON types, such as objects, arrays, strings, etc, that are JSON Schema types; (iii) definitions of external or custom defined JSON types; currently only definitions that occur within a container named resourceDefinitions is available; this is the container used by Microsoft's Azure Resource Manager for JSON definitions. Add a new resourceDefinitions section to the schema document via the Append Definitions Section or Insert Definitions Section icon in the grid's toolbar (see screenshot below).
Defining a JSON type globally is useful if that type needs to be reused within the same schema or in another schema. For example, you can define a JSON string type for US telephone numbers in one JSON schema—say, a library of such definitions—and then reference this definition from within the same schema as well as from other JSON schemas.

Adding a definition, and related actions
The following actions are available for adding and editing definitions in the Definitions Overview Grid.

- To add a definition: Click the Append Named Schema Definition or Insert Named Schema Definition icon at the top left of the Definitions Overview Grid (see screenshot above). A new empty definition will be created in the grid; it will have a default name. The new definition will also be listed in the Overview entry helper as a Def (see screenshot above).
- To change the type of a definition: Every new definition is created with a type of Any. You can change its type in the Details entry helper (see screenshot below, where the type is String) or by editing the definition in Design View.

- To rename a definition: Double-click its name and edit the name. Alternatively, edit the Name field in the Details entry helper.
- To enter a description of the definition: Edit the Description field in the Details entry helper. The description appears in the Definitions Overview Grid next to the name of the definition (see screenshot below). You can also double-click in the Description field of Definitions Overview Grid to edit a description.
To reference a definition: See the description of the Overview entry helper and the section Global and Local Definitions.

To edit a definition: Click the definition's icon in the Definitions Overview Grid or double-click the definition in the Overview entry helper. This opens the definition in Design View, where it can be edited.

12.10.3 Entry Helpers: Overview, Details, Constraints

The JSON Schema View entry helpers are located by default on the right-hand side of the application window. They are available in both modes of the main window: (i) Definitions Overview Grid, and (ii) Design View. You can drag entry helper windows by their title bars to other locations on the screen, and you can double-click an entry helper's title bar to alternatively dock and undock that entry helper. For more information about these actions, see the section Entry Helpers.

Overview entry helper

The Overview entry helper (screenshot below) lists the current schema definition and all the global definitions of the current schema. Double-clicking a definition, opens that definition in Design View, where it can be edited. If you wish to use definitions from external schemas, first add the external schema, then reuse the definition you want.

Adding the external schema

Add the external schema by clicking the Add New Schema icon in the Overview entry helper and then browsing for the schema you wish to add. Once a schema has been added, its definitions are displayed in the Overview entry helper. The screenshot below, for example, shows that the schema TelNumbers.json has been added, and that this schema has one definition named USTelephoneNumbers. You can add as many external schemas as you like.

Reusing an external definition

After an external schema has been added, its definitions become available for reuse in the definitions of the importing schema. When one definition reuses another definition (by referencing
it), it takes on the properties of that definition. The referencing can be done in two ways:

- **In Design View:** By dragging a definition from the Overview entry helper onto the definition where it is wanted.
- **In Definitions Overview Grid or Design View:** Via the Reference field of the Details entry helper of the definition where the reuse is wanted. This is explained below in the description of the Details entry helper.

**Note:** The Refresh icon next to the External Schemas entry in the Overview window updates all added external schemas. Note that, if no definition from an added external schema has been reused, then that schema will be removed from the list when the list is refreshed.

**Details entry helper**

The properties of a definition can be entered in the Details and Constraints windows when the definition is selected in either mode of the main window: Definitions Overview Grid or Design View. The screenshot below shows the definition of `USTelephoneNumbers` in Design View, together with the Detail and Constraints entry helpers. Notice that the information in the two entry helpers is also displayed in the definition's (blue) box in Design View. The properties that can be set in these two entry helpers are listed below.

The following details can be entered in the Details entry helper:

- **Name:** The name of the definition.
- **Reference:** If you want a definition to reuse another definition, click the Additional Dialog button of the Reference field. This displays the Edit Reference dialog (screenshot below), which lists all available definitions (from the current schema and external schemas). Select the definition you want to reuse, select the Relative Path option if you want a relative path, and click OK.
• **Type:** Select the definition's datatype from the dropdown list of the combo box. Note that changing the type will lead to the removal of keywords specific to the previous type. If you wish to go back to the previous definitions, press **Undo (Ctrl-Z)**. The types are explained in [JSON Data](#) and [Type Selectors (Any, Multiple, etc)](#).

• **ID:** This is an optional keyword that is used to alter the resolution scope of the current definition (which can be regarded as a sub-schema within its parent schema). The ID value must be a string that is a URI. Note that the Altova JSON validator uses canonical de-referencing only. See the [JSON specification](#) for more information.

• **Title, Description:** The values of these two keywords are used for descriptive purposes that can be read by the end-user.

• **Comment (new in draft-07):** Intended for notes to schema maintainers, as opposed to Description, which is intended for end-users.

• **Const (new in draft-06):** A constant value, like a one-value enumeration.

• **Default:** The default value of the definition.

• **Read-only, Write-only (new in draft-07):** These indicate, respectively, read-only and write-only fields. An example of a write-only field would be a password field.

### Constraints entry helper
A definition's constraints depends on its type. The constraints of each type are described below. (See also [Atomic Types](#).)
If a type does not appear in the list below, no constraint can be defined for it. Note, however, that enumerations can be defined for all types:

- **String**: The length of the string, and the pattern of the string; the pattern is specified by means of a regular expression. In the Format field, you can select one of the string formats defined in the specification (see screenshot above, which shows the formats available in draft-04); additional formats have been defined in later versions. Content Media Type and Content Encoding (both new in draft-07) select the media type and encoding of non-JSON data encoded in a JSON string.
- **Numeric**: The range of allowed values
- **Array**: The number of items allowed in the array
- **Object**: The number of allowed properties

The Constraints entry helper for all types has an Enumerations tab. In it, you can specify a list of allowed items of that definition's type. Additionally, an Examples tab is available (new in draft-06) for all types except Forbidden. This is an array of examples with no validation effect; the value of default is usable as an example without repeating it under this keyword.

### 12.10.4 Global and Local Definitions

JSON schema definitions can be created globally or locally.

- **Global definitions** are created in the Definitions Overview Grid of the main window by adding a definition and then specifying its properties. A global definition can be referenced by other definitions in the same schema or by definitions in other schemas. This enables the reuse of definitions across your project. All the global definitions of the current schema are displayed in the schema's Definitions Overview Grid. Global definitions from other schemas can be made available for reuse by adding the external schema in the Overview entry helper.
- **Local definitions** are created within global definitions, that is, by adding descendant or sibling definitions to a global definition.

**Reusing a global definition**

To reuse a global definition, do one of the following:
- In Design View, drag the global definition from the Overview entry helper onto the definition where it is to be used.
- In the Definitions Overview Grid or in Design View, select the definition for which you want the reuse. In the Reference field of the Details entry helper, select the global definition you want to reuse. See the description of the Details entry helper for details.

Note: If you change the name of a global definition after it has been referenced by another definition in the same schema, then the name is also changed in the reference. References from other schemas, however, will need to be edited manually to reflect the name change.

Converting local definitions to global definitions
To convert a local definition, right-click it in Design View and select Make Global. A global definition is created and a reference to it will be created on the local definition. Since the name of the global definition is generated automatically, you can edit it and the change will be passed to the reference of the local definition.

Changing a ref to a global definition into a local definition
A reference to a global definition can exist on both local and global definitions. To remove the reference and make its properties local, right-click the (local or global) definition in Design View and select Make Local. The global definition's properties are created locally on the definition.

12.10.5 Design View
In Design View, you can specify the structure and allowed values of individual global definitions. The definitions are specified via the following GUI components or mechanisms:

- the Details entry helper (also available in Definitions Overview Grid)
- the Constraints entry helper (also available in Definitions Overview Grid)
- the definition's context menu (accessed by right-clicking the definition's box in the main window)

The definitions that can be specified via the Details and Constraints entry helpers are described in the section Entry Helpers: Overview, Details, Constraints. Some of these properties can also be specified within the definition's box in the main window. In this section, and the next three sections, we describe the mainly graphical mechanism available in the main window.

Note: If you need to undo an inadvertent or unwanted change, press Ctrl+Z.

Context menu
The context menu of a definition (blue box in screenshot below) enables you to design the structure of the definition and edit its properties.
The following commands are available:

- **Add Child**: What child can be added depends on the type of the definition (see Add Child: creating structure below).
- **Reference**: Enables the definition to reference a global definition and take on the properties of that global definition. The Edit Reference dialog that the command opens is the same as that accessed via the Details entry helper and is described in the section Entry Helpers: Overview, Details, Constraints.
- **Make Global**: This command is enabled when the definition is a local definition. It makes the currently selected definition a global definition and adds a reference to that global definition in the current selection.
- **Make Local**: This command is enabled when the definition is a global definition. It converts the currently selected definition to a local definition by creating a reference to the original global definition.
- **Go to Definition**: If the selected definition is contained within a definition that references a global definition, then this command is enabled. Clicking it takes you to the global definition.
- **Content**: The Content command displays a submenu containing commands to cut, copy and reset the contents of the selected definition.
- **Edit Description**: Enables the definition's Description field to be edited.

**Add Child: creating structure**

The structure of a definition is created by adding multiple levels of descendants. These levels are created with the Add Child command of the context menu. The children that can be added to a definition depends on its type:
Objects: take properties and operators
Arrays: take array items and operators
Atomic types (string, number, boolean, null): take operators
Any: takes properties, array items, and operators
Multiple: varies according to what types are included; takes the union of allowed children for the selected types
Operators: enables logical operators to be used to determine the structure

The structures that can be created for each type are described in detail in the sections that are linked to from the list above.

12.10.6 Objects and Properties

An object is enclosed in curly braces and maps a key to a value, like this: "MyKey": Value. The key must always be a string and must therefore be enclosed in quotes. The value can be any JSON data type. Each key:value pair is known as a property of the object (see screenshot below).

Here is an example of an instantiated object that has three properties:

```json
{
    "emailtype": "home",
    "emailaddress": "contact01.home@altova.com",
    "citycode": 22
}
```

The schema for the object would look something like this in Design View.
Notice the following:

- Each of the properties must be present in the instance. This is indicated by the solid borders of the properties. If a property is optional, the border is a dashed line. You can set whether a property is required or optional in the property's context menu or via the Details entry helper.
- The order in which properties must occur in the instance is not—and cannot be—defined in the schema. This means that the order in which properties are defined in the schema is irrelevant.
- The blue-square-within-braces symbol signifies a property (as opposed to a pattern property or property wildcard, both of which are indicated by other symbols; see below).
- The type of a property can be edited by double-clicking the type in the diagram and selecting an option from the dropdown list that appears. Alternatively, the type can be selected in the Details entry helper.
- The constraint value of the `emailaddress` property is defined in the Constraints entry helper.

Properties, pattern properties, property wildcards, and property names schemas

An object can have properties, pattern properties, property wildcards, and property names schemas (new in draft-07). These can be added to the object via the context menus: (i) of the object, (ii) of the yellow properties box (right-click the Properties title of the box), and (iii) of individual properties. Properties have been described above. We now look at pattern properties and property wildcards.

A pattern property (screenshot below) defines the property's name as a regular expression. In the screenshot below, for example, the regular expression specifies that the property must: (i) have a name that begins with an underscore, and (ii) have a boolean as its value. There is no requirement constraint for a pattern property. You can add any number of pattern properties. Notice the icon for pattern properties.
A **property wildcard** *(screenshot below)* specifies that any number of properties can occur in addition to the other properties of the object's property set. The wildcard can however define a type for these occurrences. The screenshot below left shows a property wildcard that defines properties with any name but having numeric values. There can be only one property wildcard per object. If the wildcard is set to *Any* type, however, then you can set constraints for each type in the Constraints entry helper. Notice the icon for property wildcards.

![Property Wildcard Screenshot](image)

A **property names schema** *(screenshot below)* constrains the names of that object's properties. *(This feature is new in draft-07.)* For example, in the screenshot below, we can see that the names of properties must be strings. Additionally, we can specify further constraints for the property name via the Constraints entry helper: for example, that the property's name fall within a certain character length range or that it have a certain pattern.

![Property Names Schema Screenshot](image)

**Note:** There are no minimum or maximum occurrence settings for a pattern property or property wildcard. See the section about *property validation* to understand this better.

**How properties are validated**

When a property is encountered in the instance, it is validated as follows:

1. The property's name is checked in the schema against all the named properties of that object.
2. If no match is found, the name is checked against all pattern properties in the object's property set.
3. If still no match is found, then the wildcard is invoked if it exists.
4. If still no match is found for the name, a validity error is reported. If the name matches that of a property or pattern property, or if a wildcard exists, then the value is checked against the value of the corresponding property definition.
5. If the instance value matches the type and constraints of the corresponding property definition, then the property is valid. Otherwise it is invalid.

**Example**

The screenshot below defines an object which:
- must have three properties named `emailtype`, `emailaddress`, and `citycode`.
- can have one or more properties with a name that begins with an underscore and a value that is a boolean (see the pattern property in the screenshot below).
- can have one or more additional properties with any name and any value.

![JSON Schema Diagram](https://via.placeholder.com/150)

### 12.10.7 Unspecified Properties

In the code listing below, the `required` keyword specifies that four properties are required for this object. However, only three of these four properties are defined. The fourth property, `city`, is undefined. The defined properties are said to be `specified`, while the undefined property is said to be `unspecified`. See the screenshots below the listing.

```json
Code listing: specified and unspecified properties
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "description": "JSON Schema generated by XMLSpy v2016 (http://www.altova.com)",
  "type": "object",
  "properties": {
    "emailtype": {
      "type": "string"
    },
    "emailaddress": {
      "type": "string"
    },
    "city": {
      "type": "string"
    },
    "citycode": {
      "type": "string"
    }
  }`
```
{
  "type": "string",
  "format": "email"
},
  "citycode": {
    "type": "number"
  }
],
  "required": [
    "emailtype",
    "emailaddress",
    "citycode",
    "city"
  ],
  "additionalProperties": false
}
In Design View, the unspecified property is flagged in red because it is required by the schema, but is not defined. Although the JSON schema itself is valid, an instance document that is validated against it will not be valid. This is because: (i) If the city property is not present, the document will be invalid because the city property is required; (ii) If the city property is present, the document will be invalid because the city property is undefined and there is no property wildcard to allow its presence (see Implicitly Specifying a Property below).

To create a definition for an unspecified property, do the following:

1. Select the unspecified property in Design View.
2. In the Details entry helper, check the Specified check box (see screenshot above). Alternatively, the Specified flag can be modified via the context menu.
3. Modify the property's definition as required.

Implicitly specifying a property

A property can be implicitly specified by adding a suitable pattern property or property wildcard. The screenshot below shows that a property wildcard has been added. An instance property named city will match this wildcard. In the schema, therefore, the city property is said to be implicitly specified by the wildcard. An instance file containing the city property will be valid against this schema.
Notice the respective icons in the implicitly specified property and in the property wildcard. Each icon is a link to the other property. Double-clicking one icon selects the other property.

12.10.8 Objects and Dependencies

Within the definition of an object, you might want to specify that a certain property is to be present only if another property is present. The first property is said to be dependent on the second property. Here is a scenario containing a dependency. An object (named, say, `member`) has a property called `credit_card`, which is defined as optional. The object’s `billing_address` property can be made dependent on the `credit_card` property: Only if the `credit_card` property is present will the `billing_address` property be present.

This kind of dependency can be specified in one of two ways:

- as a property dependency (the dependent structure is a property)
- as a schema dependency (the dependent structure is a schema)

Property dependencies

The screenshot below shows an object having a `name` property (required), a `credit_card` property (optional), and a `billing_address` property (dependent). The `billing_address` property is dependent on the `credit_card` property. The code of this JSON object definition is listed below the screenshot. How to create a property dependency is described further below.
To create a property dependency, do the following:

1. Right-click the property on which the dependency will be based. (In our example this is the `credit_card` property.)
2. In the context menu that appears, select Add Dependency | Dependent Property. A new property is added with an Occurrence value of Dependent.
3. Define the name and value of this property, and add any additional details or constraints you want.

To specify a property as being dependent on another property, do the following:

1. Right-click the property you want to make dependent on another property. (In our example this is the `billing_address` property.)
2. In the context menu that appears, select Dependent. Alternatively, in the Details entry helper, go to the Occurrence entry, and select Dependent (see screenshot above).
3. In the Details entry helper, click the dropdown list icon of the Dependent On entry. The dropdown list displays all the other properties of the object. Select the property on which you want the current property to depend.

**Note:** An icon appears in the boxes of both properties involved in a dependency (see screenshot above). Double-clicking the icon of one property takes you to the other property.

**Note:** A property can have multiple dependent properties.

### Schema dependencies

The screenshot below shows an object that describes the same instance data structure as the object discussed in the previous section. The definitions of the two objects, however, are different. While the previous definition used a *property dependency* to define the `billing_address` property as being dependent on the `credit_card` property, the current definition uses a *schema dependency* to define this dependency. The code of this latter JSON object definition is listed below the screenshot. How to create a schema dependency is described further below.

```json
{
  "type": "object",
  "properties": {
    "name": {
      "type": "string"
    },
    "credit_card": {
      "type": "integer"
    }
  }
}
```
To create a schema dependency, do the following:

1. Right-click the property on which the dependency will be based. (In our example this is the credit_card property.)
2. In the context menu that appears, select **Add Dependency | Schema Dependency**. A new object definition is created. It will have the same name as the property on which it is dependent (in our example, credit_card), and it will have a child sub-schema.
3. Define the sub-schema the way you want it, adding any additional details or constraints you may want.

**Note:** An icon appears in the boxes of the property and object involved in a dependency (see screenshot above). Double-clicking the icon in one box takes you to the other box.

**Note:** If you wish to set multiple dependencies, do this within the dependent sub-schema (see screenshot above).

### 12.10.9 Arrays

An array is a list of zero or more ordered items; it is delimited by square brackets. Each item in the list is assigned a type. The instance listing below is of an object with three properties. The value of each property is an array *(delimiters highlighted in yellow)*.

```json
{
    "x": [ 1, 2, "abc" ],
    "y": [ 3, 4, "def" ],
    "z": [ 5, 6, "ghi" ]
}
```

All three arrays in the listing above have the same definition. Each contains three ordered items in the following order: (i) a number item, a (ii) a number item, (iii) a string item. A schema description of this object is shown in the screenshot below. Since the definition is the same for all three arrays, the definition has been created in a global array named array_01. Each of the three arrays *(x, y, and z)* references the global array array_01.
In the screenshot above, array $x$ is selected (indicated by its blue highlight), and its details and constraints are shown in the respective entry helpers (see screenshot above). Notice the constraint on the number of allowed items. The number can be edited in the Constraints entry helper and is displayed in the diagram. The array items can be defined in the definition of the array itself, which in this case is the global definition $array_01$ (screenshot below).

Note the following points:

- The **unique** constraint specifies that all items in the array must be unique.
- The numbering of items starts with 0.
The following phrasing in the diagram, $3 \leq \text{items} \leq 3$ and \text{Items: 3..3} (see screenshot above), both indicate the minimum and maximum allowed items. In this case, exactly three items must be present.

Adding array items, array item wildcards, and the contains keyword

Array items, array item wildcards, and an array's contains keyword are added via the context menu of a definition or an array item. An array wildcard enables a broader range objects to be included in the array. The contains keyword on the other hand specifies that the value of the contains keyword must be a valid JSON schema and that at least one of the array's elements must be valid against the given schema.

12.10.10 Atomic Types

There are five JSON atomic (aka simple or primitive) types: (i) string, (ii) number, (iii) integer, (iv) boolean, and (v) null. To specify that a definition is one of these atomic types, do one of the following:

- Double-click the Type value field in the definition's box, and select the type
- In the Details entry helper, select the type from the dropdown list in the Type field.

The constraints of each atomic type are described below.

String

For the string type, you can specify the following constraints: (i) length of the string, (ii) a regular expression that describes the pattern of the string, (iii) a predefined format from the specification.

Numeric

The numeric type is a collective name for two types (number and integer; see screenshot below). The actual type is set in the Mode field (the default of which is number). The difference between the two types is that the number type allows decimals, whereas the integer type does not. If a value exists in the MultipleOf field, then the instance value must be an integer multiple of the MultipleOf value.
Valid values for the number type defined in the screenshot above are: 5.94, 6.93, 7.92, and 8.91.

Boolean and Null

The boolean type takes either true or false as its values. The null type takes null as its value. Neither type takes any constraint.

12.10.11 Type Selectors (Any, Multiple, etc)

In the dropdown lists of the Type combo boxes of JSON Schema View, there are four "types" that are not JSON types: any, multiple, unconstrained, and forbidden. These are actually type selectors.

- The any type selector selects any JSON type. This means that, in the instance, any JSON type will be valid for that particular definition.
- The multiple type selector selects one or more JSON types. This means that if the instance type is one of the JSON types selected in the schema, then the instance type will be valid for that particular definition.
- The unconstrained type selector (new in draft-06) sets no constraint on the JSON type. This means that, in the instance, any JSON type will be valid for a definition with that name.
- The forbidden type selector (new in draft-06) forbids any JSON type, effectively not allowing a definition with that name to exist.

The any type selector

The any type selector can be selected everywhere that a type can be selected. When a definition is added to the schema, any is the default type selection. It specifies that any of the JSON types is valid. This means that the instance type could validly be an object, an array, or any of the atomic types (string, number, integer, boolean, and null).
In the screenshot above, the sub-schema has a type of Any. So, all JSON types are valid for this definition. The following is implied and is implemented accordingly in the UI:

- Since objects are allowed, a properties box is automatically created (see screenshot above). The properties box is defined by default to allow any number of properties of any type (via a property wildcard with a type of Any). You can modify the property definitions as you like.
- Since arrays are allowed, an items box is automatically created (see screenshot above). The array items box is defined by default to allow any number of array items of any type (via an array item wildcard with a type of Any). You can modify the item definitions as you like.
- Since string and numeric (number and integer) types are allowed, constraints for these atomic types can be defined in the Constraints entry helper.

All of these types are therefore implicitly defined with the Any type selector. In order to change the type to a specific type, select that type. There is an alternative way to specify objects and arrays as the type: Right-click the object or array, and select Make Explicit. This makes that type the selected type and removes the other types or makes defined object/array types inactive.

**The multiple type selector**

The multiple type selector can be selected everywhere that a type can be selected. It allows you to select one or more JSON types by checking the types you want to allow (see screenshot below). You can then specify constraints for the selected types in the Constraints entry helper.

In the screenshot above, the sub-schema allows types of string, null, and array. Constraints for these types can be defined in the Constraints entry helper (see screenshot).
String constraints are defined in the Constraints entry helper.
The null type takes no further constraints.
An array items box is automatically created. You can define the number and types of allowed array items.

In an instance document, the selected types will be allowed at the location corresponding to that of the sub-schema.

The unconstrained and forbidden type selectors
The unconstrained and forbidden type selectors can be selected everywhere that a type can be selected. They enable you to specify, respectively, that an object of any type is allowed or that no object of that name is allowed.

In the screenshot above, a definition has two properties. PropertyOne can have a value of any type, whereas no property named PropertyTwo is allowed (see screenshot). In text form, this construct will look like the code listing below.

```
"Definition": {
    "properties": {
      "PropertyOne": true,
      "PropertyTwo": false
    }
}
```

12.10.12 Operators
There are four operators: (i) allOf, (ii) anyOf, (iii) oneOf, and (iv) not. Operators are used to specify conditions of validity as explained below. You can add an operator to any definition. To access the operator sub-menu, right-click the definition to which you wish to add an operator, and then select Add Child | Operator (see screenshot below).
These operators specify conditions for successful validation, as follows:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>All Of</strong></td>
<td>✅✅✅</td>
<td>Contains one or more sub-schemas (definitions), added as children of the operator. An instance is valid if it is valid against all these sub-schemas.</td>
</tr>
<tr>
<td><strong>Any Of</strong></td>
<td>✅</td>
<td>Contains one or more sub-schemas (definitions), added as children of the operator. An instance is valid if it is valid against at least one of these sub-schemas.</td>
</tr>
<tr>
<td><strong>One Of</strong></td>
<td>✅</td>
<td>Contains one or more sub-schemas (definitions), added as children of the operator. An instance is valid if it is valid against exactly one of these sub-schemas.</td>
</tr>
<tr>
<td><strong>Not</strong></td>
<td>❌</td>
<td>Contains exactly one sub-schema (definition), added as a child of the operator. An instance is valid if it is invalid against the given definition.</td>
</tr>
</tbody>
</table>

The screenshot below shows a One Of operator that contains three child sub-schemas (definitions). For the instance to be valid, it must have one JSON data structure (at this point in the document structure) that matches one of the three sub-schema definitions.
Operators can be useful for specifying inheritance and restriction. The screenshot below, for example, shows how to use the All Of operator to define an array containing non-empty unique strings.

12.10.13 Conditionals

Conditionals are a new feature in draft-07. They enable you to specify that validation restrictions are to be different depending on certain aspects of the object, such as its type and/or additional type specific restrictions.

Adding a conditional

You can add a conditional to any definition via the definition's context menu (see screenshot below). To access the conditional's sub-menu, right-click the definition to which you wish to add the conditional, and then select Add Child | Conditional.
Setting up conditional validation

The conditional is added as a box with three elements: **If-Then-Else** (see screenshot below).

![Conditional Diagram](image)

To set up conditional validation do the following:

1. Set up the condition in the **If** box by first selecting a type in the Details entry helper and then a type-based constraint in the Constraints entry helper.
2. In the **Then** box, set up the validation requirements in the event that the condition (specified in the **If** box) is fulfilled.
3. In the **Else** box, set up the validation requirements in the event that the condition (specified in the **If** box) is not fulfilled.

12.10.14 Configuring Design View

When the main window is in Design View mode, you can access the Display Configuration dialog *(screenshot below)* via the menu command **Schema Design | Configure View**. Here you can configure the appearance of Design View.
You can configure the following aspects of Design View:

- **Widths**: Two sliders determine, respectively, the minimum and maximum widths of boxes in Design View. Together they determine the allowed width of boxes.
- **Parent/child distances**: Sets the horizontal distance between each level in the hierarchy.
- **Child/child distances**: Sets the vertical distances between boxes.
- **Width of descriptions**: Sets the width of description lines. If text length exceeds this width, the text wraps to the next line.
- **Details display**: The details of definitions can be switched to display or not in the definitions' boxes by checking or unchecking this option. There is a corresponding toolbar icon.
- **Placeholders display**: Placeholders are items that have not yet been defined; they represent potential items. This option sets whether the display of placeholders is switched on or not. There is a corresponding toolbar icon. For example, the **Add Property** item in the screenshot below is a placeholder.

**Note:** The **Configure View** menu command is enabled only in the **Design View mode**, which shows the detailed definition of an object. It is not available in **Definitions Overview Grid**.
12.10.15 Generating JSON Schema Documentation

If a JSON schema is the active document, you can generate documentation for it by clicking the **Schema Design | Generate Documentation** command. You can output the documentation as an HTML, MS Word, or RTF file and specify the components you want to include. Related JSON components are hyperlinked in the generated documentation, allowing easy navigation.

**Note:** In order to generate documentation in MS Word format, you must have MS Word (version 2000 or later) installed.

Steps to generate JSON schema documentation

To generate documentation for a JSON schema file, do the following:

1. Make the JSON schema the active document.
2. Switch to Schema View.
3. Select the menu command **Schema Design | Generate Documentation**. This opens the JSON Schema Documentation dialog box (**screenshot below**).
4. Select the type of output you want to generate, HTML, MS Word, or RTF.
5. Select the specific components and details you want to include in the documentation, and set other options (**see JSON Schema Documentation Options below**).

![JSON Schema Documentation Dialog Box](screenshot_below)

- **Documentation Design**
  - Use fixed design for Schema documentation in HTML, Word or RTF format.
  - Use user-defined design for HTML, Word, RTF or PDF format; Requires StyleVision.

- **Output format**
  - HTML
  - MS Word
  - RTF
  - PDF (see above)

- **Generate links to local files**
  - Absolute
  - Relative to result file

- **Include**
  - Overview
  - Properties
  - Pattern Properties
  - Property Wildcards
  - Property Names
  - External Schemas

- **Details**
  - Diagram
  - Location
  - Used by
  - Specifying
  - Property Details
  - Schema Details
  - Source Code

- **Options**
  - Show result file after generation
  - Save schema documentation settings

![JSON Schema Documentation Options](screenshot_below)
6. Click **OK** and enter the name of the JSON schema documentation file in the Save As dialog box that appears.

**JSON schema documentation options**

You can select from among the following documentation options:

- The design template can be the built-in (fixed) XMLSpy design, or it can be a user-defined design that is saved in an SPS file. For a description of how to use a user-defined design, see the section **User-Defined Design**.
- The required format is specified in the Output Format pane: either HTML, Microsoft Word, or RTF. The documentation can be generated either as a single file or be split into multiple files. When multiple files are generated, each file corresponds to a component. What components are included in the output is specified using the check boxes in the **Include** pane.
- The **Embed Diagrams** option is enabled for the MS Word and RTF output options. When this option is checked, diagrams are embedded in the result file, either in PNG or EMF format. Otherwise diagrams are created as PNG or EMF files, which are displayed in the result file via object links. When the output is HTML, all diagrams are created as document-external PNG files.
- In the **Include** pane, you select which items you want to include in the documentation. The **Overview** option lists all components, organized by component type, at the top of the file. If **Schema Definitions** is not selected, then all child components are disabled (that is, everything except **External Schemas**).
- The **Details** pane lists the details that may be included for each component. If **Schema Definitions** is not selected, then all details are disabled. Select the details you wish to include in the documentation.
- The **Show Result File** option is enabled for all three output options. When this option is checked, the result files are displayed in Browser View (HTML output), MS Word (MS Word output), and the default application for .rtf files (RTF output).
HTTP

HTTP (Hypertext Transfer Protocol) is the protocol (or set of rules) that defines how files (text, images, audio, video, and other multimedia files) are transmitted over the Internet. Every web server runs a program (known as a daemon) that continuously waits for HTTP requests and handles each as it arrives. For example, when you visit a website's home page, your browser sends an HTTP command to the website's web server that requests the download of the home page; the server's HTTP daemon receives the request and sends the requested page. One significant property of HTTP bears noting: that it is stateless, which means that each HTTP command is carried out independently, without any reference to previous or following commands.

In XMLSpy, you can test HTTP commands in the HTTP output window (screenshot below). Here you can create and send an HTTP request to a web server, and receive and check the response.

Parts of the HTTP output window

The HTTP output window has nine tabs (see screenshot below). You can store a separate request in each tab, and switch between tabs. After creating a request in the window, you can send the request by clicking the Send button. The response is displayed directly in the window.

The window consists of the following parts:

- At the top: (i) a combo box in which to select the HTTP method you want to use; (ii) an entry field for the URL of the web server; (iii) buttons related to the execution of HTTP requests (Send, Import, and Reset).
- A left-hand pane for creating the request.
- A right-hand pane for displaying the response.

How the HTTP output window works is described in the sub-sections of this section.
13.1 Sending the Request

You can send an HTTP request in the HTTP output window (screenshot below). A request is defined in the left-hand pane of the window. For each of the nine tabs of the window you can define a different request, with each request consisting of: (i) the HTTP method of the request and the target URL (defined in the top part of the dialog); (ii) the HTTP headers of the request (in the Headers tab); (iii) connection settings (in the Settings tab); and (iv) in the case of the POST and PUT methods, the HTTP message body (in the Body tab; not shown in the screenshot below).

You can revert a request to the empty state by clicking Reset (located at the top right of the window).

To send an HTTP request, do the following:

1. In the combo box at top left (see screenshot) select an HTTP method (GET, POST, PUT, DELETE, HEAD, or OPTIONS).
2. Enter the URL of the target web page (for example, http://www.altova.com. You can also enter just altova.com; the http:// part of the URL will be completed for you).
3. In the Headers tab, you can specify HTTP header values (see screenshot above). You can select or enter an header, and then enter its value. (For a list of HTTP 1.1 headers, see here.) Use the Insert, Append, and Delete icons in the tab's toolbar to add or delete headers. Instead of deleting a header, you can deactivate a header by unchecking the Activate check box to the left of the header's name; this will save you the trouble of having to re-enter a deleted header if you ever want to use it later. Also see the section The Accept Header. (If you set a value for any header that would be added automatically at send-time, then the value you enter will be used instead of the value that would have been automatically added.)
4. If you are sending a POST or PUT request, a Body tab will become available in addition to the Headers and Settings tabs. How to create the body of a POST or PUT request is described in the section The body of POST and PUT requests below.
5. You can specify timeouts and security settings in the Settings tab. For a description of this tab, see Settings for the HTTP request below.
6. Click Send (located at the top right side of the window) to send the request.
7. If you wish to revert to the tab's empty state, click Reset. The following happens: (i) The method to use is reset to the first method in the dropdown list of the combo box (which is GET); (ii) the current URL entry is removed; (iii) All header, setting, and body definitions are removed.

Note: You can also import a request from a WADL file into the HTTP output window via the window's Import button.
Note: The request is sent in UTF-8 encoding. Any other encoding is converted to UTF-8, and the UTF-8 data is sent.

HTTP methods
The following HTTP methods are supported:

**GET**
The GET method requests the resource located at the specified URL. You can also add a query to the URL; for example: http://www.altova.com?name1=value1&name2=value2. The resource is returned in a message that contains a header and a body.

**HEAD**
The HEAD method is identical to the GET request, but returns no message body, only a message header containing meta information about the resource located at the specified URL.

**POST**
The POST method is used to update an existing resource located at the specified URL, or to create a new resource at the specified URL. The data to be submitted to the resource is placed in the body of the HTTP request; see The body of POST and PUT requests for information about how to do this.

**PUT**
The PUT method is used to create a new resource at the specified URL. The data to be submitted to the resource is placed in the body of the HTTP request; see The body of POST and PUT requests for information about how to do this.

**DELETE**
The DELETE method deletes the resource located at the specified URL.

**OPTIONS**
The OPTIONS method returns a list of the HTTP methods that the server supports.

The body of POST and PUT requests
For POST and PUT requests, an additional Body tab becomes available, in which the body of the request can be specified (see screenshot below). The Body tab has two modes: Editor mode and File mode. You can switch between these two modes via toolbar buttons at the top left of the Body tab (see screenshot). In Editor mode (shown selected in the screenshot below), you can edit the HTTP request directly in the pane, whereas in File mode you can select a file that contains the body of the HTTP request.

The Content Type field enables you to specify the Content-Type header of the request. The combo box options of this field are different for each mode (Editor and File). You can select from the available combo box options or enter a a MIME type. Note that the value specified in this field overrides any Content-Type header that might be specified in the Headers tab or Body tab.
Editor mode
The style of the editor depends on the selected content type:

- For the `text/plain`, `text/xml`, `application/xml`, and `application/json` content types: A text editor that provides intelligent editing features such as syntax coloring for XML and JSON documents and line-numbering. The screenshot above shows the editor for the `text/xml` content type. The body of the request is entered in the editor. The content-type of the request is specified in the `Content Type` field and cannot be overridden by entries elsewhere in the request.

- For the `application/x-www-form-urlencoded` content type: The editor is a grid view (screenshot below) in which each new line represents a name–value pair in the body of the request.

File mode
In File mode (screenshot below), the body of the request will be the contents of the selected file. This file can be either the file that is currently active in the Main Window (Current file option) or an external file that can be browsed for.
To switch to File mode, select the File icon near the top left of the Body tab (see screenshot above). To enable the content type of the body to be determined automatically, select automatic in the Content Type field. Automatic determination of the the content type is based on the file's extension. If you enter a content type, the request will be sent with the content type you enter; in this case, you must ensure that the content type is the correct one.

The headers and settings of the request can be specified in the same way as for other requests (that is, in the Headers tab and Settings tab, respectively).

**Settings for the HTTP request**

In the Settings tab of the HTTP output window (screenshot below) you can define (i) a connection timeout, and (ii) the security settings of a request. Note that you can define connection settings separately for each of the window's nine tabs. The screenshot below shows the settings for an HTTPS URL.

**Timeout**

You can specify the amount of time in seconds that XMLSpy will try to make a connection with the web server. If this amount of time is reached without a connection being made, then you will get I/O Error 28: Timeout was reached. If you wish to not specify a timeout period, then check the Infinite check box.

**Security Settings**

Click Edit to edit the security settings of a request. The HTTP Security Settings dialog (screenshot below) will be displayed. Here you can specify HTTPS security settings and set the HTTP authentication credentials for the request being made via that tab. If the request's target web server does not use SSL, then only the HTTP authentication credentials will be used. If the target
web server uses SSL, then both the HTTPS security settings as well as the HTTP authentication credentials will be used.

- **HTTPS security settings:** By default, the Check server certificate option will be checked, and you can specify whether the host name in the request may be different than the host name in the certificate. If you are targeting an Intranet URL (say, in your company network), then a client certificate (typically located in your local certificate store) can be used to verify a certificate on the Intranet server.

- **HTTP authentication:** Some requests to a server might require user authentication. For such cases, you can enter a user name and password here. Now, when authentication is required by the server, it will be supplied automatically. Otherwise, you might be prompted for it after the connection to the server is made. When the initial request to the server contains the authentication information, this process is referred to as preemptive authentication. If this is required by the server, select the Preemptive authentication option.
13.2 Importing a Request to Send

In the HTTP output window, you can import a request from a WSDL 1.1, WSDL 2.0, or Web Application Development Language (WADL) file, and then send it. This is done by using XMLSpy's WSDL/WADL Import Wizard. The wizard opens a WSDL or WADL file, selects a request from one of the file's WSDL endpoints or WADL resources, enables you to modify the editable parameters of the request, and then imports the request into the HTTP output window. Do this as follows:

1. In the HTTP output window, click the Import button to start the WSDL/WADL Import Wizard.
2. In the files-selection dialog that appears, browse for the WSDL or WADL file that contains the request you want to import, and click OK. This starts the Import WSDL/WADL Wizard (screenshot below).
3. In the left-hand pane (see screenshot above), select the relevant WSDL endpoint or WADL resource (the one containing the request you want to import).
4. In the right-hand pane (see screenshot above), select the request (the WSDL operation or WADL method) that you want to import. Note that, for import via WSDL: (i) HTTP import is provided only for SOAP and HTTP extensibilities, and (ii) only supported bindings (SOAP and HTTP) are displayed in this (the right-hand) pane.
5. Click OK. If the request contains one or more parameters, then the next screen of the wizard (screenshot below) shows the parameters of the request you selected; otherwise, the request is imported into the HTTP output window and the wizard closes; see point 7 below. Parameters are parts of the request. In a search request, for example, one parameter might be the search term. The wizard validates a parameter's value against its datatype, and indicates one of three states. A pink background indicates an invalid value; a beige background indicates an incorrect value that will nevertheless be entered in the request and sent; a white background indicates a valid value.
6. In the HTTP Request Parameters screen (screenshot above), enter or edit parameter values as needed (in the Values column). Note that you might not be allowed to edit some parameter values; the cells of such values are disabled for editing. If you wish to not use a parameter, then deactivate it by unchecking its Activate check box (in the first column). Note that some parameters are mandatory, so the Activate check box will be locked and you will not be able to uncheck it. Notice that, as you edit the parameter values, the request is being built in the Output URI field. The parameter grid also contains one or more rows for headers (at the bottom of the grid). These headers come from the WADL file, and their values can be edited in the grid if this is allowed according to the definitions in the WADL file. A summary of the headers is listed in the Header field at the bottom of the window.

7. Click OK. The request is imported into the HTTP output window, and is shown there in the following way: (i) In the method combo box, the request's HTTP method will be displayed; (ii) the URL will be constructed on the basis of the request's parameters; (ii) the HTTP headers of the request will be entered in the Headers tab. Note that, if the request is a POST or PUT request, the body of the request will not be entered in the Body tab; it will need to be added manually.

8. Check the Settings tab to see if you need to modify the settings.

9. Click Send to send the request.
13.3 Receiving the Response

The response to an HTTP request is received in the right-hand pane of the HTTP output window (see screenshot below).

To the right of the Body and Headers tab names are listed the following details about the response:

- The HTTP status code (explained below)
- The time from connection made to last response-chunk received
- The Content-Type of the response

The Response pane has two tabs: Body (screenshot below left) and Headers (screenshot below right).
**Body tab:** The body of the response is displayed with syntax coloring if the document is HTML, XML, or JSON, and with line-numbering. The tab has two buttons: (i) **Save to File** to save the body to a file, and (ii) **Create New Document** to create a new document in XMLSpy and display the newly created document in the Main Window of the GUI; a newly created document can be edited and saved in the usual way. If a new document cannot be created from the body of the response (for example if the body is an image), then the **Create New Document** button is disabled (see screenshot further below).

**Headers tab:** Contains the headers of the response. The **Content-Type** header is also displayed at the top of the pane.

### HTTP status codes

Status codes are categorized as follows:

- **2XX codes** are used for successful requests.
- **3XX codes** are used for redirects.
- **4XX codes** are used if there was a problem with the request.
- **5XX codes** are used if there was a problem with the server.

Some commonly encountered codes:

- **200 OK:** Sent in response to a successful request.
- **206 Partial Content:** The server sends only a part of the resource because only a range of the resource was requested.
- **301 Moved Permanently:** The request should be redirected to the given URL.
- **401 Unauthorized:** The resource requires authentication, and authentication has either failed or not been provided.
- **403 Forbidden:** Valid request, but the server is refusing action. This might be because the user does not have the necessary credentials.
- **404 Not Found:** The resource could not be found.
- **500 Internal Server Error:** A generic error message; sent when no more specific message is available.

*See:* [A complete list of HTTP status codes](#).

### The Accept Header

The Accept header of the request specifies the content type to accept in the response. For example, see the difference between the responses when **Accept=image** *(first screenshot below)*
and when `Accept=image/png` *(second screenshot below)*. In the first case, since it is not specified what image format should be sent in the response, an error status code and a JSON message containing more information is sent.
14 Office Open XML, ZIP, EPUB

Office Open XML (OOXML), ZIP files, and EPUB files are similar in that all are packages containing other files. XMLSpy's Archive View provides an interface that enables you to view the internal structure of these packages, modify these structures, and access the files in the package for editing in XMLSpy. In the case of EPUB files, Archive View also enables you to directly view the EPUB book in the Browser View of XMLSpy.

Office Open XML (OOXML)

OOXML is a file format for describing documents, spreadsheets, and presentations. It was originally developed by Microsoft for the company's Office suite of products but is now an open ECMA specification.

Structure of an OOXML file

Each OOXML document is a package of multiple files that follows the Open Packaging Convention. A package consists of XML and other data files (such as image files) plus a relationships file that specifies the relationships among the various files in the package.

The internal structure and internal folder and file names of an OOXML file vary according to the document type. However, there is a common basic structure: an XML file called [Content_Types].xml at the root of the directory structure, and three directories: _rels, docProps, and a directory specific to the document type (in the case of .docx documents, for example, this folder would be called word; xl in .xlsx documents, and ppt in .pptx documents).

OOXML in XMLSpy's Archive View

In XMLSpy's Archive View (screenshot below), you can view and edit the contents of an OOXML file.
Folder View on the left-hand side shows the folders in the package, whereas the Main Window shows the files in the folder selected in Folder View. In Archive View, files and folders can be added to and deleted from the archive. Also, files can be opened quickly for editing in XMLSpy by double-clicking the file in Archive View.

**Intelligent editing of OOXML's internal files**

The XML documents within OOXML packages are based on standard schemas. XMLSpy provides intelligent editing support for OOXML documents, in the form of entry helpers, auto-completion, and validation.

**ZIP files**

ZIP files archive multiple files in a lossless data compression package. These files can be of various types. In XMLSpy's Archive View, ZIP files can be created, the internal structure modified, and files in the archive edited. These operations are described in the ZIP Files sub-section of this section.

**EPUB files**

An EPUB file is a zipped group of files used for the distribution of digital publications (EPUB books). In Archive View, you can open EPUB files, create and edit EPUB files, preview the digital EPUB book, edit component files of the EPUB archive directly in XMLSpy, validate the EPUB file, and save the component files back to the EPUB archive. See the section, EPUB Files, for details.
14.1 Working with OOXML Files

This section describes how to work with OOXML documents in Archive View. The following procedures are discussed:

- Creating, opening, and saving OOXML files
- Editing the structure of an OOXML file
- Opening, editing, and saving internal OOXML documents
- Intelligent editing of internal OOXML documents
- Addressing documents in OOXML files
- Comparing OOXML archives

Creating, opening, and saving OOXML files

OOXML files are created via the Create New Document dialog (File | New command), in which you select the required file type (.docx, .pptx, or .xlsx). You are prompted for a file name and a location at which to save the file. The new file is created at the specified location and then opened in Archive View (screenshot below). Notice that the basic internal structure of the OOXML document has been created.

An existing OOXML file is opened in Archive View via the Open dialog (File | Open) of XMLSpy. OOXML files are saved with the File | Save (Ctrl+S) command. This command saves the structure and relationships of the OOXML file.

Editing the structure of an OOXML file

The contents of an OOXML file can be modified by adding and deleting folders and documents to it using Archive View functionality. After these structural changes have been made, the OOXML file must be saved (File | Save) for the modifications to take effect. You should note the following points:

- When a new folder or document is added using the command buttons in Archive View, it should be named immediately on its being created. It is not possible to rename a folder or document in Archive View.
- After a new document has been added to an archive folder, it is saved to the archive by saving it in its own window or by saving the OOXML file.
Opening, editing, saving internal OOXML documents

An internal OOXML document—that is, a document within an OOXML file package—is opened from Archive View by double-clicking it, or by selecting it in the Main Window and clicking the Open document command button. The document opens in a separate XMLSpy window. After editing it, simply save the document to save it back to the OOXML archive; there is no need to save the OOXML file itself.

Intelligent editing of internal OOXML documents

XMLSpy provides intelligent editing features for internal Office Open XML documents—that is, for documents within an OOXML file package. These features include entry helpers, auto-completion, and validation.

Addressing documents in OOXML files

Documents in OOXML files can be addressed using normal file paths plus the pipe character. For example, the file path:

C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2019\Examples\Office20XX\ExcelDemo.xlsx|zip\xl\tables\table1.xml

locates the file table1.xml, which is in the xl\tables folder of the OOXML file ExcelDemo.xlsx located in the Examples\Office20XX folder of the XMLSpy examples folder.

Comparing OOXML archives

When an OOXML file is open in Archive View, you can compare it with another archive by using the command Tools | Compare Directories.
14.2 OOXML Example Files

In the Examples\Office2007 folder of your XMLSpy application folder are the following example files:

- OOXML files: (i) a Word Open XML file (.docx), (ii) an Excel Open XML file (.xlsx), and (iii) a PowerPoint Open XML file (.pptx)
- XSLT files: (i) docx2html.xslt (to convert the sample .docx file to HTML), (ii) xlsx2html.xslt (to convert the sample .xlsx file to HTML), and (iii) pptx2html.xslt (to convert the sample .pptx file to HTML)
- An XQuery file: ExcelDemo.xq (to retrieve data from the .xlsx file)

The XSLT and XQuery files are intended to demonstrate how XSLT and XQ can be used to access and transform data in OOXML files. To run the XSLT and XQuery documents, you can use any of the following options:

- Open the OOXML file in Archive View. In Folder View, select Archive and then click the menu command XSL/XQuery | XSL Transformation (for an XSLT transformation) or XSL/XQuery | XQuery Execution (for an XQuery execution). Browse for the XSLT or XQuery file and click OK.
- In the Project Window of XMLSpy, right-click the .xlsx, .pptx or .docx file in the Office2007 folder of the Examples project (screenshot below), and select the transformation command. Browse for the transformation file and click OK.
Open the XSLT or XQuery file in XMLSpy and click the menu command **XSL/XQuery** | **XSL Transformation** and **XSL/XQuery** | **XQuery Execution**, respectively. When prompted for the XML file to transform, browse for the .docx, .xlsx, or .pptx file (according to whether the XSLT/XQ document is intended for MS Word, MS Excel, or MS PowerPoint).
14.3 ZIP Files

In Archive View, you can create WinZip files, modify the internal structure of ZIP files (WinZip, WinRAR, etc), and edit files in the ZIP package directly in XMLSpy and save the files back to the ZIP archive.

Creating and saving a WinZip file

A WinZip file is created via the Create New Document dialog (File | New command), in which you select the file type *.zip*. An empty WinZip archive is created in a new window in XMLSpy (screenshot below). You must now save the ZIP file to the desired location with the File | Save (Ctrl+S) command. Add folders and files as described below, and then save the ZIP file to save your additions and changes.

![ZIP Archive](image)

An existing ZIP file is opened in Archive View via the Open dialog (File | Open) of XMLSpy.

**Note:** Creating a new ZIP file is different than creating a new OOXML file in that you are not prompted for a location to save the file before the archive is opened in Archive View. For the ZIP file to be saved from the empty archive that is opened in Archive View, you must explicitly use the File | Save (Ctrl+S) command.

Adding folders and files and modifying the archive structure

You can add folders (click the New Folder button), existing files (Add Document), and new files (Add New Document) to the selected Archive folder. Note that when you add a new folder or new document, you must immediately enter a name for the folder or file; it is not possible to rename folders or documents in Archive View.

Addressing documents in ZIP files

Documents in ZIP files can be addressed using normal file paths plus the pipe character. For example, the file path:

```
C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2019\Examples\Test.zip|zip\TestFolder\MyFile.xml
```

locates the file MyFile.xml, which is in the TestFolder folder of the ZIP file Test.zip located in the Examples folder of the XMLSpy examples folder.
Comparing ZIP archives

When a ZIP file is open in Archive View, you can compare it with another archive by using the command **Tools | Compare Directories.**
14.4 EPUB Files

An EPUB file is a zipped group of files conforming to the EPUB standard of the International Digital Publishing Forum (IDPF). This standard is the distribution and interchange standard for digital web publications. In Archive View, you can open EPUB files, view the EPUB file's digital publication in a preview tab, edit component files of the EPUB archive directly in XMLSpy, validate the EPUB file, and save the component files back to the EPUB archive.

Note: (i) XMLSpy supports EPUB 2.0.1. (ii) A sample EPUB file is available in the Examples project and in the (My) Documents/Altova/XMLSpy2019/Examples folder.

Terminology

In the descriptions below, terms are used as follows:

- **EPUB file** is used to indicate the EPUB file having the file extension .epub. This is the ZIP file that contains the whole archive and is the file that will be opened in Archive View.
- **An archive file** is any one of the files contained in the EPUB archive.
- **EPUB book** is the term used to indicate the digital publication generated by the zipped EPUB file.

In this section

The description below of EPUB functionality in XMLSpy is structured into the following parts:

- Opening EPUB files in Archive View
- Creating a new EPUB file
- Previewing an EPUB book
- Modifying the contents and structure of an EPUB archive
- Info and Settings
- Editing archive files directly in XMLSpy
- Entry helpers for archive files
- Validating EPUB file

Opening EPUB files in Archive View

Select the menu command File | Open, navigate to the EPUB file, and click Open. The EPUB file opens in Archive View (screenshot below). Alternatively, you can right-click the EPUB file in Windows Explorer and select the context menu command to open the file with XMLSpy. If you have set XMLSpy to be the default editor of EPUB files, then double-clicking the EPUB file will open the file in Archive View.
Folder View on the left-hand side shows the folders in the archive, whereas the Main Window shows the files in the folder selected in Folder View. The EPUB archive will have the following structure and the following key components.

```
Archive
|-- Mimetype file
|-- META-INF folder
    |-- container.xml
|-- DOCUMENT folder (In the screenshot above, OEBPS is the Document folder.)
    |-- Contains HTML, CSS, image files, plus OPF and NCX files
```

Creating a new EPUB file

To create a new EPUB file, select the menu command **File | New**. In the Create New Document dialog that pops up, select the file type `.epub`. In the Save As dialog that now pops up, give a name for your EPUB document and click **Save**. A skeleton EPUB archive containing all the folders and files of a valid EPUB archive (see archive structure above) will be created in a new window in Archive View. Add the folders and files you wish to add to the archive, as described below, and then save the EPUB file. To edit an archive file directly in XMLSpy, double-click the file in Archive View. The file will open in a new XMLSpy window. Edit it and then save it with the **File | Save (Ctrl+S)** command.

Previewing an EPUB book

To preview an EPUB book, make the EPUB file active in Archive View, then click the **Preview** button in the toolbar of Archive View. The EPUB book will open in a separate (Internet Explorer) browser window in XMLSpy. If any of the files that will be used for the preview—whether a content...
file or a structure-related file—has been modified but not yet saved, you will be prompted to save the file. If you do not save the modifications, the preview will use the previously saved data and might not be up-to-date. You can specify that all modified files be saved automatically before previewing by toggling on this setting (via the Settings button in the the toolbar of Archive View).

Note the following:

- If the Preview button in Archive View is clicked while a Preview window of that EPUB publication is still open, then the EPUB publication will be reloaded in the open Preview window.
- Refreshing the Preview window itself (using the Refresh (F5) command of Internet Explorer) will not update the Preview window. The EPUB publication in the Preview window must be updated using the Preview button (of Archive View) of the corresponding EPUB file (see previous point).
- To close the preview, close the Preview window.

Note: Not all EPUB markup is supported in Internet Explorer, so previews could be distorted. Additionally, if the digital publication document is XML—and not HTML—the preview might not work. Newer versions of Internet Explorer provide improved handling of EPUB markup, so if you experience problems, try updating to the latest version of Internet Explorer.

Modifying the contents and structure of an EPUB archive

You can add folders (click the New Folder button), new files (Add New Document), and existing files (Add Document) to the selected archive folder. Note that when you add a new folder or new document, you must immediately enter a name for the folder or file; it is not possible to rename folders in Archive View. You can delete a file or folder by selecting it and clicking the Delete from Archive button.

After you have modified the archive you must save the EPUB file (File | Save) for the changes to be saved.

Info and Settings

Clicking the Info button displays, at the bottom of Archive View, a summary of key archive information (screenshot below). Clicking the Info button again removes the summary. The summary reports the number of files in the archive (including the Mimetype file and container.xml), the size of the compressed EPUB file, and the cumulative size of the uncompressed files.

The Settings button contains drops down two automatic file-saving options that can be toggled on and off: to automatically save the EPUB file (i) before validation, and (ii) before previewing the EPUB file in (via the Preview button) in XMLSpy.
Editing archive files directly in XMLSpy

To edit an archive file directly in XMLSpy, double-click the file in Archive View. Alternatively, select the file in Archive View and click the Open Document button in the toolbar of Archive View. The file will open in a new XMLSpy window. Edit it and then save it with the File | Save (Ctrl+S) command.

Entry helpers for archive files

Entry helpers for standards-based archive files are available when these archive files are opened in XMLSpy. These archive files are:

- The OPF file, traditionally named content.opf, contains the EPUB book's metadata. It is based on the Open Packaging Format (OPF) specification.
- The NCX file (Navigation Control file for XML), traditionally named toc.ncx, contains the publication's table of contents. It is based on the NCX part of the OPF specification.
- The folder named META-INF must contain the file container.xml, which points to the file defining the contents of the book (the OPF file). The file container.xml specifies how the archive files should be organized according to rules in the Open Container Format (OCF) specification.

Validating an EPUB file

To validate an EPUB file, select the command XML | Validate XML (F8). The validation results are displayed in the Messages window (screenshot below). If any of the archive files—whether a content file or a structure-related file—has been modified but not yet saved, you will be prompted to save the file. You must save the modified files in order to validate the EPUB file. You can specify that all modified files be saved automatically before validation by toggling on this setting (via the Settings button in the toolbar of Archive View).

Error messages display: (i) the file in which the error was found, and, if applicable, the number of the line in which the error occurs; (ii) a description of the error. In the screenshot above, the highlighted error occurs in line 21 of the file content.opf. Clicking on the error line in the Messages window opens the relevant file and highlights the error.

Note: The EPUB validation engine is a Java utility, so Java must be installed on your machine for the validation engine to run.
# 15 Databases

XMLSpy enables you to connect to a variety of databases (DBs) and then perform operations such as querying the DB, importing the DB structure as an XML Schema, generating an XML data file from the DB, and exporting data to a DB. Each DB-related feature is available in XMLSpy as a menu command, and is described in the User Reference section of this documentation under the respective command. A complete list of these command is given below, with links to the respective descriptions.

In this section, we do the following:

- Describe how to connect to a database, which is an operation that is required for executing any of XMLSpy’s DB-related commands; and
- List DBs that have been successfully tested for use with XMLSpy.

**Note:** If you are using the 64-bit version of XMLSpy, ensure that you have access to the 64-bit database drivers needed for the specific database you are connecting to.

### XMLSpy’s DB-related features

XMLSpy’s DB-related features are executed with commands in the **DB** and **Convert** menus.

- **Query Database**: In the **DB** menu. Loads the structure of the DB in a separate Database Query window and enables queries to the DB. Results are displayed in the Database Query window.
- **IBM DB2**: In the **DB** menu. IBM DB2 is an XML DB, and XMLSpy enables management of the XML Schemas of the XML DB as well as editing and validation of the XML DB.
- **Oracle XML DB**: In the **DB** menu. Provides a range of functionality for Oracle XML DBs, including XML Schema management, database querying, and generation of XML files based on DB schemas.
- **Import Database Data**: In the **Convert** menu. Imports DB data into an XML file.
- **Create XML Schema from DB Structure**: In the **Convert** menu. Generates an XML Schema that is based on the structure of the DB.
- **DB Import Based on XML Schema**: In the **Convert** menu. With an XML Schema document active in XMLSpy, a DB connection is made and the data of a selected DB table can be imported. The resulting XML document will have a structure based on the XML Schema that was active when the DB connection was made.
- **Create DB Structure from XML Schema**: In the **Convert** menu. DB tables with no data are created based on the structure of an existing XML Schema.
- **Export to Database**: In the **Convert** menu. Data from an XML document can be exported to a DB. Existing DB tables can be updated with the XML data, or new tables can be created that contain the XML data.

### Datatype conversions

When converting data between XML documents and DBs, datatypes must necessarily be converted to types appropriate for the respective formats. The way XMLSpy converts datatypes is given in the appendices **Datatypes in DB-Generated XML Schemas** and **Datatypes in DBs Generated from XML Schemas**.
Altova DatabaseSpy

Altova’s DatabaseSpy is a multi-database query and DB design tool that offers additional DB functionality to that available in XMLSpy. For more details about Altova DatabaseSpy, visit the Altova website.
15.1 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 XMLSpy 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, XMLSpy 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 XMLSpy (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, XMLSpy 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 XMLSpy. The available features, performance parameters, and the known issues will typically vary based on the data access technology or drivers used.

### 15.1.1 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 start the database connection wizard:

- On the **DB** menu, click **Query 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
15.1.2 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 IBMNDARLA OLE DB Provider</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• IBM DB2 for i5/OS IBMDA400 SQLJ Driver</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 Jet 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>
</tbody>
</table>
## Connecting to a Database

<table>
<thead>
<tr>
<th>Database</th>
<th>Interface</th>
<th>Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDBC</td>
<td></td>
<td>Connector/J [<a href="https://dev.mysql.com/downloads/connector/j/">https://dev.mysql.com/downloads/connector/j/</a>]</td>
</tr>
<tr>
<td>ODBC</td>
<td></td>
<td>Connector/ODBC [<a href="https://dev.mysql.com/downloads/connector/odbc/">https://dev.mysql.com/downloads/connector/odbc/</a>]</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] |
|          | 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] |
|          | ODBC      | psqLODBC [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] |
|          | ODBC      | ODBC Connector [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 |
| Teradata | ADO.NET   | .NET Data Provider for Teradata [https://downloads.teradata.com/download/connectivity/net-data-provider-for-teradata] |
|          | JDBC      | Teradata JDBC Driver [https://downloads.teradata.com/download/connectivity/jdbc-driver] |
|          | ODBC      | Teradata ODBC Driver for Windows [https://downloads.teradata.com/download/connectivity/odbc-driver/windows] |
15.1.3 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.
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><strong>Microsoft Access</strong></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 <strong>Microsoft Jet OLE DB Provider</strong>.</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

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. Select Use an existing MS Access database.
4. Browse for the database file, or enter the path to it (either relative or absolute).
5. Click Connect.

Creating a New Microsoft Access Database

As an alternative to connecting to an existing database file, you can create a new Microsoft Access database file (.accdb, .mdb) and connect to it, even if Microsoft Access is not installed on the computer. The database file created by XMLSpy is empty. To create the required database structure, use Microsoft Access or a tool such as DatabaseSpy (https://www.altova.com/databasespy).

To create a new 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. Select **Create a new MS Access database**, and then enter the path (either relative or absolute) of the database file to be created (for example, `c:\users\public\products.mdb`). Alternatively, click **Browse** to select a folder, type the name of the database file in the “File name” text box (for example, `products.mdb`), and click **Save**.

   **Notes**
   - Make sure that you have write permissions to the folder where you want to create the database file.
   - The database file name must have the `.mdb` or `.accdb` extension.

4. Click **Connect**.

**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.
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>

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.
### Data Link Properties dialog box

<table>
<thead>
<tr>
<th>Property</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Source</strong></td>
<td>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</td>
</tr>
<tr>
<td><strong>Jet OLEDB:System Database</strong></td>
<td>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 &quot;workgroup information file&quot; 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.</td>
</tr>
</tbody>
</table>
527  Databases  Connecting to a Database

<table>
<thead>
<tr>
<th>Property</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Jet OLEDB:Database</strong></td>
<td>If the database is password-protected, set the value of this property to the database password.</td>
</tr>
</tbody>
</table>

15.1.4 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 XMLSpy 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 XMLSpy 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.
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.

5. Click **Connect**.
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 XMLSpy, you will need to replace the asterisk (*) characters with the actual password.
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

```
Database=PRODUCTS;UID=user;Password=password;Server=localhost:50000;
```
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.

See also: https://www-01.ibm.com/support/knowledgecenter/en/SSEP00_10.1.0/com.ibm.swg.im.dbclient.adonet.ref.doc/doc/DB2ConnectionClassConnectionStringProperty.html

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

ADO.NET Support Notes
The following table lists known ADO.NET database drivers that are currently not supported or have limited support in XMLSpy.

<table>
<thead>
<tr>
<th>Database</th>
<th>Driver</th>
<th>Support notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>All databases</td>
<td>.Net Framework Data Provider for ODBC</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>.Net Framework Data Provider for OleDb</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>Firebird ADO.NET Data Provider</td>
<td>Limited support. It is recommended to use ODBC or JDBC instead.</td>
</tr>
<tr>
<td>Informix</td>
<td>IBM Informix Data Provider for .NET Framework 4.0</td>
<td>Not supported. Use DB2 Data Server Provider instead.</td>
</tr>
</tbody>
</table>
### 15.1.5 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 XMLSpy. 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 XMLSpy. 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 XMLSpy 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).
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.

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).

15.1.6 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. XMLSpy 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; b) The JVM path found in the Windows registry; c) The JAVA_HOME environment variable.
- Make sure that the platform of XMLSpy (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).

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>
<tr>
<td>IBM Informix</td>
<td>jdbc:informix-sqli://hostName:port/</td>
</tr>
<tr>
<td></td>
<td>databaseName:INFORMIXSERVER=myserver</td>
</tr>
<tr>
<td>MariaDB</td>
<td>jdbc:mariadb://hostName:port/databaseName</td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>jdbc:sqlserver://hostName:port;databaseName=name</td>
</tr>
<tr>
<td>MySQL</td>
<td>jdbc:mysql://hostName:port/databaseName</td>
</tr>
<tr>
<td>Oracle</td>
<td>jdbc:oracle:thin:@&lt;hostName&gt;:port:service</td>
</tr>
<tr>
<td>Oracle XML DB</td>
<td>jdbc:oracle:oci:@&lt;hostName&gt;:port:service</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>jdbc:postgresql://hostName:port/databaseName</td>
</tr>
<tr>
<td>Progress OpenEdge</td>
<td>jdbc:datadirect:opendebug://host:port;databaseName=db_name</td>
</tr>
<tr>
<td>Sybase</td>
<td>jdbc:sybase:Tds:hostName:port/databaseName</td>
</tr>
<tr>
<td>Teradata</td>
<td>jdbc:teradata://databaseServerName</td>
</tr>
</tbody>
</table>

**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.

**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 system.
operating system.

<table>
<thead>
<tr>
<th>Database</th>
<th>Sample CLASSPATH entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firebird</td>
<td><code>C:\Program Files\Firebird\Jaybird-2.2.8-JDK_1.8\jaybird-full-2.2.8.jar</code></td>
</tr>
<tr>
<td>IBM DB2</td>
<td><code>C:\Program Files (x86)\IBM\SQLLIB\java\db2jcc.jar;C:\Program Files (x86)\IBM\SQLLIB\java\db2jcc_license_cu.jar;</code></td>
</tr>
<tr>
<td>IBM DB2 for i</td>
<td><code>C:\jt400\jt400.jar;</code></td>
</tr>
<tr>
<td>IBM Informix</td>
<td><code>C:\Informix_JDBC_Driver\lib\ifxjdbc.jar;</code></td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td><code>C:\Program Files\Microsoft JDBC Driver 4.0 for SQL Server\sqljdbc_4.0\enu\sqljdbc.jar</code></td>
</tr>
<tr>
<td>MariaDB</td>
<td><code>&lt;installation directory&gt;\mariadb-java-client-2.2.0.jar</code></td>
</tr>
<tr>
<td>MySQL</td>
<td><code>&lt;installation directory&gt;\mysql-connector-java-version-bin.jar;</code></td>
</tr>
<tr>
<td>Oracle</td>
<td><code>ORACLE_HOME\jdbc\lib\ojdbc6.jar;</code></td>
</tr>
<tr>
<td>Oracle (with XML DB)</td>
<td><code>ORACLE_HOME\jdbc\lib\ojdbc6.jar;ORACLE_HOME\LIB\xmlparserv2.jar;ORACLE_HOME\RDBMS\jlib\xdb.jar;</code></td>
</tr>
<tr>
<td>PostgreSQL</td>
<td><code>&lt;installation directory&gt;\postgresql.jar</code></td>
</tr>
<tr>
<td>Progress OpenEdge</td>
<td><code>%DLC%\java\openedge.jar;%DLC%\java\pool.jar;</code> Note: Assuming the Progress OpenEdge SDK is installed on the machine, <code>%DLC%</code> is the directory where OpenEdge is installed.</td>
</tr>
<tr>
<td>Sybase</td>
<td><code>C:\sybase\jConnect-7.0\classes\jconn4.jar</code></td>
</tr>
<tr>
<td>Teradata</td>
<td><code>&lt;installation directory&gt;\tdgssconfig.jar;&lt;installation directory&gt;\terajdbc4.jar</code></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 ( ; ).

### 15.1.7 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.

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 (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.

15.1.8 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 XMLSpy, you do not need to install any drivers to connect to them.

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. Select Use an existing SQLite database, and then 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. Optionally, select the Disable Foreign Keys check box, see Foreign Key Constraints.
5. Click Connect.

Creating a New SQLite Database

You can create a new SQLite database file and connect to it, as an alternative to connecting to an existing database file. The database file created by XMLSpy is empty; use queries or scripts to create the required database structure and populate it with data.

To create a new SQLite database:

1. Run the database connection wizard (see Starting the Database Connection Wizard).
2. Select SQLite, and then click Next.
3. Select **Create a new SQLite database**, and then enter the path (either relative or absolute) of the database file to be created (for example, `c:\users\public\products.sqlite`). Alternatively, click **Browse** to select a folder, type the name of the database file in the "File name" text box (for example, `products.sqlite`), and click **Save**.

Make sure that you have write permissions to the folder where you want to create the database file.

4. Optionally, select the **Disable Foreign Keys** check box, see **Foreign Key Constraints**.
5. Click **Connect**.

### Foreign Key Constraints

When you connect to an existing SQLite database from XMLSpy, or when you create a new one, foreign key constraints are enabled by default. Foreign key constraints help preserve the integrity of data in your database. For example, when foreign keys are enabled, it is not possible to delete a record from a table if it has dependencies in another table.

In certain cases, you may want to temporarily override this behavior and disable foreign keys, perhaps, in order to update or insert multiple rows of data without getting data validation errors. To explicitly disable foreign keys before connecting to the SQLite database, select the **Disable Foreign Keys** option available on the database connection wizard.
When foreign keys are disabled, you will be able to perform operations against data that would otherwise not be possible due to validation checks. At the same time, however, there is also the risk of introducing incorrect data in the database, or creating "orphaned" rows. (An example of an "orphaned" row would be an address in the "addresses" table not linked to any person in the "person" table, because the person was deleted but its associated address was not.)

15.1.9 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.

### 15.1.10 Database Connection Examples

This section includes sample procedures for connecting to a database from XMLSpy. 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 XMLSpy.

**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 XMLSpy.
- 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.
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).
5. Enter the database connection details as follows:

<table>
<thead>
<tr>
<th>Data Source Name (DSN)</th>
<th>Enter a descriptive name for the data source you are creating.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</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 firebirdserv, and the database alias is products, as follows: firebirdserv:products</td>
</tr>
</tbody>
</table>

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
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. XMLSpy 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; b) The JVM path found in the Windows registry; c) The JAVA_HOME environment variable.
- Make sure that the platform of XMLSpy (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).

6. Click OK.
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:firebirdsql://<host>[:<port>]/<database path or alias>

7. Click Connect.

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.

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)

**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**.

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**.

**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. XMLSpy 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**; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.
- Make sure that the platform of XMLSpy (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.

```
jdbc:informix-sqli://hostName:port/databaseName:INFORMIXSERVER=myserver;
```

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.

   ![Connecting to MariaDB](image)

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.

   ![Create a new Data Source to MariaDB](image)

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:

![Connection test](image)

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.

**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.

**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.
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, **Nanornull**).

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**.

**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,
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).

   If you installed XMLSpy 64-bit, then the 64-bit ODBC drivers are shown in the list. Otherwise, the 32-bit ODBC drivers are shown. See also Viewing the Available ODBC Drivers.

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.

**Connecting to Oracle (ODBC)**

This example illustrates a common scenario where you connect from XMLSpy 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 XMLSpy. 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:

```sql
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%28v=vs.85%29.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.

**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. XMLSpy 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; b) The JVM path found in the Windows registry; c) The JAVA_HOME environment variable.
- Make sure that the platform of XMLSpy (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.
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:**

- `psql/ODBC` 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.

5. Click User DSN.

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.
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 XMLSpy, 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), and then click OK. To verify connectivity before saving the entered data, click Test Connect.

![ODBC Progress OpenEdge Wire Protocol Driver Setup](image)

7. Click OK. The new data source now appears in the list of ODBC data sources.
8. Click **Connect**.

**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. XMLSpy 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**; b) The JVM path found in the Windows registry; c) The **JAVA_HOME** environment variable.
- Make sure that the platform of XMLSpy (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).

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.
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. XMLSpy 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; b) The JVM path found in the Windows registry; c) The JAVA_HOME environment variable.
- Make sure that the platform of XMLSpy (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:sybase:Tds:hostName:port/databaseName
```

7. Click Connect.

**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 XMLSpy 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 XMLSpy 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.

---

**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. XMLSpy 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**; b) The JVM path found in the Windows registry; c) The **JAVA_HOME** environment variable.
- Make sure that the platform of XMLSpy (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).
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.

7. Click **Connect**.
## 15.2 Supported Databases

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></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. In Authentic view, data coming from a SQLite database is not editable. When you attempt to save SQLite data from the Authentic view, a message box will inform you of this known limitation.</td>
</tr>
<tr>
<td>Sybase ASE 15, 16</td>
<td></td>
</tr>
<tr>
<td>Teradata 16</td>
<td></td>
</tr>
</tbody>
</table>
16 Altova Global Resources

Altova Global Resources is a collection of aliases for file, folder, and database resources. Each alias can have multiple configurations, and each configuration maps to a single resource (see screenshot below). Therefore, when a global resource is used as an input, the global resource can be switched among its configurations. This is done easily via controls in the GUI that let you select the active configuration. For example, if an XSLT stylesheet for transforming an XML document is assigned via a global resource (an alias), then we can set up multiple configurations for the global resource, each of which points to a different XSLT file. After setting up the global resource in this way, switching the configuration would switch the XSLT file used for the transformation.

A global resource can not only be used to switch resources within an Altova application, but also to generate and use resources from other Altova applications. So, files can be generated on-the-fly in one Altova application for use in another Altova application. All of this tremendously eases and speeds up development and testing. For example, an XSLT stylesheet in XMLSpy can be used to transform an XML file generated on-the-fly by an Altova MapForce mapping.

Using Altova Global Resources involves two processes:

- **Defining Global Resources:** Resources are defined and the definitions are stored in an XML file. These resources can be shared across multiple Altova applications.
- **Using Global Resources:** Within XMLSpy, files can be located via a global resource instead of via a file path. The advantage is that the resource can be switched by changing the active configuration in XMLSpy.

Global resources in other Altova products
Currently, global resources can be defined and used in the following individual Altova products: XMLSpy, StyleVision, MapForce, Authentic Desktop, MobileTogether Designer, and DatabaseSpy.
Defining Global Resources

Altova Global Resources are defined in the Manage Global Resources dialog, which can be accessed in two ways:

- Click the menu command **Tools | Global Resources**.
- Click the **Manage Global Resources** icon in the Global Resources toolbar (screenshot below).

The Global Resources Definitions file

Information about global resources is stored in an XML file called the Global Resources Definitions file. This file is created when the first global resource is defined in the Manage Global Resources dialog (screenshot below) and saved.

When you open the Manage Global Resources dialog for the first time, the default location and name of the Global Resources Definitions file is specified in the **Definitions File** text box (see screenshot above):

```
C:\Users\<username>\My Documents\Altova\GlobalResources.xml
```

This file is set as the default Global Resources Definitions file for all Altova applications. So a global resource can be saved from any Altova application to this file and will be immediately available to all other Altova applications as a global resource. To define and save a global resource to the Global Resources Definitions file, add the global resource in the Manage Global Resources dialog and click **OK** to save.

To select an already existing Global Resources Definitions file to be the active definitions file of a particular Altova application, browse for it via the **Browse** button of the **Definitions File** text box.
Note: You can name the Global Resources Definitions file anything you like and save it to any location accessible to your Altova applications. All you need to do in each application, is specify this file as the Global Resources Definitions file for that application (in the Definitions File text box). The resources become global across Altova products when you use a single definitions file across all Altova products.

Note: You can also create multiple Global Resources Definitions files. However, only one of these can be active at any time in a given Altova application, and only the definitions contained in this file will be available to the application. The availability of resources can therefore be restricted or made to overlap across products as required.

Managing global resources: adding, editing, deleting, saving

In the Manage Global Resources dialog (screenshot above), you can add a global resource to the selected Global Resources Definitions file, or edit or delete a selected global resource. The Global Resources Definitions file organizes the global resources you add into groups: of files, folders, and databases (see screenshot above).

To add a global resource, click the Add button and define the global resource in the appropriate Global Resource dialog that pops up (see the descriptions of files, folders, and databases in the sub-sections of this section). After you define a global resource and save it (by clicking OK in the Manage Global Resources dialog), the global resource is added to the library of global definitions in the selected Global Resources Definitions file. The global resource will be identified by an alias.

To edit a global resource, select it and click Edit. This pops up the relevant Global Resource dialog, in which you can make the necessary changes (see the descriptions of files, folders, and databases in the sub-sections of this section).

To delete a global resource, select it and click Delete.

After you finish adding, editing, or deleting, make sure to click OK in the Manage Global Resources dialog to save your modifications to the Global Resources Definitions file.

Relating global resources to alias names via configurations

Defining a global resource involves mapping an alias name to a resource (file, folder, or database). A single alias name can be mapped to multiple resources. Each mapping is called a configuration. A single alias name can therefore be associated with several resources via different configurations (screenshot below).
In an Altova application, you can then assign aliases instead of files. For each alias you can switch between the resources mapped to that alias simply by changing the application’s active Global Resource configuration (active configuration). For example, in Altova’s XMLSpy application, if you wish to run an XSLT transformation on the XML document MyXML.xml, you can assign the alias MyXSLT to it as the global resource to be used for XSLT transformations. In XMLSpy you can then change the active configuration to use different XSLT files. If Configuration-1 maps First.xslt to MyXSLT and Configuration-1 is selected as the active configuration, then First.xslt will be used for the transformation. In this way multiple configurations can be used to access multiple resources via a single alias. This mechanism can be useful when testing and comparing resources. Furthermore, since global resources can be used across Altova products, resources can be tested and compared across multiple Altova products as well.

16.1.1 Files

The Global Resource dialog for Files (screenshot below) is accessed via the Add | Files command in the Manage Global Resources dialog. In this dialog, you can define configurations of the alias that is named in the Resource Alias text box. After specifying the properties of the configurations as explained below, save the alias definition by clicking OK.

After saving an alias definition, you can add another alias by repeating the steps given above (starting with the Add | Files command in the Manage Global Resources dialog).

Global Resource dialog

An alias is defined in the Global Resource dialog (screenshot below).
Global Resource dialog icons

- **Add Configuration**: Pops up the Add Configuration dialog in which you enter the name of the configuration to be added.

- **Add Configuration as Copy**: Pops up the Add Configuration dialog in which you can enter the name of the configuration to be created as a copy of the selected configuration.

- **Delete**: Deletes the selected configuration.

- **Open**: Browse for the file to be created as the global resource.

**Defining the alias**

Define the alias (its name and configurations) as follows:

1. **Give the alias a name**: Enter the alias name in the Resource Alias text box.
2. **Add configurations**: The Configurations pane will have, by default, a configuration named
Default (see screenshot above), which cannot be deleted or renamed. You can add as many additional configurations as you like by: (i) clicking the Add Configuration or Add Configuration as Copy icons, and (ii) giving the configuration a name in the dialog that pops up. Each added configuration will be shown in the Configurations list. In the screenshot above, two additional configurations, named Long and Short, have been added to the Configurations list. The Add Configuration as Copy command enables you to copy the selected configuration and then modify it.

3. **Select a resource type for each configuration:** Select a configuration from the Configurations list, and, in the Settings for Configuration pane, specify a resource for the configuration: (i) File, (ii) Output of an Altova MapForce transformation, or (iii) Output of an Altova StyleVision transformation. Select the appropriate radio button. If a MapForce or StyleVision transformation option is selected, then a transformation is carried out by MapForce or StyleVision using, respectively, the .mfd or .sps file and the respective input file. The result of the transformation will be the resource.

4. **Select a file for the resource type:** If the resource is a directly selected file, browse for the file in the Resource File Selection text box. If the resource is the result of a transformation, in the File Selection text box, browse for the .mfd file (for MapForce transformations) or the .sps file (for StyleVision transformations). Where multiple inputs or outputs for the transformation are possible, a selection of the options will be presented. For example, the output options of a StyleVision transformation are displayed according to what edition of StyleVision is installed (the screenshot below shows the outputs for Enterprise Edition).

![Outputs (use radio button to select)](image)

Select the radio button of the desired option (in the screenshot above, 'HTML output' is selected). If the resource is the result of a transformation, then the output can be saved as a file or itself as a global resource. Click the icon and select, respectively, Global Resource (for saving the output as a global resource) or Browse (for saving the output as a file). If neither of these two saving options is selected, the transformation result will be loaded as a temporary file when the global resource is invoked.

5. **Define multiple configurations if required:** You can add more configurations and specify a resource for each. Do this by repeating Steps 3 and 4 above for each configuration. You can add a new configuration to the alias definition at any time.

6. **Save the alias definition:** Click OK to save the alias and all its configurations as a global resource. The global resource will be listed under Files in the Manage Global Resources dialog.

**Result of MapForce transformation**

Altova MapForce maps one or more (existing) input document schemas to one or more (new) output document schemas. This mapping, which is created by a MapForce user, is known as a MapForce Design (MFD). XML files, text files, databases, etc, that correspond to the input schema/s can be used as data sources. MapForce generates output data files that correspond to
the output document schema. This output document is the Result of MapForce Transformation file that will become a global resource.

If you wish to set a MapForce-generated data file as a global resource, the following must be specified in the Global Resource dialog (see screenshot below):

- **A .mfd (MapForce Design) file.** You must specify this file in the Resource will point to generated output of text box (see screenshot above).
- **One or more input data files.** After the MFD file has been specified, it is analyzed and, based on the input schema information in it, default data file/s are entered in the Inputs pane (see screenshot above). You can modify the default file selection for each input schema by specifying another file.
- **An output file.** If the MFD document has multiple output schemas, all these are listed in the Outputs pane (see screenshot above) and you must select one of them. If the output file location of an individual output schema is specified in the MFD document, then this file location is entered for that output schema in the Outputs pane. From the screenshot above we can see that the MFD document specifies that the Customers output schema has a default XML data file (CustomersOut.xml), while the Text file output schema does not have a file association in the MFD file. You can use the default file location in the Outputs pane or specify one yourself. The result of the MapForce transformation will be saved to the file location of the selected output schema. This is the file that will be used as the global resource.

**Note:** The advantage of this option (Result of MapForce transformation) is that the transformation is carried out at the time the global resource is invoked. This means that the global resource will contain the most up-to-date data (from the input file/s).
Note: Since MapForce is used to run the transformation, you must have Altova MapForce installed for this functionality to work.

Result of StyleVision transformation
Altova StyleVision is used to create StyleVision Power Stylesheet (SPS) files. These SPS files generate XSLT stylesheets that are used to transform XML documents into output documents in various formats (HTML, PDF, RTF, Word 2007+, etc). If you select the option Result of StyleVision Transformation, the output document created by StyleVision will be the global resource associated with the selected configuration.

For the StyleVision Transformation option in the Global Resource dialog (see screenshot below), the following files must be specified.

- A .sps (SPS) file. You must specify this file in the Resource will point to generated output of text box (see screenshot above).
- Input file/s. The input file might already be specified in the SPS file. If it is, it will appear automatically in the Inputs pane once the SPS file is selected. You can change this entry. If there is no entry, you must add one.
- Output file/s. Select the output format in the Outputs pane, and specify an output file location for that format.
Note: The advantage of this option (Result of StyleVision transformation) is that the transformation is carried out when the global resource is invoked. This means that the global resource will contain the most up-to-date data (from the input file/s).

Note: Since StyleVision is used to run the transformation, you must have Altova StyleVision installed for this functionality to work.

16.1.2 Folders
In the Global Resource dialog for Folders (screenshot below), add a folder resource as described below.

Global Resource dialog icons

- Add Configuration: Pops up the Add Configuration dialog in which you enter the name of the configuration to be added.
- Add Configuration as Copy: Pops up the Add Configuration dialog in which you can enter the name of the configuration to be created as a copy of the selected configuration.
- Delete: Deletes the selected configuration.
- Open: Browse for the folder to be created as the global resource.
Defining the alias

Define the alias (its name and configurations) as follows:

1. **Give the alias a name:** Enter the alias name in the Resource Alias text box.
2. **Add configurations:** The Configurations pane will have a configuration named Default (see screenshot above). This Default configuration cannot be deleted nor have its name changed. You can enter as many additional configurations for the selected alias as you like. Add a configuration by clicking the Add Configuration or Add Configuration as Copy icons. In the dialog which pops up, enter the configuration name. Click OK. The new configuration will be listed in the Configurations pane. Repeat for as many configurations as you want.
3. **Select a folder as the resource of a configuration:** Select one of the configurations in the Configurations pane and browse for the folder you wish to create as a global resource. If security credentials are required to access a folder, then specify these in the Username and Password fields.
4. **Define multiple configurations if required:** Specify a folder resource for each configuration you have created (that is, repeat Step 3 above for the various configurations you have created). You can add a new configuration to the alias definition at any time.
5. **Save the alias definition:** Click OK in the Global Resource dialog to save the alias and all its configurations as a global resource. The global resource will be listed under Folders in the Manage Global Resources dialog.

### 16.1.3 Databases

In the Global Resource dialog for Databases (screenshot below), you can add a database resource as follows:
Global Resource dialog icons

- **Add Configuration**: Pops up the Add Configuration dialog in which you enter the name of the configuration to be added.
- **Add Configuration as Copy**: Pops up the Add Configuration dialog in which you can enter the name of the configuration to be created as a copy of the selected configuration.
- **Delete**: Deletes the selected configuration.

Defining the alias

Define the alias (its name and configurations) as follows:

1. **Give the alias a name**: Enter the alias name in the Resource Alias text box.
2. **Add configurations**: The Configurations pane will have a configuration named Default (see
screenshot above). This Default configuration cannot be deleted nor have its name changed. You can enter as many additional configurations for the selected alias as you like. Add a configuration by clicking the Add Configuration or Add Configuration as Copy icons. In the dialog which pops up, enter the configuration name. Click OK. The new configuration will be listed in the Configurations pane. Repeat for as many configurations as you want.

3. Start selection of a database as the resource of a configuration: Select one of the configurations in the Configurations pane and click the Choose Database icon. This pops up the Create Global Resources Connection dialog.

4. Connect to the database: Select whether you wish to create a connection to the database using the Connection Wizard, an existing connection, an ADO Connection, an ODBC Connection, or JDBC Connection. Complete the definition of the connection method as described in the section Connecting to a Database. You can use either the Connection Wizard, ADO Connections, or ODBC Connections. If a connection has already been made to a database from XMLSpy, you can click the Existing Connections icon and select the DB from the list of connections that is displayed.

5. Select the root object: If you connect to a database server where a root object can be selected, you will be prompted, in the Choose Root Object dialog (screenshot below), to select a root object on the server. Select the root object and click Set Root Object. The root object you select will be the root object that is loaded when this configuration is used.

If you choose not to select a root object (by clicking the Skip button), then you can select the root object at the time the global resource is loaded.

6. Define multiple configurations if required: Specify a database resource for any other configuration you have created (that is, repeat Steps 3 to 5 above for the various configurations you have created). You can add a new configuration to the alias definition at any time.

7. Save the alias definition: Click OK in the Global Resource dialog to save the alias and all its configurations as a global resource. The global resource will be listed under databases in the Manage Global Resources dialog.
16.2 Using Global Resources

There are several types of global resources (file-type, folder-type, and database-type). Some scenarios in which you can use global resources in XMLSpy are listed here: Files and Folders and Databases.

Selections that determine which resource is used

There are two application-wide selections that determine what global resources can be used and which global resources are actually used at any given time:

- The active Global Resources XML File is selected in the Global Resource dialog. The global-resource definitions that are present in the active Global Resources XML File are available to all files that are open in the application. Only the definitions in the active Global Resources XML File are available. The active Global Resources XML File can be changed at any time, and the global-resource definitions in the new active file will immediately replace those of the previously active file. The active Global Resources XML File therefore determines: (i) what global resources can be assigned, and (ii) what global resources are available for look-up (for example, if a global resource in one Global Resource XML File is assigned but there is no global resource of that name in the currently active Global Resources XML File, then the assigned global resource (alias) cannot be looked up).

- The active configuration is selected via the menu item Tools | Active Configuration or via the Global Resources toolbar. Clicking this command (or drop-down list in the toolbar) pops up a list of configurations across all aliases. Selecting a configuration makes that configuration active application-wide. This means that wherever a global resource (or alias) is used, the resource corresponding to the active configuration of each used alias will be loaded. The active configuration is applied to all used aliases. If an alias does not have a configuration with the name of the active configuration, then the default configuration of that alias will be used. The active configuration is not relevant when assigning resources; it is significant only when the resources are actually used.

16.2.1 Assigning Files and Folders

File-type and folder-type global resources are assigned differently. In any one of the usage scenarios below, clicking the Switch to Global Resources button displays the Open Global Resource dialog (screenshot below).
Manage Global Resources: Displays the Manage Global Resources dialog.

Selecting a file-type global resource assigns the file. Selecting a folder-type global resource causes an Open dialog to open, in which you can browse for the required file. The path to the selected file is entered relative to the folder resource. So if a folder-type global resource were to have two configurations, each pointing to different folders, files having the same name but in different folders could be targeted via the two configurations. This could be useful for testing purposes.

You can switch to the file dialog or the URL dialog by clicking the respective button at the bottom of the dialog. The Manage Global Resources icon in the top right-hand corner pops up the Manage Global Resources dialog.

Usage scenarios

File-type and folder-type global resources can be used in the following scenarios:

- Opening global resources
- Saving as global resource
- Assigning files for XSLT transformations
- XSLT transformation
- XQuery executions
- Assigning an SPS
Opening global resources

A global resource can be opened in XMLSpy with the **File | Open (Switch to Global Resource)** command and can be edited. In the case of a file-type global resource, the file is opened directly. In the case of a folder-type global resource, an Open dialog pops up with the associated folder selected. You can then browse for the required file in descendant folders. One advantage of addressing files for editing via global resources is that related files can be saved under different configurations of a single global resource and accessed merely by changing configurations. Any editing changes would have to be saved before changing the configuration.

Saving as global resource

A newly created file can be saved as a global resource. Also, an already existing file can be opened and then saved as a global resource. When you click the **File | Save** or **File | Save As** commands, the Save dialog appears. Click the **Switch to Global Resource** button to access the available global resources (screenshot below), which are the aliases defined in the current Global Resources XML File.

![Save As dialog](screenshot)

Select an alias and then click **Save**. If the alias is a **file alias**, the file will be saved directly. If the alias is a **folder alias**, a dialog will appear that prompts for the name of the file under which the file is to be saved. In either case the file will be saved to the location that was defined for the **currently active configuration**.

**Note:** Each configuration points to a specific file location, which is specified in the definition of that configuration. If the file you are saving as a global resource does not have the same filetype extension as the file at the current file location of the configuration, then there might be editing and validation errors when this global resource is opened in XMLSpy. This is because XMLSpy will open the file assuming the filetype specified in the definition
Assigning files for XSLT transformations

XSLT files can be assigned to XML documents and XML files to XSLT documents via global resources. When the commands for assigning XSLT files (XSL/XQuery | Assign XSL and XSL/XQuery | Assign XSL:FO) and XML files (XSL/XQuery | Assign Sample XML) are clicked the assignment dialog pops up. Clicking the Browse button pops up the Open dialog, in which you can switch to the Open Global Resource dialog and select the required global resource. A major advantage of using a global resource to specify files for XSLT transformations is that the XSLT file (or XML file) can be changed for a transformation merely by changing the active configuration in XMLSpy; no new file assignments have to be made each time a transformation is required with a different file. When an XSLT transformation is started, it will use the file/s associated with the active configuration.

XSLT transformations and XQuery executions

Clicking the command XSL/XQuery | XSL Transformation or XSL/XQuery | XSL:FO Transformation or XSL/XQuery | XQuery Execution pops up a dialog in which you can browse for the required XSLT, XQuery, or XML file. Click the Browse button and then the Switch to Global Resource button to pop up the Open Global Resource dialog (screenshot at top of section). The file that is associated with the currently active configuration of the selected global resource is used for the transformation.

Assigning an SPS

When assigning a StyleVision stylesheet to an XML file (Authentic | Assign StyleVision Stylesheet), you can select a global resource to locate the stylesheet. Click the Browse button and then the Switch to Global Resource button to pop up the Open Global Resource dialog (screenshot at top of section). With a global resource selected as the assignment, the Authentic View of the XML document can be changed merely by changing the active configuration in XMLSpy.

16.2.2 Assigning Databases

When a command is executed that imports data or a data structure (as an XML Schema) from a DB into XMLSpy (for example, with the Convert | Import Database Data command), you can select the option to use a global resource (screenshot below). Other commands where a database-type global resource can be used are database-related commands in the menu.
In the Connection dialog (screenshot above), all the database-type global resources that have been defined in the currently active Global Resources XML File are displayed. Select the required global resource and click Connect. If the selected global resource has more than one configuration, then the database resource for the currently active configuration is used (check Tools | Active Configuration or the Global Resources toolbar), and the connection is made. You must now select the data structures and data to be used as described in Creating an XML Schema from a DB and Importing DB data.

16.2.3 Changing the Active Configuration

One configuration of a global resource can be active at any time. This configuration is called the active configuration, and it is active application-wide. This means that the active configuration is active for all global resources aliases in all currently open files and data source connections. If an alias does not have a configuration with the name of the active configuration, then the default configuration of that alias will be used. As an example of how to change configurations, consider the case in which a file has been assigned via a global resource with multiple configurations. Each configuration maps to a different file. So, which file is selected depends on which configuration is selected as the application's active configuration.

Switching the active configuration can be done in the following ways:
• Via the menu command **Tools | Active Configuration.** Select the configuration from the command's submenu.
• In the combo box of the Global Resources toolbar (*screenshot below*), select the required configuration.

![Default](<image-url>)

In this way, by changing the active configuration, you can change source files that are assigned via a global resource.
A project is a collection of files that are related to each other in some way you determine. For example, in the screenshot below, a project named Examples collects the files for various examples in separate example folders, each of which can be organized further into sub-folders. Within the Examples project, for instance, the OrgChart example folder is organized further into sub-folders for XML, XSL, and Schema files.

Projects thus enable you to gather together files that are used together and to access them quicker. Additionally, you can define schemas and XSLT files for individual folders, thus enabling the batch processing of files in a folder.

This section describes how to create and edit projects and how to use projects.
17.1 Creating and Editing Projects

Projects are managed via the Project Window (screenshot below) and the Project menu. One project can be open at a time in the application. The open project is displayed in the Project Window.

Creating new projects, opening existing projects

A new project is created with the menu command Project | New Project. An existing project is opened with the menu command Project | Open Project. The newly opened project (whether new or existing) replaces the previously opened project in the Project Window. If the previously opened project contains unsaved changes (indicated by an asterisk next to the folder name; see screenshot below), you are asked whether you wish to save these changes.

Naming and saving projects

A new project is named when you save it. A project is saved with the Project | Save Project command and has the .spp file extension. After a project has been modified, the project must be saved for the modifications to be stored. Note that a project (indicated by the top-level folder in the Project Window) can only be re-named by changing its name in Windows File Explorer; the name cannot be changed in the GUI. (The names of sub-folders, however, can be changed in the GUI.)

Project structure

A project has a tree structure of folders and files. Folders and files can be created at any level and to an unlimited depth. Do this by selecting a folder in the Project Window and then using the commands in the Project menu or context menu to add folders, files, or resources. Folders, files,
and resources that have been added to a project can be deleted or dragged to other locations in the project tree.

When a new project is created, the default project structure organizes the project by file type (XML, XSL, etc) (see screenshot below).

![Project Structure Screenshot]

File-type extensions are associated with a folder via the property definitions for that folder. When a file is added to a folder, it is automatically added to the appropriate child folder according to the file-type extension. For each folder, you can define what file-type extensions are to be associated with it.

**What can be added to a project**

Folder, files, and other resources can be added either to the top-level project folder or to a folder at any level in the project. There are three types of folders: (i) project folders; (ii) external folders; (iii) external web folders.

To add an object, select the relevant folder and then the required command from the Project menu or context menu of the selected folder. The following objects are available for addition to a project folder:

- **Project folders** (green) are folders that you add to the project in order to structure the project's contents. You can define what file extensions are to be associated with a project folder (in the properties of that folder). When files are added to a folder, they are automatically added to the first child folder that has that file's extension associated with it. Consequently, when multiple files are added to a folder, they will be distributed by file extension among the child folders that have the corresponding file-extension associations.

- **External folders** (yellow) are folders in a file system. When an external folder is added to a folder, the external folder and all its files, sub-folders, and sub-folder files are included in the project. Defining file extensions on an external folder serves to filter the files available in the project.

- **External web folders** are like external folders, except that they are located on a web server and require user authentication to access. Defining file extensions on an external web folder serves to filter the files available in the project.

- **Files** can be added to a folder by selecting the folder and then using one of the three Add-File commands: (i) **Add Files**, to select the file/s via an Open dialog; (ii) **Add Active File**, to add the file that is active in the Main Window; (iii) **Add Active and Related Files**, additionally adds files related to an active XML file, for example, an XML Schema or DTD. Note that files associated by means of a processing instruction (for example, XSLT files), are not considered to be related files.
Global Resources are aliases for file, folder, and database resources. How they are defined and used is described in the section on Global Resources.

URLs identify a resource object via a URL.

An Altova Scripting Project, which is a .asprj file, can be assigned to an XMLSpy project. This will make macros and other scripts available to the project. How to create a Scripting Project and assign one to an XMLSpy project is described in the section, Scripting.

Project properties

The properties of a folder are stored in the Properties dialog of that folder. It is accessed by first selecting the folder and then the Properties command in the Project menu or context menu (obtained by right-clicking the folder). Note that properties can be defined not only for the top-level project folder, but also for folders at various levels of the project hierarchy. The following properties of a folder can be defined and edited in the Properties dialog:

- **Folder name**: cannot be edited for the top-level project folder (for which, instead of a name, a filepath is displayed).
- **File extensions**: cannot be edited for the top-level project.
- **Validation**: specifies the DTD or XML Schema file that should be used to validate XML files in a folder.
- **Transformations**: specifies (i) the XSLT files to be used for transforming XML files in the folder, and (ii) the XML files to be transformed with XSLT files in the folder.
- **Destination files**: for the output of transformations, specifies the file extension and the folder where the files are to be saved.
- **SPS files for Authentic View**: specifies the SPS files to be used so that XML files in a folder can be viewed and edited in Authentic View.

See the description of the Project | Properties command for more detailed information.

Source control in projects

Source control systems that are compatible with Microsoft Visual Source-Safe are supported in projects. How to use this feature is described in the User Reference section of the manual.

Saving projects

Any changes you make to a project, such as adding or deleting a file, or modifying a project property, must be saved with the Save Project command.

Find in project

You can search for project files and folders using their names or a part of their name. If the search is successful, files or folders that are located are highlighted one by one.

To start a search, activate the Project window by clicking it (or in it), then select the command Edit | Find (or the shortcut Ctrl+F). In the Find dialog that pops up (screenshot below) enter the text string you wish to search for and select or deselect the search options (explained below) according to your requirements.
The following search options are available:

- Whole-word matching is more restricted since the entire string must match an entire word in the file or folder name. In file names, the parts before and after the dot (without the dot) are each treated as a word.
- It can be specified that casing in the search string must exactly match the text string in the file or folder name.
- Folder names can be included in the search. Otherwise, only file names are searched.
- External folders can be included or excluded from the search. External folders are actual folders on the system or network, as opposed to project folders, which are created within the project and not on the system.

If the search is successful, the first matching item is highlighted in the Project sidebar. You can then browse through all the returned matching items by clicking the Find Next and Find Prev buttons in the Find dialog.

**Refreshing projects**
If a change is made to an external folder, this change will not be reflected in the Project Window till the project is refreshed.
17.2 Using Projects

Projects are very useful for organizing your workspace, applying settings to multiple files, and for setting up and executing batch commands. Using projects can therefore greatly help speed up and ease your work.

Benefits of using projects
The following list lists the benefits of using projects.

- Files and folders can be grouped into folders by file extension or any other desired criterion.
- Schemas and XSLT files can be assigned to a folder. This can be useful if you wish to quickly validate or transform a single XML file using different schema or XSLT files. Add the XML file to different folders and define different schemas and XSLT files for the different folders.
- Batch processing can be applied to individual folders. The commands available for batch processing are listed below.
- Output folders can be specified for transformations.

Organizing resources for quick access
Folder and file resources can be organized into a tree structure, giving you a clear overview of the various folders and files in your project, and enabling you to quickly access any and all files in a project. Simply double-click a file in the Project window to open it. You can quickly add files and folders to a project as required and delete unwanted files and folders. When you wish to work with another project, close the project currently open in the Project Window and open the required project.

Batch processing
The commands for batch processing of files in a folder, whether the top-level project folder or a folder at any other level, are available in the context menu of that folder (obtained by right-clicking the folder). The steps for batch processing are as follows:

1. Define the files to be used for validation or transformation in the Properties dialog of that folder.
2. Specify the folder in which the output of transformations should be saved. If no output folder is specified for a folder, the output folder of the next ancestor folder in the project tree is used.
3. Use the commands in the context menu for batch execution. If you use the corresponding commands in the XML, DTD/Schema, or XSL/XQuery menus, the command will be executed only on the document active in the Main Window, not on any project folder in the Project Window.

The following commands in the context menu of a project folder (top-level or other) are available for batch processing:

- **Well-formed check**: If any error is detected during the batch execution, it is reported in the Messages Window.
- **Validation**: If any error is detected during the batch execution, it is reported in the Messages Window.
- **Transformations**: Transformation outputs are saved to the folder specified as the output
folder in the Properties dialog of that folder. If no folder is specified, the output folder of the next ancestor project folder is used. If no ancestor project folder has an output folder defined, a document window is opened and the results of each transformation is displayed successively in this document window. An XSL-FO transformation transforms an XML document or FO document to PDF.

- **Generate DTD / XML Schema:** Before the schemas are generated, you are prompted to specify an output folder. The generated schema files are saved to this folder and displayed in separate windows in the GUI.

**Note:** To execute batch commands use the context menu of the relevant folder in the Project Window. Do not use the commands in the XML, DTD/Schema, or XSL/XQuery menus. These commands will be executed on the document active in the Main Window.

**Validation and XSLT/XQuery with RaptorXML Server**
Context menu commands on project folder enable you to use RaptorXML Server for high-performance XML validation and XSLT/XQuery transformations. See the section [RaptorXML Server](#) for more information.
18  **RaptorXML Server**

If Altova RaptorXML Server is installed and licensed on your network and if your XMLSpy installation has access to it, then you can use RaptorXML Server (hereafter also called RaptorXML or Raptor for short) to validate XML and XBRL* documents, as well as run XSLT and XQuery transformations. You can validate the active document or all the documents in an XMLSpy project folder. The validation results are displayed in the Messages window of the GUI.

In XMLSpy, you can (i) validate documents or (ii) run XSLT/XQuery transformation by using either XMLSpy's engines or RaptorXML Server. One of the main advantages of using Raptor is that you can configure individual validations by means of a large range of validation options. Furthermore, you can store a set of Raptor options as a "configuration" in XMLSpy, and then select one of your defined configurations for a particular Raptor validation. Using Raptor is also advantageous when large data collections are to be validated.

**Note:** The actual performance depends on the number of PC processor cores used by RaptorXML Server for the validation: The higher the number of cores used, the faster will be the processing.

*Note:* There are two editions of Raptor: RaptorXML Server (for XML validations) and RaptorXML +XBRL Server (for XML and XBRL validations). If you wish to validate XBRL documents, you must use RaptorXML+XBRL Server. For more information about RaptorXML(+XBRL) Server, please see the Altova website and the user manuals: **RaptorXML Server** and **RaptorXML+XBRL Server**.

**Note:** RaptorXML Server cannot be used with HTTP proxies because these do not support websocket upgrades. If you encounter this problem, add the RaptorXML Server host to the proxy-ignore list.

**How to validate or transform using RaptorXML Server**

To validate an XML or XBRL document using RaptorXML Server, or to run an XSLT or XQuery transformation, XMLSpy must know which RaptorXML Server to use, how to access this server, and what options to pass to Raptor for the validation. This information is managed in XMLSpy as follows:

1. **By adding a server to the pool of Raptor servers.** In this step, RaptorXML Servers are added to a pool, and the access information of each server is stored in XMLSpy. Each server is identified by a name.
2. **By defining configurations for each server.** A configuration is a set of Raptor validation options. Each server can have multiple configurations. For a validation, you select one configuration, which becomes the active configuration.
3. **Selecting a server configuration with which to validate.** A server and one of its configurations is selected to be the active configuration. The active configuration is used for all subsequent validations.
4. **Validate or run the XSLT/XQuery transformation with Raptor.**
18.1 Adding Servers and Server Configurations

In the RaptorXML Server Options dialog (screenshot below, Tools | Manage Raptor Servers), you can add multiple Raptor Servers to the pool of available Raptor Servers and then define multiple configurations for each server. The added servers, together with the configurations you define for each of them, will appear in the Tools | Raptor Servers and Configurations submenu. In this submenu, you can select the server configuration you want to use for a Raptor validation.

Adding a Raptor Server

In the dialog's Servers pane (screenshot below), click the Add Server icon, then enter the name by which you wish to identify the Raptor server, the network name of the machine on which Raptor is installed (host name), and the port of the Raptor Server. Click OK to save the settings.
• **Name:** Any string you choose. It is used in XMLSpy to identify a particular RaptorXML Server.

• **Host name:** The name or IP address of the network machine on which the Raptor server is installed. Processing will be faster if you use an IP address rather than a host name. The IP address corresponding to `localhost` (the local machine) is `127.0.0.1`.

• **Port:** The port via which the Raptor server is accessed. This port is specified in Raptor's configuration file (called `server_config.xml`). The port must be fixed and known so that requests can be correctly addressed to the service. For more information about the Raptor configuration file, see the user manuals: *RaptorXML Server* and *RaptorXML+XBRL Server*.

After entering the server information, click **OK**. The server name you entered appears in the server list (in the left of the pane). A green icon appears next to the server's name, indicating that the Raptor server has been started and is running. The details of the server are displayed in the pane (see screenshot above). A red icon indicates that the server is offline. If the server cannot be found, an error message is displayed.

**Note:** The Raptor server must be running when the server is added. This is necessary so that XMLSpy can obtain information about the server and store it. If, after the server has been added, the server is offline or cannot be found, then these situations are indicated, respectively, by a red icon or an error message.

To edit a server's name, host name, or port, select the server in the left-hand pane, click the **Edit** button, and, in the dialog that appears, edit the information you want to change. To remove a server from the pool, select the server and click the **Remove Selected Server** icon.

**Server Configurations**

A configuration is a set of RaptorXML validation options. When a server is added, it will have a configuration named `default`. This is a set of RaptorXML options set to their default values. You can edit these values. You can also add new configurations that contain other option values. After you have defined multiple server configurations, you can select one configuration to be the active configuration. This is the configuration that will be used when the **Validate on Server** command is executed.

The **Configurations** pane has two parts: (i) a left-hand pane, which shows the configurations and the types of document that can be validated; (ii) a right-hand pane, which displays the options, organized in groups, of the validation type that is selected in the left-hand pane; at the bottom of the right-hand pane is a description of the selected option (see screenshot above).
Adding a configuration

In the Configurations pane of the RaptorXML Server Options dialog (screenshot above), click **Add a Configuration**. A new configuration is added with default option values. You can also create a new configuration by clicking **Copy Selected Configuration**. This creates a new configuration with option values that are the same as that of the copied configuration. New configurations are created with default names of the type `config<X>`; you can edit the name of a configuration by double-clicking it and entering the new name. You can then edit any of the configuration's option values.

Editing a configuration's option values

First select the validation document in the left-hand pane. This displays the options of that group in the right-hand pane. To edit the value of an option, do one of the following (depending on the type of option value):

- If the value can be one of a set of predefined values, select the value you want from the combo box of that option's value column.
- If the value is not constrained, click in the option's value filed and enter the value you want.
- If the value is a file path, in addition to being able to enter the value, you can also browse for the file you want by using the **Browse** button in the option's value column.

If you select an option, its description is displayed in a box at the bottom of the right-hand pane.
For more detailed descriptions of each option, see the command line interface chapters of the RaptorXML Server and RaptorXML(+XBRL) Server user manuals.

Removing a configuration
In the left-hand pane, select the configuration to be removed and click Remove Selected Configuration.

XMLSpy in Visual Studio and Eclipse
When XMLSpy is integrated in Visual Studio and Eclipse, the active configuration in these IDEs will be the one that is currently set as the active configuration in the standalone version of XMLSpy.
18.2 Validating with RaptorXML Server

You can validate XML and XBRL* documents with RaptorXML Server. Validating involves two steps:

- Selecting the server and server configuration to use for the validation
- Running the validation (by using one of the Validate on Server commands; see below)

*Note: There are two editions of Raptor: RaptorXML Server (for XML validations) and RaptorXML +XBRL Server (for XML and XBRL validations). If you wish to validate XBRL documents, you must use RaptorXML+XBRL Server. For more information about RaptorXML(+XBRL) Server, please see the Altova website and the user manuals: RaptorXML Server and RaptorXML+XBRL Server.

Selecting the server configuration to use

If you have defined multiple configurations on multiple servers, you can select a server and one of its configurations as the active configuration. The active configuration will be used for subsequent validations. On placing the cursor over the Tools | Raptor Servers and Configurations command (see screenshot below), a submenu appears that contains all the added servers, together with the configuration of each. Select the server configuration you want to make the active configuration. In the screenshot below, the xbrl configuration of the server named Raptor-01 has been selected as the active configuration (indicated by the green arrow).

Note: You can also select the active configuration in the dropdown menu of the Validate on Server icon. This menu also has a command to validate EDGAR on the active server.

Validating with RaptorXML Server

You can validate XML and XBRL documents by using the validation engines of XMLSpy or by using RaptorXML Server. To validate using RaptorXML Server, do one of the following:

- Click the toolbar icon Validate on Server
- Select the command XML | Validate XML on Server (high-performance) (Ctrl+F8)
- In the Project entry helper, right-click the project, a folder, or a file, and select Validate XML on Server (high performance) to validate XML or XBRL data in the selected object.

Note: Raptor validation is available in Text View, Schema View, and Grid View.
18.3 Validation Options

This section is organized by the type of document being validated (see the left-hand pane of the screenshot below). For example, XML with W3C Schema validates an XML document against a W3C XML Schema. When a validation type is selected in the left-hand pane, the RaptorXML Server validation options available for that kind of validation are displayed in the right-hand pane. These options are organized into groups, such as Scripting and XML Schema (see screenshot below).

The sub-sections of this section contain links to the descriptions of the respective RaptorXML Server validation options.

18.3.1 Common Options

Options that are common to all types of validation.

- Common
  - Verbose
  - Network Timeout
18.3.2 XML with DTD
Options for validating XML data against a DTD.

- Common
  - Verbose
  - Network Timeout

- Catalog
  - XML User Catalog

- XML
  - Enable Namespaces
  - External DTD
  - Streaming Mode

18.3.3 DTD
Options for validating DTDs.

- Common
  - Verbose
  - Network Timeout

- Catalog
  - XML User Catalog

- Scripting
  - Script
  - Script Parameters
  - Script API Version
18.3.4 XML with W3C Schema

Options for validating XML data against XML Schema.

- Common
  - Verbose
  - Network Timeout

- Catalog
  - XML User Catalog

- Scripting
  - Script
  - Script Parameters
  - Script API Version

- XML
  - Streaming Mode

- XML Schema
  - External Schemas (xsd)
  - Import Strategy (schema-imports)
  - xsi:schemaLocation Strategy (schemalocation-hints)
  - Mapping Strategy (schema-mapping)
  - XML Processing Mode (xml-mode)
  - Enable XInclude (xinclude)
  - Assessment Mode
  - Parallel Assessment

18.3.5 W3C Schema

Options for validating XML Schemas.

- Common
  - Verbose
  - Network Timeout

- Catalog
  - XML User Catalog
Scripting

Script
Script Parameters
Script API Version

XML Schema

Import Strategy (schema-imports)
xsi:schemaLocation Strategy (schemalocation-hints)
Mapping Strategy (schema-mapping)
XML Processing Mode (xml-mode)
Enable XInclude (xinclude)

18.3.6 XBRL Instance
Options for validating XBRL instance documents.

Common

Verbose
Network Timeout

Catalog

XML User Catalog

Scripting

Script
Script Parameters
Script API Version

XML Schema

Import Strategy (schema-imports)
xsi:schemaLocation Strategy (schemalocation-hints)
Mapping Strategy (schema-mapping)
Enable XInclude (xinclude)
Parallel Assessment

XBRL

Enable Dimensions Extension (dimensions)
Enable Extensible Enumerations Extension (extensible-enumerations)
Enable Unit Registry Extensions (utr)
Preload XBRL Spec Schemas (preload-xbrl-schemas)
Taxonomy Packages
Validate Referenced DTS Only (validate-dts-only)
Treat XBRL Inconsistencies as Errors (treat-inconsistencies-as-errors)

- XBRL Formula
  - Enable Formula Extension (formula)
  - Enable Assertion Severity Extension (assertion-severity)
  - Preload Formula Spec Schemas (preload-formula-schemas)
  - Report Unsatisfied Assertion Evaluations
  - Validation Message Language (message-lang)
  - Validation Message Role (message-role)

- XBRL Table
  - Enable Table Extension (table)
  - Preload Table Spec Schemas (preload-table-schemas)
  - Table Linkbase Namespace

18.3.7 XBRL Taxonomy
Options for validating XBRL taxonomies.

- Common
  - Verbose
  - Network Timeout

- Catalog
  - XML User Catalog

- Scripting
  - Script
  - Script Parameters
  - Script API Version

- XML Schema
  - Import Strategy (schema-imports)
  - xsi:schemaLocation Strategy (schemalocation-hints)
  - Mapping Strategy (schema-mapping)
  - Enable XInclude (xinclude)

- XBRL
  - Enable Dimensions Extension (dimensions)
Enable Extensible Enumerations Extension (extensible-enumerations)
Preload XBRL Spec Schemas (preload-xbrl-schemas)
Taxonomy Packages
Treat XBRL Inconsistencies as Errors (treat-inconsistencies-as-errors)

XBRL Formula
Enable Formula Extension (formula)
Enable Assertion Severity Extension (assertion-severity)
Preload Formula Spec Schemas (preload-formula-schemas)

XBRL Table
Enable Table Extension (table)
Preload Table Spec Schemas (preload-table-schemas)
Table Linkbase Namespace

18.3.8 XSLT
Options for validating XSLT documents.

Common
Verbose
Network Timeout

Catalog
XML User Catalog

XML Schema
Import Strategy (schema-imports)
xsi:schemaLocation Strategy (schemalocation-hints)
Mapping Strategy (schema-mapping)
XML Processing Mode (xml-mode)
Enable XInclude (xinclude)

Java Extension
Disable Java Extensions (javaext-disable)
Barcode Extension Location (javaext-barcode-location)

Chart Extensions
Disable Chart Extensions (chartext-disable)
## .NET Extensions

- Disable .NET Extensions (dotnetext-disable)

## XEngines Common

- Load XML with PSVI (load-xml-with-psvi)

## XSLT

- XSLT Engine Version (xslt-version)
- Template Mode
- Template Entry Point

### 18.3.9 XQuery

Options for validating XQuery documents.

- Common
  - Verbose
  - Network Timeout

- Catalog
  - XML User Catalog

- XML Schema
  - Import Strategy (schema-imports)
  - xsi:schemaLocation Strategy (schemalocation-hints)
  - Mapping Strategy (schema-mapping)
  - XML Processing Mode (xml-mode)
  - Enable XInclude (xinclude)

- Java Extension
  - Disable Java Extensions (javaext-disable)
  - Barcode Extension Location (javaext-barcode-location)

- Chart Extensions
  - Disable Chart Extensions (chartext-disable)
18.3.10 JSON
Options for validating JSON (instance) documents.

- **Common**
  - **Verbose**
  - **Network Timeout**

- **Catalog**
  - **XML User Catalog**

18.3.11 JSON Schema
Options for validating JSON Schema documents.

- **Common**
  - **Verbose**
  - **Network Timeout**

- **Catalog**
  - **XML User Catalog**
18.3.12 EDGAR

EDGAR (Electronic Data Gathering, Analysis, and Retrieval) is a system that performs automated collection, validation, and indexing of financial statements filed by companies to the United States SEC (Securities and Exchange Commission). When you validate via EDGAR, Raptor validates the XBRL instance document using an internal EDGAR script. You can set the following additional options.

EDGAR Script Parameters

The EDGAR script performs extra checks as prescribed in the EDGAR Filing Manual Volume II: EDGAR Filing. The script allows the following script parameters to be additionally specified:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIK</td>
<td>The CIK of the registrant</td>
</tr>
<tr>
<td>submissionType</td>
<td>The EDGAR submission type, for example: '10-K'</td>
</tr>
<tr>
<td>cikList</td>
<td>A list of CIKs, each separated by a comma: ','</td>
</tr>
<tr>
<td>cikNameList</td>
<td>A list of official registrant names for each CIK in cikList, separated by '</td>
</tr>
<tr>
<td>forceUtrValidation</td>
<td>Set to true to force-enable UTR validation</td>
</tr>
<tr>
<td>edbody-url</td>
<td>The path to the edbody.dtd that is used to validate embedded HTML fragments</td>
</tr>
<tr>
<td>edgar-taxonomies-url</td>
<td>The path to the edgartaxonomies.xml, which contains a list of taxonomy files that are allowed to be referenced from the company extension taxonomy</td>
</tr>
</tbody>
</table>

Common

Verbose
Network Timeout

Catalog

XML User Catalog

XML Schema

Import Strategy (schema-imports)
xs:schemaLocation Strategy (schemalocation-hints)
Mapping Strategy (schema-mapping)
Enable XInclude (xinclude)
Parallel Assessment

XBRL

Enable Dimensions Extension (dimensions)
Enable Extensible Enumerations Extension (extensible-enumerations)
Preload XBRL Spec Schemas (preload-xbrl-schemas)
Taxonomy Packages
Treat XBRL Inconsistencies as Errors (treat-inconsistencies-as-errors)

- XBRL Formula
  Enable Formula Extension (formula)
  Enable Assertion Severity Extension (assertion-severity)
  Preload Formula Spec Schemas (preload-formula-schemas)
  Report Unsatisfied Assertion Evaluations
  Validation Message Language (message-lang)
  Validation Message Role (message-role)

- XBRL Table
  Enable Table Extension (table)
  Preload Table Spec Schemas (preload-table-schemas)
  Table Linkbase Namespace
XSLT and XQuery with RaptorXML Server

You can use RaptorXML Server to run (i) XSLT transformations, (ii) and XQuery updates or executions on XML documents. These actions are available only via Projects, and involve three steps:

- Selecting the server and server configuration to use for the job.
- Setting up the project folder, and specifying the XSLT/XQuery files to use (in the Project Properties dialog). The XSLT/XQuery files that are assigned in the Project Properties dialog of a folder are the files that will be used for XSLT and XQuery transformations of all XML files in that project folder. You cannot assign XSLT/XQuery files for individual XML files in a project folder; XSLT/XQuery files can only be assigned for an entire folder.
- Running the XSLT transformation or XQuery update/execution.

Note: If the XSLT or XQuery document uses Java extension functions or .NET extension functions, then file paths are used to locate JAR files (Java) or external (unregistered) assembly files (.NET). This means that, if the same XSLT/XQuery document is used for transformations/executions via XMLSpy as well as RaptorXML Server, then file paths in it to JAR files and/or assembly files must correctly locate these files.

Note: If RaptorXML Server is on the same machine as XMLSpy, you should, for best performance, specify that the server setting server.unrestricted-filesystem-access has a value of true. For more information, see the documentation of the RaptorXML Server configuration file.

Selecting the server configuration to use

If you have defined multiple configurations on multiple servers, you can select a server and one of its configurations as the active configuration. The active configuration will be used for subsequent validations. On placing the cursor over the Tools | Raptor Servers and Configurations command (see screenshot below), a submenu appears that contains all the added servers, together with the configuration of each. Select the server configuration you want to make the active configuration. In the screenshot below, the xbrl configuration of the server named Raptor-01 has been selected as the active configuration (indicated by the green arrow).

Note: You can also select the active configuration in the dropdown menu of the Validate on Server icon. This menu also has a command to validate EDGAR on the active server.

Running an XSLT transformation

You can carry out an XSLT transformation by using the XSLT engines of XMLSpy or by using
RaptorXML Server. To run XSLT transformations using RaptorXML Server, do the following:

- Right-click the project folder where the XML files to transform are located. This folder can be the entire project folder or an individual folder anywhere in the project hierarchy.
- In the menu that appears, select the command **XSL Transformation on Server (high-performance)**

**Note:** You cannot assign XSLT/XQuery files for individual XML files in a project folder; XSLT/XQuery files can only be assigned for an entire folder. See [start of section](#).

For more related information, see the sections [XSLT](#) and [XSLT Transformation](#).

**Running an XQuery update/execution**

You can carry out an XQuery update/transformation by using the XQuery engines of XMLSpy or by using RaptorXML Server. To run XQuery updates/transformation using RaptorXML Server, do the following:

- Right-click the project folder where the XQuery or XML files to, respectively, update or execute are located. This folder can be the entire project folder or an individual folder anywhere in the project hierarchy.
- In the menu that appears, select the command **XQuery/Update Execution on Server (high-performance)**

**Note:** You cannot assign XSLT/XQuery files for individual XML files in a project folder; XSLT/XQuery files can only be assigned for an entire folder. See [start of section](#).

For more related information, see the sections [XQuery](#) and [XQuery/Update Execution](#).
19 File/Directory Comparisons

XMLSpy provides a File Comparison feature and a Directory Comparison feature that are linked to each other. File Comparisons and Directory Comparisons are started with the Compare Open File With and Compare Directories commands in the Tools menu, respectively. Comparison options for file comparisons can be defined in the Settings dialog, which is accessed by clicking the Compare Options command in the Tools menu.

Each of these commands is described in detail in the User Reference section. In the sub-sections of this section we provide an overview of the File Comparisons and Directory Comparisons mechanisms.
19.1 File Comparisons

The File Comparisons feature enables you to compare the active file with another file, which is selected via an Open File dialog or via a global resource. The following points provide an overview of the mechanism. For details, see the User Reference section.

- The settings current in the Compare Options dialog when a File Compare session is started are the settings that will be active for that session.
- You can choose to compare the files as XML files (where document structure is also evaluated) or as Text files. This choice is made by selecting, in the Settings dialog, either (i) Grid View or Text View (Textual Comparison Only unchecked) for XML comparisons, or (ii) Text View (Textual Comparison Only checked) for text comparisons.
- The two files appear in adjacent panes in the selected view (Grid View or Text View) and the differences are highlighted in both files (screenshot below).

A Compare Files control window also pops up which enables you to navigate through the differences and to merge them.

The Settings dialog offers several options for specifying what aspects of the XML documents should be considered for the comparison, and what aspects ignored. For more details, see the Compare Options section in the User Reference.
19.2 Directory Comparisons

The Directory Comparisons feature enables you to compare directories, each of which you select via separate Browse for Folder dialogs. You can also select whether sub-directories are to be compared or not, and what file types should be considered for the directory comparison.

Directories are compared to indicate missing files and whether files of the same name are different or not. The comparisons between files are based on the settings in the Settings dialog. The results of the directory comparison are displayed in a separate window (screenshot below).

For details about how to read the symbols and manage the view in the Compare Directories window, see the description of the Compare Directories command in the User Reference. You can then double-click a file row to directly start a file comparison.
20 Source Control

The source control support in XMLSpy is available through the Microsoft Source Control Plug-in API (formerly known as the MSSCCI API), versions 1.1, 1.2 and 1.3. This enables you to run source control commands such as "Check in" or "Check out" directly from XMLSpy to virtually any source control system that lets native or third-party clients connect to it through the Microsoft Source Control Plug-in API.

You can use as your source control provider any commercial or non-commercial plug-in that supports the Microsoft Source Control Plug-in API, and can connect to a compatible version control system. For the list of source control systems and plug-ins tested by Altova, see Supported Source Control Systems.

Installing and configuring the source control provider

To view the source control providers available on your system, do the following:

1. On the Tools menu, click Options.
2. Click the Source Control tab.

Any source control plug-ins compatible with the Microsoft Source Code Control Plug-in API are displayed in the Current source control plug-in drop-down list.

![Source Control Plug-in Configuration](image)

If a compatible plug-in cannot be found on your system, the following message is displayed:

"Registration of installed source control providers could not be found or is incomplete."

Some source control systems might not install the source control plug-in automatically, in which case you will need to install it separately. For further instructions, refer to the documentation of the respective source control system. A plug-in (provider) compatible with the Microsoft Source
Code Control Plug-in API is expected to be registered under the following registry entry on your operating system:

HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider\InstalledSCCProviders

Upon correct installation, the plug-in becomes available automatically in the list of plug-ins available to XMLSpy.

**Accessing the source control commands**
The commands related to source control are available in the **Project | Source Control** menu.

**Resource / Speed issues**
Very large source control databases might be introducing a speed/resource penalty when automatically performing background status updates.

You might be able to speed up your system by disabling (or increasing the interval of) the **Perform background status updates every ... seconds** option in the **Source Control** tab accessed through **Tools | Options**.

**Note:** The 64-bit version of your Altova application automatically supports any of the supported 32-bit source control programs listed in this documentation. When using a 64-bit Altova application with a 32-bit source control program, the **Perform background status updates every ... seconds** option is automatically grayed-out and cannot be selected.

**Differencing with Altova DiffDog**
You can configure many source control systems (including Git and TortoiseSVN) so that they use Altova DiffDog as their differencing tool. For more information about DiffDog, see [https://www.altova.com/diffdog](https://www.altova.com/diffdog). For DiffDog documentation, see [https://www.altova.com/documentation.html](https://www.altova.com/documentation.html).
20.1 Setting Up Source Control

The mechanism for setting up source control and placing files in a XMLSpy project under source control is as follows:

1. If this hasn't been done already, install the source control system (see Supported Source Control Systems) and set up the source control database (repository) to which you wish to save your work.
2. Create a local workspace folder that will contain the working files that you wish to place under source control. The folder that contains all your workspace folders and files is called the local folder, and the path to the local folder is referred to as the local path. This local folder will be bound to a particular folder in the repository.
3. In your Altova application, create an application project folder to which you must add the files you wish to place under source control. This organization of files in an application project is abstract. The files in a project reference physical files saved locally, preferably in one folder (with sub-folders if required) for each project.
4. In the source control system's database (also referred to as source control or repository), a folder is created that is bound to the local folder. This folder (called the bound folder) will replicate the structure of the local folder so that all files to be placed under source control are correctly located hierarchically within the bound folder. The bound folder is usually created when you add a file or an application project to source control for the first time. See the section, Application Project, for information about the repository's folder structure.
5. Project files are added to source control using the command Project | Source Control | Add to Source Control. When you add a project or a file in a project for the first time to source control, the correct bindings and folder structure will be created in the repository.
6. Source control actions, such as the checking in and out of files, and the removing of files from source control, can be carried out via commands in the Project | Source Control submenu. These commands are described in the Project menu subsection of the User Reference.

Note: If you wish to change the current source control provider, this can be done in one of two ways: (i) via the Source Control options (Tools | Options | Source Control), or (ii) in the Change Source Control dialog (Project | Source Control | Change Source Control).
20.2 **Supported Source Control Systems**

The list below shows the Source Control Servers (SCSs) supported by XMLSpy, together with their respective Source Control Clients (SCCs). The list is organized alphabetically by SCS. Note the following:

- Altova has implemented the Microsoft Source Control Plug-in API (versions 1.1, 1.2, and 1.3) in XMLSpy, and has tested support for the listed drivers and revision control systems. It is expected that XMLSpy will continue to support these products if, and when, they are updated.
- Source Code Control clients not listed below, but which implement the Microsoft Source Control Plug-in API, should also work with XMLSpy.

<table>
<thead>
<tr>
<th>Source Control System</th>
<th>Source Code Control Clients</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccuRev 4.7.0 Windows</td>
<td>AccuBridge for Microsoft SCC 2008.2</td>
</tr>
<tr>
<td>Bazaar 1.9 Windows</td>
<td>Aigenta Unified SCC 1.0.6</td>
</tr>
<tr>
<td>Borland StarTeam 2008</td>
<td>Borland StarTeam Cross-Platform Client 2008 R2</td>
</tr>
<tr>
<td>Codice Software Plastic SCM Professional 2.7.127.10 (Server)</td>
<td>Codice Software Plastic SCM Professional 2.7.127.10 (SCC Plugin)</td>
</tr>
<tr>
<td>Collabnet Subversion 1.5.4</td>
<td>• Aigenta Unified SCC 1.0.6</td>
</tr>
<tr>
<td></td>
<td>• PushOK SVN SCC 1.5.1.1</td>
</tr>
<tr>
<td></td>
<td>• PushOK SVN SCC x64 version 1.6.3.1</td>
</tr>
<tr>
<td></td>
<td>• TamTam SVN SCC 1.2.24</td>
</tr>
<tr>
<td>ComponentSoftware CS-RCS (PRO) 5.1</td>
<td>ComponentSoftware CS-RCS (PRO) 5.1</td>
</tr>
<tr>
<td>Dynamsoft SourceAnywhere for VSS 5.3.2 Standard/Professional Server</td>
<td>Dynamsoft SourceAnywhere for VSS 5.3.2 Client</td>
</tr>
<tr>
<td>Dynamsoft SourceAnywhere Hosted</td>
<td>Dynamsoft SourceAnywhere Hosted Client (22252)</td>
</tr>
<tr>
<td>Dynamsoft SourceAnywhere Standalone 2.2 Server</td>
<td>Dynamsoft SourceAnywhere Standalone 2.2 Client</td>
</tr>
<tr>
<td>Git</td>
<td>PushOK GIT SCC plug-in (see <a href="#">Source Control with Git</a>)</td>
</tr>
<tr>
<td>IBM Rational ClearCase 7.0.1 (LT)</td>
<td>IBM Rational ClearCase 7.0.1 (LT)</td>
</tr>
<tr>
<td>March-Hare CVSNT 2.5 (2.5.03.2382)</td>
<td>Aigenta Unified SCC 1.0.6</td>
</tr>
<tr>
<td>March-Hare CVS Suite 2008</td>
<td>• Jalindi Igloo 1.0.3</td>
</tr>
<tr>
<td></td>
<td>• March-Hare CVS Suite Client 2008 (3321)</td>
</tr>
<tr>
<td></td>
<td>• PushOK CVS SCC NT 2.1.2.5</td>
</tr>
<tr>
<td></td>
<td>• PushOK CVS SCC x64 version 2.2.0.4</td>
</tr>
<tr>
<td></td>
<td>• TamTam CVS SCC 1.2.4</td>
</tr>
<tr>
<td>Mercurial 1.0.2 for Windows</td>
<td>Sergey Antonov HgSCC 1.0.1</td>
</tr>
<tr>
<td>Microsoft SourceSafe 2005 with CTP</td>
<td>Microsoft SourceSafe 2005 with CTP</td>
</tr>
<tr>
<td>Source Control System</td>
<td>Source Code Control Clients</td>
</tr>
<tr>
<td>------------------------------------------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>PureCM Server 2008/3a</td>
<td>PureCM Client 2008/3a</td>
</tr>
<tr>
<td>QSC Team Coherence Server 7.2.1.35</td>
<td>QSC Team Coherence Client 7.2.1.35</td>
</tr>
<tr>
<td>Reliable Software Code Co-Op 5.1a</td>
<td>Reliable Software Code Co-Op 5.1a</td>
</tr>
<tr>
<td>Seapine Surround SCM Client/Server for Windows 2009.0.0</td>
<td>Seapine Surround SCM Client 2009.0.0</td>
</tr>
<tr>
<td>Serena Dimensions Express/CM 10.1.3 for Win32 Server</td>
<td>Serena Dimensions 10.1.3 for Win32 Client</td>
</tr>
<tr>
<td>Softimage Alienbrain Server 8.1.0.7300</td>
<td>Softimage Alienbrain Essentials/Advanced Client 8.1.0.7300</td>
</tr>
<tr>
<td>SourceGear Fortress 1.1.4 Server</td>
<td>SourceGear Fortress 1.1.4 Client</td>
</tr>
<tr>
<td>SourceGear SourceOffsite Server 4.2.0</td>
<td>SourceGear SourceOffsite Client 4.2.0 (Windows)</td>
</tr>
<tr>
<td>SourceGear Vault 4.1.4 Server</td>
<td>SourceGear Vault 4.1.4 Client</td>
</tr>
<tr>
<td>VisualSVN Server 1.6</td>
<td>• Aigenta Unified SCC 1.0.6</td>
</tr>
<tr>
<td></td>
<td>• PushOK SVN SCC 1.5.1.1</td>
</tr>
<tr>
<td></td>
<td>• PushOK SVN SCC x64 version 1.6.3.1</td>
</tr>
<tr>
<td></td>
<td>• TamTam SVN SCC 1.2.24</td>
</tr>
</tbody>
</table>
20.3 **Local Workspace Folder**

The files you will be working with should be saved in a hierarchy inside a local workspace folder *(see diagram below)*.

![Local Workspace Folder Diagram]

The application project file *(.spp file)* typically will be located directly inside the local workspace folder *(see diagram above)*.

When one or more files in this (workspace) folder are placed under source control, the local workspace folder's structure is partly or wholly reproduced in the repository. For example, if the file `Persons.xml` from the local folder shown above is placed under source control, then the path to it in the repository will be:

```
[RepositoryFolder]/MyProject/Grouping/Persons/Persons.xml
```

The `MyProject` folder in the repository folder is bound to the local folder. Typically it would be the name of the project, but you could give it any name.

If the entire application project is placed under source control (by selecting the project name in the Projects window and placing it under source control), then the entire local folder structure is recreated in the repository.

**Note:** Files from outside the local workspace folder can be added to the application project. But whether you can place such a file under source control depends upon the source control system you are using. Some source control systems could have a problem placing a file from outside the local folder into the repository. We therefore recommend that all project files you wish to place under source control be located in the local workspace folder.
20.4 Application Project

Create or load the Altova application project you wish to place under source control. If you wish to place a single file under source control, this file must be included in a project—since source control can only be accessed via a project.

For example, consider a project in Altova's XMLSpy application. The project's properties are saved in a .spp file. In the application, the project is displayed in the application's Project window (see screenshot below). The project in the screenshot below is named MyProject and the project's properties are saved in the file MyProject.spp.

You can place the entire project (all files in the project) or only some project files under source control. Only files that are in the project can be placed under source control. So you will need to add files to the project before you can place them under source control. The project file (.spp file) will automatically be placed under source control as soon as a file from within the project is placed under source control.

The entire project, or one or more project files, is placed under source control via the command Project | Source Control | Add to Source Control (see next section below).

Note, however, that the folder structure of the repository corresponds not to the project's folder structure (screenshot above) but to the structure of the local workspace folder (see folder diagram below). In the diagram below, notice that the MyProject folder in the repository has a folder structure corresponding to that of the local workspace folder. Note that the bound folder occurs within the repository folder.

<table>
<thead>
<tr>
<th>Local Workspace Folder</th>
<th>Repository</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MyProject.spp</td>
</tr>
<tr>
<td></td>
<td>QuickStart</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Grouping</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An application project can contain project folders (green) and external folders (yellow). Only files in (green) project folders can be placed under source control. Files in (yellow) external folders cannot be placed under source control.

Files from outside the local workspace folder can be added to the application project. But whether you can place such a file under source control depends upon the source control system you are using. Some source control systems could have a problem placing a file from outside the local folder into the repository. We therefore recommend that all project files you wish to place under source control be located in the local workspace folder.
20.5 Add to Source Control

Adding the project to source control will automatically create the correct bindings and repository structure before adding the project file (.spp file) or individual files to source control. Add the project to source control as follows.

Select the project in the Project window (MyProject in the screenshot below) so that it is highlighted (as in the screenshot below). Alternatively select a single file, or select multiple files by clicking them with the Ctrl key pressed. Adding a single file to source control will automatically add the project file (.spp file) to source control as well.

Next, select the menu command Project | Source Control | Add to Source Control. This pops up the connection and configuration dialogs of the currently selected source control system. (You can change the source control system via the Change Source Control dialog (Project | Source Control | Change Source Control).)

Follow the source control system's instructions to make the connection and configuration. After this has been completed, all the files selected for addition plus the project file (.spp file) are displayed in an Add to Source Control dialog (screenshot below). Select the files you wish to add and click OK.
The files will be added to the repository and be either checked in or checked out depending on whether the *Keep Checked Out* check box has been checked or not.

**Configuration notes**

You might be prompted to create a folder in the repository for the project if it has not already been created. If you are, go ahead and create it. The *local workspace folder* will be bound to this folder created in the repository (see diagrams below).

```
Local Workspace Folder                             Repository
|-- MyProject.spp                                    |-- MyProject (bound to Local Workspace)
 ||-- QuickStart                                    |-- MyProject.spp
 |     |-- QuickStart.css                            |-- QuickStart
 |     |-- QuickStart.xml                            |-- QuickStart.xml
 |     |-- QuickStart.xsd                            |-- QuickStart.xsd
|-- Grouping                                        |-- Grouping
 |     |-- Persons                                    |-- Persons
 |     |      |-- Persons.xml                           |-- Persons.xml
```

The configuration dialog of Jalindi Igloo is show below. The CVSROOT field is the path to the repository folder.

In the screenshot above, the local path locates the local workspace folder, which corresponds to the CVS module, *MyProject*, and is bound to it.
20.6 **Working with Source Control**

To work with source control, select the project, a project folder, or a project file in the Project window *(screenshot below)* and then select the command you want in the **Project | Source Control** menu. The **Check In** and **Check Out** commands are available as context menu commands of Project window items.

![Project window](image)

In this section, we describe the main source control features in detail:

- **Add to, Remove from Source Control**
- **Check Out, Check In**
- **Getting Files as Read-Only**
- **Copying and Sharing from Source Control**
- **Changing Source Control**

Additional commands in the **Project | Source Control** menu are described in the **User Reference section** of the manual. For information specific to a particular source control system, please see the user documentation of that system.

### 20.6.1 Add to, Remove from Source Control

#### Adding

After a project has been added to source control, you can place files either singly or in groups under source control. This is also known as adding the files to source control. Select the file in the Project window and then click the command **Project | Source Control | Add to Source Control**. To select multiple files, keep the **Ctrl** key pressed while clicking on the files you wish to add. Running the command on a (green) project folder *(see screenshot below)* adds all files in the folder and its sub-folders to source control.
When files are added to source control, the local folder hierarchy is replicated in the repository (it is not the project folder hierarchy that is replicated). So, if a file is in a sub-folder X levels deep in the local folder, then the file's parent folder and all other ancestor folders are automatically created in the repository.

When the first file from a project is added to source control, the correct bindings are created in the repository and the project file (.spp file) is added automatically. For more details, see the section Add to Source Control.

Source control symbols
Files and the project folder display certain symbols, the meanings of which are given below.

- ![Check In](image)
  - Checked in. Available for check-out.
- ![Check Out](image)
  - Checked out by another user. Not available for check-out.
- ![Local Check Out](image)
  - Checked out locally. Can be edited and checked-in.

Removing
To remove a file from source control, select the file and click the command Project | Source Control | Remove from Source Control. You can also remove: (i) files in a project folder by executing the command on the folder, and (ii) the entire project by executing the command on the project.

20.6.2 Check Out, Check In

After a project file has been placed under source control, it can be checked out or checked in by selecting the file (in the Project window) and clicking the respective command in the Project | Source Control menu: Check Out and Check In.

When a file is checked out, a copy from the repository is placed in the local folder. A file that is checked out can be edited. If a file that is under source control is not checked out, it cannot be edited. After a file has been edited, the changes can be saved to the repository by checking in the file. Even if the file is not saved in the application, checking it in will save the changes to the repository. Whether a file is checked out or not is indicated with a tick or lock symbol in its
Project window icon.

Files and the project folder display certain symbols, the meanings of which are given below.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Symbol]</td>
<td>Checked out by another user. Not available for check-out.</td>
</tr>
<tr>
<td>![Symbol]</td>
<td>Checked out locally. Can be edited and checked-in.</td>
</tr>
</tbody>
</table>

Selecting the project or a folder within the project selects all files in the selected object. To select multiple objects (files and folders), press the Ctrl key while clicking the objects. The screenshot below shows a project that has been checked out. The file `QuickStart.css` has subsequently been checked in.

![Project Window](image)

**Saving and rejecting editing changes**

Note that, when checking in a file, you can choose to leave the file checked out. What this does is save editing changes to the repository while continuing to keep the file checked out, which is useful if you wish to periodically save editing changes to the repository and then continue editing.

If you have checked out a file and made editing changes, and then wish to reject these changes, you can revert to the document version saved in the repository by selecting the command **Project | Source Control | Undo Check Out**.

**Checking out**

The Check Out dialog (screenshot below) allows you: (i) to select the files to check out, and (ii) to select whether the repository version or the local version should be checked out.
Checking in

The Check In dialog (screenshot below) allows you: (i) to select the files to check in, and (ii) if you wish, to keep the file checked out.

Note: In both dialogs (Check Out and Check In), multiple files appear if the selected object (project or project folder/s) contain multiple files.

20.6.3 Getting Files as Read-Only

The Get command (in the Project | Source Control menu) retrieves files from the repository as read-only files. (To be able to edit a file, you must check it out.) The Get dialog lists the files in the object (project or folder) on which the Get command was executed (see screenshot below). You can select the files to retrieve by checking them in the Get dialog list.

Note: The Get Folders command allows you to select individual sub-folders in the repository if this is allowed by your source control system.
You can choose to overwrite changed checked-out files by checking this option at the bottom of the Get dialog. On clicking **OK**, the files will be overwritten. If any of the overwritten files is currently open, a dialog pops up (screenshot below) asking whether you wish to reload the file/s (**Reload** button), close the file/s (**Close**), or retain the current view of the file (**Cancel**).

**Advanced Get Options**

The Advanced Get Options dialog (screenshot below) is accessed via the **Advanced** button in the Get dialog (see first screenshot in this section).

Here you can set options for (i) replacing writable files that are checked out, (ii) the timestamp, and (iii) whether the read-only property of the retrieved file should be changed so that it will be writable.
Get latest version

The Get Latest Version command (in the Project | Source Control menu) retrieves and places the latest source control version of the selected file(s) in the working directory. The files are retrieved as read-only and are not checked out. This command works like the Get command (see above), but does not display the Get dialog.

If the selected files are currently checked out, then the action taken will depend on how your source control system handles such a situation. Typically, the source control system will ask whether you wish to replace, merge with, or leave the checked-out file as it is.

Note: This command is recursive when performed on a folder, that is, it affects all files below the current one in the folder hierarchy.

20.6.4 Copying and Sharing from Source Control

The Open from Source Control command creates a new application project from a project under source control.

Create the new project as follows:

1. Depending on the source control system used, it might be necessary, before you create a new project from source control, to make sure that no file from the source-controlled project is checked out.
2. No project need be open in the application, but can be.
3. Select the command Project | Source Control | Open from Source Control.
4. The source control system that is currently set will pop up its verification and connection dialogs. Make the connection to the bound folder in the repository that you want to copy.
5. In the dialog that pops up (screenshot below), browse for the local folder to which the contents of the bound folder in the repository (that you have just connected to) must be copied. In the screenshot below the bound folder is called MyProject and is represented by the $ sign; the local folder is C:\M20130326.
6. Click OK. The contents of the bound folder (MyProject) will be copied to the local folder \C:\M20130326\, and a dialog pops up asking you to select the project file (.spp file) that is to be created as the new project.

7. Select the .spp file that will have been copied to the local folder. In our example, this will be MyProject.spp located in the \C:\M20130326 folder. A new project named MyProject will be created in the application and will be displayed in the Project window. The project's files will be in the folder \C:\M20130326.

Sharing from source control

The Share from Source Control command is supported when the source control system being used supports shares. You can share a file, so that it is available at multiple local locations. A change made to one of these local files will be reflected in all the other "shared" versions.

In the application's Project window first select the project (highlighted in the screenshot below). Then click the Share from Source Control.
The Share To [Folder] dialog (screenshot below) pops up.

To select the files to share, first choose, in the project tree in the right-hand pane of the dialog (see screenshot above), the folder in which the files are. The files in the chosen folder are displayed in the left-hand pane. Select the file you wish to share (multiple files by pressing the Ctrl key and clicking the files you want to share). The selected file/s will be displayed in the Files to Share text box (at top left). The files disappear from the left hand pane. Click Share and then Close to copy the selected file/s to the local share folder. When you click Close, the files to share will be copied to the selected local location.

The share folder is noted in the name of the Share to [Folder] dialog. In the screenshot above it is the local folder (since the $ sign is the folder in the repository to which the local folder is bound). You can see and set the share folder in the Change Source Control dialog (screenshot below, Change Source Control) by changing the local path and server binding.
For more details about sharing using your source control system, see the source control system's user documentation.

### 20.6.5 Changing Source Control

Source control settings can be changed via two commands in the Project | Source Control menu:

- **Source Control Manager**, which opens the source control system application and allows you to set up databases and configure bindings.
- **Change Source Control**, which pops up the Change Source Control dialog, in which you can change the source control system being used by the Altova application and the current binding. This dialog is described below.

The current binding is what the active application project will use to connect to the source control database. The current binding is correct when the application project file (`.spp` file) is in the local folder and the bound folder in the repository is where this project's files are stored. Typically the bound folder and its sub-structure will correspond with the local workspace folder and its sub-structure.

In the Change Source Control dialog (*screenshot below*), you can change the source control system (**SCC Provider**), the local folder (**Local Path**), and the repository binding (**Server Name** and **Server Binding**).

Only after undoing the current binding can the settings be changed. Undo the current binding with the **Undo** button. All the settings are now editable.

![Change Source Control dialog](image)
Change source control settings as follows:

1. Use the **Browse** button to browse for the local folder and the **Select** button to select from among the installed source control systems.

2. After doing this you can bind the local folder to a repository database. Click the **Bind** button to do this. This pops up the connection dialog of your source control system.

3. If you have entered a **Logon ID**, this will be passed to the source control system; otherwise you might have to enter your logon details in the connection dialog.

4. Select the database in the repository that you wish to bind to this local folder. This setting might be spread over more than one dialog.

5. After the setting has been created, click **OK** in the Change Source Control dialog.
20.7 Source Control with Git

Support for Git as a source control system in XMLSpy is available through a third-party plug-in called GIT SCC plug-in (http://www.pushok.com/software/git.html).

At the time when this documentation is written, the GIT SCC plug-in plug-in is available for experimental use. Registration with the plug-in publisher is required in order to use the plug-in.

The GIT SCC plug-in enables you to work with a Git repository using the commands available in the Project | Source Control menu of XMLSpy. Note that the commands in the Project | Source Control menu of XMLSpy are provided by the Microsoft Source Control Plug-in API (MSSCCI API), which uses a design philosophy different from Git. As a result, the plug-in essentially intermediates between "Visual Source Safe"-like functionality and Git functionality. On one hand, this means that a command such as Get latest version may not be applicable with Git. On the other hand, there are new Git-specific actions, which are available in the "Source Control Manager" dialog box provided by the plug-in (under the Project | Source Control | Source Control Manager menu of XMLSpy).

Other commands that you will likely need to use frequently are available directly under the Project | Source Control menu.

The following sections describe the initial configuration of the plug-in, as well as the basic workflow:

- Enabling Git Source Control with GIT SCC Plug-in
- Adding a Project to Git Source Control
- Cloning a Project from Git Source Control
20.7.1 Enabling Git Source Control with GIT SCC Plug-in

To enable Git source control with XMLSpy, the third-party PushOK GIT SCC plug-in must be installed, registered, and selected as source control provider, as follows:

1. Download the plug-in installation file from the publisher's website (http://www.pushok.com), run it, and follow the installation steps.

2. On the Project menu of XMLSpy, click Change Source Control, and make sure PushOk GITSCC is selected as source control provider. If you do not see Push Ok GITSCC in the list of providers, it is likely that the installation of the plug-in was not successful. In this case, check the publisher's documentation for a solution.

3. When a dialog box prompts you to register the plug-in, click Registration and follow the wizard steps to complete the registration process.

20.7.2 Adding a Project to Git Source Control

You can save XMLSpy projects as Git repositories. The structure of files or folders that you add to the project would then correspond to the structure of the Git repository.

To add a project to Git source control:

1. Make sure that PushOK GIT SCC Plug-in is set as source control provider (see Enabling Git Source Control with GIT SCC Plug-in).

2. Create a new project using the menu command Project | Create Project.

3. Save the project to a local folder, for example C:\MyRepo\Project.spp

4. On the Project menu, under Source Control, click Add to Source Control.
5. Click OK.

6. Enter the text of your commit message, and click OK.

You can now start adding files and folders to your project. Note that all project files and folders must be under the root folder of the project. For example, if the project was created in the \C:\MyRepo\ folder, then only files under \C:\MyRepo should be added to the project. Otherwise, if you attempt to add to your project files that are outside the project root folder, a warning message is displayed:

20.7.3 Cloning a Project from Git Source Control
Projects that have been previously added to Git source control (see Adding a Project to Git Source Control) can be opened from the Git repository as follows:

1. Make sure that PushOK GIT SCC Plug-in is set as source control provider (see Enabling Git Source Control with GIT SCC Plug-in).
2. On the **Project** menu, click **Source Control | Open from Source Control**.
3. Enter the path or the URL of the source repository. Click **Check** to verify the validity of the path or URL.

![Open from Source Control Wizard](image)

4. Under **Local Path**, enter the path to local folder where you want the project to be created, and click **Next**. If the local folder exists (even if it is empty), the following dialog box opens:

![Question Dialog Box](image)

5. Click **Yes** to confirm, and then click **Next**.
6. Follow the remaining wizard steps, as required by your specific case.
7. When the wizard completes, a Browse dialog box appears, asking you to open the XMLSpy Project (*.spp) file. Select the project file to load the project contents into XMLSpy.
21 XMLSpy in Visual Studio

XMLSpy can be integrated into the Microsoft Visual Studio IDE versions 2010/2012/2013/2015/2017. This unifies the best of both worlds, integrating advanced XML editing capabilities with the advanced development environment of Visual Studio.

In this section, we describe:

- The broad installation process and the integration of the XMLSpy plugin in Visual Studio.
- Differences between the Visual Studio version and the standalone version.
- XMLSpy’s Debuggers in Visual Studio.
21.1 Installing the XMLSpy Plugin

To install the XMLSpy Plugin for Visual Studio, you need to do the following:

2. Install XMLSpy (Enterprise or Professional Edition).
3. Download and run the XMLSpy integration package for Microsoft Visual Studio. This package is available on the XMLSpy (Enterprise and Professional Editions) download page at www.altova.com. **(Please note:** You must use the integration package corresponding to your XMLSpy version (current version is 2019).)

Once the integration package has been installed, you will be able to use XMLSpy in the Visual Studio environment.

**How to enable the plug-in**

If the plug-in was not automatically enabled during the installation process, do the following:

1. Navigate to the directory where the Visual Studio IDE executable was installed, for example in C:\Program Files\MS Visual Studio\Common7\IDE
2. Enter the following command on the command-line `devenv.exe /setup`
3. Wait for the process to terminate normally before starting to use the application within Visual Studio.
21.2 Differences with XMLSpy Standalone

This section lists the ways in which the Visual Studio versions differ from the standalone versions of XMLSpy. The listing starts with features that are unsupported in the Visual Studio version, and continues with a listing of other ways in which the Visual Studio version differs from the standalone version.

- Unsupported features in Visual Studio
- Additional XMLSpy menus in Visual Studio
- Entry helpers in Visual Studio
- Same functionality, different command
- XMLSpy commands as Visual Studio commands

Unsupported features in Visual Studio

The following XMLSpy features are not available in Visual Studio:

- The Scripting environment (Tools | XMLSpy Options | Scripting) is currently not supported.
- Separate browser window (an option in the Tools | Options | View section) is not supported. This means the Text View and Browser View are always in the same window.
- The text state icons of Authentic View are not supported.
- All Source Control functionality.
- All comparison functionality (available in the Tools menu of the standalone version).

Additional XMLSpy menus in Visual Studio

The following commands are specific to XMLSpy in Visual Studio:

- View | XMLSpy Tool Windows
- View | XMLSpy View
- XMLSpy (includes Global Resources menu items)
- Tools | XMLSPY Options

Entry helpers (Tool windows in Visual Studio)

The entry helpers of XMLSpy are available as Tool windows in Visual Studio. The following points about them should be noted. (For a description of entry helpers and the XMLSpy GUI, see the section, Introduction.)

- You can drag entry helper windows to any position in the development environment.
- Right-clicking an entry helper tab allows you to further customize your interface. Entry helper configuration options are: dockable, hide, floating, and auto-hide.

Same functionality, different command

Some functionality of XMLSpy is available in Visual Studio under differently named commands. These are:

<table>
<thead>
<tr>
<th>XMLSpy</th>
<th>Visual Studio</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
XMLSpy commands as Visual Studio commands

Some XMLSpy commands are present as Visual Studio commands in the Visual Studio GUI. These are:

- **Undo, Redo**: These Visual Studio commands affect all actions in the Visual Studio development environment.
- **Projects**: XMLSpy projects are handled as Visual Studio projects.
- **Customize Toolbars, Customize Commands**: The Toolbars and Commands tabs (screenshot below) in the Customize dialog (Tools | Customize) contain both Visual Studio commands as well as XMLSpy commands.

- **Views**: In the View menu, the two commands, XMLSpy Tool Windows and XMLSpy View, contain options to toggle on entry helper windows and other sidebars, switch between the editing views, and toggle certain editing guides on and off.
- **XMLSpy Help**: This XMLSpy menu appears as a submenu in Visual Studio's Help menu.
21.3 XMLSpy's Debuggers in Visual Studio

XMLSpy contains an XSLT/XQuery Debugger (Enterprise and Professional editions) and a SOAP Debugger (Enterprise edition). A debugger process involves the display of more than one file (for example, XML, XSLT, and XSLT output files), all of which are displayed in Visual Studio as a single tabbed group. To make the debugging easier to follow, you can create one or more additional tab groups in Visual Studio. Do this as follows:

1. Click the tab you wish to separate from the single tabbed group, then drag and drop it somewhere in the currently active tab. This opens a pop-up menu which allows you to define the type of tab you want to create.

   ![New Horizontal Tab Group]

   ![New Vertical Tab Group]

   ![Cancel]

2. Select New Vertical Tab Group. This creates a new tab containing just the selected tab (screenshot below).
22 XMLSpy in Eclipse

Eclipse is an open source framework that integrates different types of applications delivered in the form of plugins.

The XMLSpy Plugin for Eclipse enables you to access the functionality of XMLSpy from within the Eclipse 4.7 / 4.8 / 4.9 / 4.10 Platform. It is available on Windows platforms. In this section, we describe how to install the XMLSpy Plugin for Eclipse and how to set up the XMLSpy perspective and XMLSpy Debugger perspectives. After you have done this, components of the XMLSpy GUI and XMLSpy menu commands will be available within the Eclipse GUI.

**Note:** Source Control functionality, which is available in the standalone version, is not supported in the Eclipse version.
### 22.1 Installing the XMLSpy Plugin for Eclipse

Before installing the XMLSpy Plugin for Eclipse, ensure that the following are already installed:

- XMLSpy Enterprise or Professional Edition.
- Java SE Runtime Environment 6.0 (JRE 6.0) or higher, which is required for Eclipse. See the [Eclipse website](https://eclipse.org) for more information. Install a 32-bit or 64-bit JRE to match your version of XMLSpy (32-bit or 64-bit).
- Eclipse Platform 4.7 / 4.8 / 4.9 / 4.10. Install a 32-bit or 64-bit Eclipse to match your version of XMLSpy (32-bit or 64-bit).

After these have been installed, you can install the XMLSpy Plugin for Eclipse, which is contained in the XMLSpy Integration Package (see below).

### XMLSpy Integration Package

The XMLSpy Plugin for Eclipse is contained in the XMLSpy Integration Package and is installed during the installation of the XMLSpy Integration Package. Install as follows:

1. Ensure that XMLSpy (Enterprise or Professional Edition), JRE, and Eclipse are already installed (see above).
2. From the [Components Download](https://www.altova.com/products/altova/xmlspy/download) page of the Altova website, download and install the XMLSpy Integration Package. There are two important steps during the installation; these are described in Steps 3 and 4 below.
3. During installation of the XMLSpy Integration Package, a dialog will appear asking whether you wish to install the XMLSpy Plugin for Eclipse (see screenshot below). Check the option and then click **Next**.

![XMLSpy Integration Package Setup](image-url)
4. In the next dialog ((Eclipse) Installation Location, screenshot below), you can choose whether the Install Wizard should integrate the XMLSpy Plugin into Eclipse during the installation (the *Automatically* option) or whether you will integrate the XMLSpy Plugin into Eclipse (via the Eclipse GUI) at a later time.

![installation location dialog](image)

We recommend that you let the Installation Wizard do the integration. Do this by checking the *Automatically* option and then browsing for the folder in which the Eclipse executable (*eclipse.exe*) is located. Click **Next** when done. If you choose to manually integrate XMLSpy Plugin for Eclipse in Eclipse, select the *Manually* option (screenshot below). See the section below for instructions about how to manually integrate from within Eclipse.
5. Complete the installation. If you set up automatic integration, the XMLSpy Plugin for Eclipse will be integrated in Eclipse and will be available when you start Eclipse the next time.

Manually integrating the XMLSpy plugin in Eclipse

To manually integrate the XMLSpy Plugin for Eclipse, do the following:

1. In Eclipse, click the menu command **Help | Install New Software**.
2. In the Install dialog that pops up (**screenshot below**), click the **Add** button.
3. In the Add Repository dialog that pops up (screenshot 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 \XMLSpy2019\eclipse\UpdateSite`, and providing a name such as ‘Altova XMLSpy’.
6. In the Work With combo box of the Install dialog, select the option -- All Available Sites --
(see screenshot below). This causes all available plugins to be displayed in the pane below. Check the top-level check box of the Altova category folder (see screenshot 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 XMLSpy Enterprise or Professional Edition license) is required for the XMLSpy plugin.) Then click Finish to complete the installation.

If there are problems with the plug-in (missing icons, for example), start Eclipse via the command line with the -clean flag.
Currently installed version
To check the currently installed version of the XMLSpy Plugin for Eclipse, select the Eclipse menu option Help | About Eclipse. Then select the XMLSpy icon.
22.2 XMLSpy Entry Points in Eclipse

The following entry points in Eclipse can be used to access XMLSpy functionality:

- **XMLSpy Perspective**, which provides XMLSpy's GUI features within the Eclipse GUI.
- **XMLSpy menu and toolbar**

**XMLSpy Perspective**

In Eclipse, a perspective is a configured GUI view with functionality from various applications. When the XMLSpy Plugin for Eclipse is integrated in Eclipse, a default XMLSpy perspective is automatically created. This perspective is a GUI that includes XMLSpy's GUI elements: its editing views, menus, entry helpers, and other sidebars.

When a file having a filetype associated with XMLSpy is opened (.xml, for example), this file can be edited in the XMLSpy perspective. Similarly, a file of another filetype can be opened in another perspective in Eclipse. Additionally, for any active file, you can switch the perspective, thus allowing you to edit or process that file in another environment. There are therefore two main advantage of perspectives:

1. Being able to quickly change the working environment of the active file, and
2. Being able to switch between files without having to open a new development environment (the associated environment is available in a perspective)

Working with the XMLSpy perspective involves the following:

- Switching to the XMLSpy perspective.
- Setting preferences for the XMLSpy perspective.
- Customizing the XMLSpy perspective.

**Switching to the XMLSpy perspective**

In Eclipse, select the command **Window | Open Perspective | Other**. In the dialog that pops up (screenshot below), select **XMLSpy**, and click **OK**.
The empty window or the active document will now have the XMLSpy perspective. This is how the user switches the perspective via the menu. To access a perspective faster from another perspective, the required perspective can be listed in the Open Perspective submenu, above the Other item; this setting is in the customization dialog (see further below).

Perspectives can also be switched when a file is opened or made active. The perspective of the application associated with a file's filetype will be automatically opened when that file is opened for the first time. Before the perspective is switched, a dialog appears asking whether you wish to have the default perspective automatically associated with this filetype. Check the Do Not Ask Again option if you wish to associate the perspective with the filetype without having to be prompted each time a file of this filetype is opened and then click OK.

Setting preferences for the XMLSpy perspective

The preferences of a perspective include: (i) a setting to automatically change the perspective when a file of an associated filetype is opened (see above), and (ii) options for including or excluding individual XMLSpy toolbars. To access the Preferences dialog, select the command Window | Preferences. In the list of perspectives in the left pane, select XMLSpy, then select the required preferences. Finish by clicking OK.

Customizing the XMLSpy perspective

The customization options enable you to determine what shortcuts and commands are included in the perspective. To access the Customize Perspective dialog of a perspective (screenshot below shows dialog for the XMLSpy perspective), make the perspective active (in this case the XMLSpy perspective), and select the command Window | Customize Perspective.
In the Tool Bar Visibility and Menu Visibility tabs, you can specify which toolbars and menus are to be displayed. In the Command Groups Availability tab, you can add command groups to their parent menus and to the toolbar. If you wish to enable a command group, check its check box. In the Shortcuts tab of the Customize Perspective dialog, you can set shortcuts for submenus. Select the required submenu in the Submenus combo box. Then select a shortcut category, and check the shortcuts you wish to include for the perspective. Click **OK** to complete the customization and for the changes to take effect.

### XMLSpy menu and toolbar

The **XMLSpy** menu contains commands that are relevant even if a document type recognized by XMLSpy is not currently open in Eclipse. In the standalone version of XMLSpy, some of these commands are in the **File** menu.

The **XMLSpy** toolbar contains the following buttons (screenshot below).

![XMLSpy toolbar](screenshot below)

These are for, from left: (i) opening the **XMLSpy** Help, and (ii) accessing **XMLSpy** commands (as an alternative to accessing them from the **XMLSpy** menu).
XMLSpy file formats and behavior of Eclipse views

When certain file types recognized by XMLSpy are active (in focus) in Eclipse, the "Elements", "Attributes", and "Entities" views appear with a name that is meaningful for that format. For example, when a .css file is active, the "Elements" view appears with the name "CSS Outline". The following table illustrates how view names change based on the active file:

<table>
<thead>
<tr>
<th>When this file format is active...</th>
<th>The &quot;Elements&quot; view becomes...</th>
<th>The &quot;Attributes&quot; view becomes...</th>
<th>The &quot;Properties&quot; view becomes...</th>
</tr>
</thead>
<tbody>
<tr>
<td>.css</td>
<td>CSS Outline</td>
<td>CSS Properties</td>
<td>HTML Elements</td>
</tr>
<tr>
<td>.xquery, .xq</td>
<td>XQuery Keywords</td>
<td>XQuery Variables</td>
<td>XQuery Functions</td>
</tr>
<tr>
<td>.xsd</td>
<td>Components</td>
<td>Details</td>
<td>Facets</td>
</tr>
</tbody>
</table>

If you close any of these views, you can restore them later using the menu command **Window | Show | View**. Note, however, that views are displayed in this menu with their generic name (that is, "Elements", "Attributes", and "Entities"). So, for example, in order to restore the view "CSS Outline", you would select **Show | View | Elements**.

As an alternative, reset the XMLSpy perspective to its default values, from **Window | Reset Perspective**.
22.3 **XMLSpy's Debugger Perspectives**

There are two debuggers in the Enterprise edition of XMLSpy (XSLT/XQuery and SOAP), and one debugger in the Professional edition of XMLSpy (XSLT/XQuery). Perspectives for these debuggers are available in Eclipse according to the XMLSpy edition that is currently installed.

To switch to a debugger perspective, select the command **Window | Open Perspective | Other**. In the dialog that pops up (screenshot below), select the debugger (for example, Debug XSLT/XQ), and click **OK**.

![Open Perspective dialog](image)

The empty window or the active document will now have the perspective of the selected debugger. This is how the user switches the perspective via the menu. To access a perspective faster from another perspective, the required perspective can be listed in the **Open Perspective** submenu, above the **Other** item; this setting is in the customization dialog.

For a description of how to use the debuggers, see the respective sections in this documentation: XSLT and XQuery, and WSDL and SOAP.
23  Menu Commands

The User Reference section contains a complete description of all XMLSpy menu commands and explains their use. We have tried to be comprehensive. If, however, you have questions which are not covered in the User Reference or other parts of this documentation, please look up the FAQs and Discussion Forums on the Altova website. If you cannot find a suitable answer at these locations, please do not hesitate to contact the Altova Support Center.

Standard Windows commands, such as (Open, Save, Cut, Copy and Paste) are in the File and Edit menus. These menus additionally contain XML- and Internet-related commands.
23.1 **File Menu**

The **File** menu contains commands for file operations, ordered as in most Windows applications. In addition to the standard **New**, **Open**, **Save**, **Print**, **Print Setup**, and **Exit** commands, XMLSpy also offers XML-specific and application-specific commands.

### 23.1.1 New

*This section:*

- **Icon and shortcut**
- **Description**
- **Templates for new documents**
- **Assigning a DTD or XML Schema to a new XML document**
- **Specifying the root element of a new XML document**
- **Assigning an SPS to a new XML document**

**Icon and shortcut**

<table>
<thead>
<tr>
<th>Icon:</th>
<th>Ctrl+N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortcut:</td>
<td>Ctrl+N</td>
</tr>
</tbody>
</table>
Description

The **New** command is used to create a new document. Clicking **New** opens the Create New Document dialog (screenshot below), in which you can select the type of document you wish to create. If the document type you wish to create is not listed in the dialog, select XML and change the file extension when you save the file. Note that you can add new document types to this dialog list using the **Tools | Options | File types section**.

![Create New Document Dialog](image)

Templates for new documents

The document-type list of the Create New Document dialog can also contain entries for user-defined document templates of any document type. These templates can be opened directly from the Create New Document dialog and edited. To create your own document template so that it appears in the list of document types in the Create New Document dialog, you first create the template document and then save it to the folder designated to contain document templates.

Create a document template as follows:

1. Open the `XMLSpy\Template` folder of the **application folder** using Windows Explorer or your preferred navigation tool, and select a rudimentary template file from among the files named `new.xxx` (where `.xxx` is a file extension, such as `.xml` or `.xslt`).
2. Open the file in XMLSpy, and modify the file as required. This file will be the template file.
3. After you have finished, select **File | Save as** to save the file back to the `\Template` folder with a suitable name, say `MyXMLTemplate.xml`. You now have a document template called `MyXMLTemplate`, which will appear in the list of document types in the Create New Document dialog.
4. To open the template, select **File | New**, and then the template (*my-xml*, in this case).

To delete a document template from the list of document types, delete (or move) the template file from the template folder.

**Assigning a DTD or XML Schema to a new XML document**

When you create a new document of a certain type via the Create New Document dialog, the document is automatically opened with the correct DTD or XML Schema association. For example, an XHTML file will be opened with the DTD `http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd` associated with it. And an XML Schema (.xsd) file is associated with the `http://www.w3.org/2001/XMLSchema` schema document.

If you are creating a new document for which the schema is not known (for example, an XML file), then you are prompted to associate a schema (DTD or XML Schema) to the new document (*screenshot below*).
If, in the dialog, you select DTD or XML Schema and click **OK**, you can browse for the schema you want. Clicking **Cancel** creates a new file that is not associated with any schema.

**Specifying the root element of a new XML document**

If an XML Schema is selected as the associated schema of an XML document and if this schema has more than one global element, each of these is a potential root element. In this case, the Select a Root Element dialog (*screenshot below*) pops up, in which you can select which global element is to be the root element of the XML document. In the screenshot below, the `OrgChart` global element is selected.

Clicking **OK** now will create a new XML document with this element (`OrgChart`) as its root element.

**Assigning an SPS to a new XML document**

When a new XML document is created, you can associate a StyleVision Power Stylesheet (.sps file) to view the document in Authentic View. In the Create New Document dialog (*see first screenshot in this section*), when you click the **Select StyleVision Stylesheet**, the Create New Document dialog (*shown below*) appears.
You can browse for the required SPS in the folder tabs, or you can click the Browse button to navigate for and select the SPS.

### 23.1.2 Open

#### Icon and shortcut

<table>
<thead>
<tr>
<th>Icon:</th>
<th>![Image]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortcut:</td>
<td>Ctrl+O</td>
</tr>
</tbody>
</table>

#### Description

The **Open** command pops up the familiar Windows Open dialog, and allows you to open any XML-related document or text document. In the Open dialog, you can select more than one file to open. Use the Files of Type combo box to restrict the kind of files displayed in the dialog box. (The list of available file types can be configured in the File Types section of the Options dialog (**Tools | Options**).) When an XML file is opened, it is checked for well-formedness. If the file is not well-formed, you will get a file-not-well-formed error. Fix the error and select the menu command **XML | Check Well-Formedness (F7)** to recheck. If you have opted for automatic validation upon opening and the file is invalid, you will get an error message. Fix the error and select the menu command **XML | Validate XML (F8)** to revalidate.

#### Selecting and saving files via URLs and Global Resources

In several File Open and File Save dialogs, you can choose to select the required file or save a file via a URL or a global resource (*see screenshot below*). Click **Switch to URL** or **Switch to Global Resource** to go to one of these selection processes.
Selecting files via URLs
To select a file via a URL (either for opening or saving), do the following:

1. Click the **Switch to URL** command. This switches to the URL mode of the Open or Save dialog (*the screenshot below shows the Open dialog*).
2. Enter the URL you want to access in the Server URL field (screenshot above). If the server is a Microsoft® SharePoint® Server, check the Microsoft® SharePoint® Server check box. See the Microsoft® SharePoint® Server Notes below for further information about working with files on this type of server.

3. If the server is password protected, enter your User-ID and password in the User and Password fields.

4. Click Browse to view and navigate the directory structure of the server.

5. In the folder tree, browse for the file you want to load and click it.
The file URL appears in the File URL field (see screenshot above). The **Open** or **Save** button only becomes active at this point.

6. Click **Open** to load the file or **Save** to save it.

**Note the following:**

- The Browse function is only available on servers which support WebDAV and on Microsoft SharePoint Servers. The supported protocols are FTP, HTTP, and HTTPS.
- To give you more control over the loading process when opening a file, you can choose to load the file through the local cache or a proxy server (which considerably speeds up the process if the file has been loaded before). Alternatively, you may want to reload the file if you are working, say, with an electronic publishing or database system; select the **Reload** option in this case.

**Microsoft® SharePoint® Server Notes**

Note the following points about files on Microsoft® SharePoint® Servers:

- In the directory structure that appears in the Available Files pane (screenshot below), file icons have symbols that indicate the check-in/check-out status of files.
Right-clicking a file pops up a context menu containing commands available for that file (screenshot above).

- The various file icons are shown below:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Icon]</td>
<td>Checked in. Available for check-out.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Checked out by another user. Not available for check-out.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Checked out locally. Can be edited and checked-in.</td>
</tr>
</tbody>
</table>

- After you check out a file, you can edit it in your Altova application and save it using **File | Save (Ctrl+S)**.
- You can check-in the edited file via the context menu in the Open URL dialog (see screenshot above), or via the context menu that pops up when you right-click the file tab in the Main Window of your application (screenshot below).

- When a file is checked out by another user, it is not available for check out.
- When a file is checked out locally by you, you can undo the check-out with the Undo Check-Out command in the context menu. This has the effect of returning the file unchanged to the server.
- If you check out a file in one Altova application, you cannot check it out in another Altova application. The file is considered to be already checked out to you. The available commands at this point in any Altova application supporting Microsoft®
SharePoint® Server will be: Check In and Undo Check Out.

Opening and saving files via Global Resources

To open or save a file via a global resources, click Switch to Global Resource. This pops up a dialog in which you can select the global resource. These dialogs are described in the section, Using Global Resources. For a general description of Global Resources, see the Global Resources section in this documentation.

23.1.3 Reload

Icon

Description

Reloads any open documents that have modified outside XMLSpy. If one or more documents is modified outside XMLSpy, a prompt appears asking whether you wish to reload the modified document/s. If you choose to reload, then any changes you may have made to the file since the last time it was saved will be lost.

23.1.4 Encoding

The Encoding command lets you: (i) view the current encoding of the active document (XML or non-XML), and (ii) select a different encoding with which the active document will be saved the next time.

In XML documents, if you select a different encoding than the one currently in use, the encoding attribute in the XML declaration will be modified accordingly. For two-byte and four-byte character encodings (UTF-16, UCS-2, and UCS-4) you can also specify the byte-order to be used for the file. Another way to change the encoding of an XML document is to directly edit the encoding attribute of the document's XML declaration. Default encodings for existing and new XML and non-XML documents can be set in the Encoding section of the Options dialog.

Note: When saving a document, XMLSpy automatically checks the encoding specification and enables you to select the required encoding via the Encoding dialog. If your document
contains characters that cannot be represented in the selected encoding and you attempt
to save the file, you will get a warning message to this effect.

23.1.5 Close, Close All, Close All But Active

Close
The Close command closes the active document window. If the file was modified (indicated by an asterisk * after the file name in the title bar), you will be asked if you wish to save the file first.

Close All
The Close All command closes all open document windows. If any document has been modified (indicated by an asterisk * after the file name in the title bar), you will be asked if you wish to save the file first.

Close All But Active
The Close All But Active command closes all open document windows except the active document window. If any document has been modified (indicated by an asterisk * after the file name in the title bar), you will be asked if you wish to save the file first.

23.1.6 Save, Save As, Save All

Icons and shortcuts

<table>
<thead>
<tr>
<th>Command</th>
<th>Icon</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save</td>
<td>![Save Icon]</td>
<td>Ctrl+S</td>
</tr>
<tr>
<td>Save All</td>
<td>![Save All Icon]</td>
<td></td>
</tr>
</tbody>
</table>

Save
The Save command (Ctrl+S) saves the contents of the active document to the file from which it has been opened. When saving a document, the file is automatically checked for well-formedness. The file will also be validated automatically if this option has been set in the File section of the Options dialog (Tools | Options). The XML declaration is also checked for the encoding specification, and this encoding is applied to the document when the file is saved.

Save As
The Save As command pops up the familiar Windows Save As dialog box, in which you enter the name and location of the file you wish to save the active file as. The same checks and validations occur as for the Save command.

Save All
The Save All command saves all modifications that have been made to any open documents. The command is useful if you edit multiple documents simultaneously. If a document has not been saved before (for example, after being newly created), the Save As dialog box is presented
Selecting and saving files via URLs and Global Resources

In several File Open and File Save dialogs, you can choose to select the required file or save a file via a URL or a global resource (see screenshot below). Click **Switch to URL** or **Switch to Global Resource** to go to one of these selection processes.

Selecting files via URLs

To select a file via a URL (either for opening or saving), do the following:

1. Click the **Switch to URL** command. This switches to the URL mode of the Open or Save dialog (the screenshot below shows the Open dialog).
2. Enter the URL you want to access in the Server URL field (screenshot above). If the server is a Microsoft® SharePoint® Server, check the Microsoft® SharePoint® Server check box. See the Microsoft® SharePoint® Server Notes below for further information about working with files on this type of server.

3. If the server is password protected, enter your User-ID and password in the User and Password fields.

4. Click Browse to view and navigate the directory structure of the server.

5. In the folder tree, browse for the file you want to load and click it.
6. Click **Open** to load the file or **Save** to save it.

**Note the following:**

- The Browse function is only available on servers which support WebDAV and on Microsoft SharePoint Servers. The supported protocols are FTP, HTTP, and HTTPS.
- To give you more control over the loading process when opening a file, you can choose to load the file through the local cache or a proxy server (which considerably speeds up the process if the file has been loaded before). Alternatively, you may want to reload the file if you are working, say, with an electronic publishing or database system; select the **Reload** option in this case.

**Microsoft® SharePoint® Server Notes**

Note the following points about files on Microsoft® SharePoint® Servers:

- In the directory structure that appears in the Available Files pane (**screenshot below**), file icons have symbols that indicate the check-in/check-out status of files.
Right-clicking a file pops up a context menu containing commands available for that file (screenshot above).

- The various file icons are shown below:
  
<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Checked in" /></td>
<td>Checked in. Available for check-out.</td>
</tr>
<tr>
<td><img src="image" alt="Checked out" /></td>
<td>Checked out by another user. Not available for check-out.</td>
</tr>
<tr>
<td><img src="image" alt="Edited locally" /></td>
<td>Checked out locally. Can be edited and checked-in.</td>
</tr>
</tbody>
</table>

- After you check out a file, you can edit it in your Altova application and save it using **File | Save (Ctrl+S)**.
- You can check-in the edited file via the context menu in the Open URL dialog (see screenshot above), or via the context menu that pops up when you right-click the file tab in the Main Window of your application (screenshot below).

- When a file is checked out by another user, it is not available for check out.
- When a file is checked out locally by you, you can undo the check-out with the Undo Check-Out command in the context menu. This has the effect of returning the file unchanged to the server.
- If you check out a file in one Altova application, you cannot check it out in another Altova application. The file is considered to be already checked out to you. The available commands at this point in any Altova application supporting Microsoft®
SharePoint® Server will be: **Check In** and **Undo Check Out**.

- Opening and saving files via Global Resources
  
  To open or save a file via a global resources, click **Switch to Global Resource**. This pops up a dialog in which you can select the global resource. These dialogs are described in the section, *Using Global Resources*. For a general description of Global Resources, see the [Global Resources](#) section in this documentation.

### 23.1.7 Send by Mail

#### Icon

![Icon](image)

#### Description

The **Send by Mail** command lets you send XML document/s or selections from an XML document by e-mail. Depending on what kind it is, a document or selection can be sent as an attachment, content, or as a link. See the table below for details.

<table>
<thead>
<tr>
<th>What can be sent</th>
<th>How it can be sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active XML document</td>
<td>As e-mail attachment</td>
</tr>
<tr>
<td>Selection in active XML document</td>
<td>As e-mail attachment or e-mail content</td>
</tr>
<tr>
<td>One or more files in Project window</td>
<td>As e-mail attachment</td>
</tr>
<tr>
<td>One or more URLs in Project window</td>
<td>As e-mail attachment or link</td>
</tr>
</tbody>
</table>

When the **Send by Mail** command is invoked on a selection in the active XML document, the Send by Mail dialog *(screenshot below)* pops up and offers the sending options shown in the screenshot. If the **Send by Mail** command is invoked with no text selected in the active file, then the **Whole File** radio button *(refer to screenshot above)* is the only option that is enabled; the other options are disabled.
Since files sent from the Project window are always sent as e-mail attachments only, the Send by Email dialog is skipped and an e-mail is opened that has the selected file/s as attachments. URLs in the project window can be sent as an attachment or as a link (see screenshot below). Select how the URL is to be sent and click OK.

### 23.1.8 Print

**Icon and shortcut**

<table>
<thead>
<tr>
<th>Icon:</th>
<th>![print_icon]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortcut:</td>
<td>Ctrl+P</td>
</tr>
</tbody>
</table>

**Description**

The Print command opens the Print dialog box, in which you can select printing options for printing the currently active document.

Clicking the Print command in Grid View opens a Print options dialog (screenshot below), which enables you to set printing options for the XML document. Clicking Print in this dialog takes you to the Print dialog for printer options.
The available options for Grid View printing are described below:

- In the Types pane, you select the items you wish to have appear in the output.
- For the What option, you specify whether the current selection or the entire file is to be printed.
- The Expand option allows you to print the document as is, or with all descendant elements expanded fully.
- The Contents option enables you to choose between printing contents of all nodes or printing node names only.
- In the If Contents Are Wider Than Page pane, you select what to do if contents are wider than the page. The Split Pages option prints the entire document at normal size, splitting contents over pages both horizontally and vertically. The pages could then be glued together to form a poster. The First Page option prints only the first, left-hand page of the print area. The area that overflows horizontally is not printed. This option is useful if most of the important information in your Grid View of the document is contained on the left side. The Shrink Horizontally option reduces the size of the output (proportionally) until it fits horizontally on the page; the document may run on for several pages. The Shrink Both option shrinks the document in both directions until it fits exactly on one sheet.
- The Print button prints the document with the selected options.
- The Preview button opens a print preview window that lets you view the final output before committing it to paper.
- The Print Setup button opens the Print Setup dialog box and allows you to adjust the paper format, orientation, and other printer options for this print job only. Also see the Print Setup command.

Note: You can change column widths in Grid View to optimize the print output.

Program logo

If you have a purchased license, you can turn off the program logo, copyright notice, and registration details when printing a document from XMLSpy. This option is available in the View section of the Options dialog.
23.1.9 Print Preview, Print Setup

Print Preview

The Print Preview command clicked in Text View, Authentic View, and Browser View opens a print preview of the currently active document. From Grid View and Schema View, it opens the Print dialog box, in which you can select print options and then click the Preview button to get the print preview.

In Print Preview mode, the Print Preview toolbar at top left of the preview window provides print- and preview-related options. The preview can be magnified or miniaturized using the the Zoom In and Zoom Out buttons. When the page magnification is such that an entire page length fits in a preview window, then the One Page / Two Page button toggles the preview to one or two pages at a time. The Next Page and Previous Page buttons can be used to navigate among the pages. The toolbar also contains buttons to print all pages and to close the preview window.

Note: To enable background colors and images in Print Preview, do the following: (i) In the Tools menu of Internet Explorer, click Internet Options, and then click the Advanced tab; (ii) In the Settings box, under Printing, select the Print background colors and images check box, and (iii) Then click OK.

Print Setup

The Print Setup command, displays the printer-specific Print Setup dialog box, in which you specify such printer settings as paper format and page orientation. These settings are applied to all subsequent print jobs.

![Print Setup Dialog Box](image-url)
23.1.10 Recent Files, Exit

Recent Files
At the bottom of the File menu is a list of the nine most recently used files, with the most recently opened file shown at the top of the list. You can open any of these files by clicking its name. To open a file in the list using the keyboard, press Alt+F to open the File menu, and then press the number of the file you want to open.

Exit
Quits XMLSpy. If you have any open files with unsaved changes, you are prompted to save these changes. XMLSpy also saves modifications to program settings and information about the most recently used files.
23.2 **Edit Menu**

The **Edit** menu contains commands for editing documents in XMLSpy. These include the familiar **Undo**, **Redo**, **Cut**, **Copy**, **Paste**, **Delete**, **Select All**, **Find**, **Find Next** and **Replace** commands.

XMLSpy also offers special commands to:

- copy a selection to the clipboard as XML-Text,
- copy as structured text
- copy an XPath selector to the selected item to the clipboard.
- insert and remove bookmarks, and to navigate to bookmarks.
23.2.1 Undo, Redo

Icons and shortcuts

<table>
<thead>
<tr>
<th>Command</th>
<th>Icon</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undo</td>
<td><img src="image" alt="Undo Icon" /></td>
<td>Ctrl+Z</td>
</tr>
<tr>
<td>Redo</td>
<td><img src="image" alt="Redo Icon" /></td>
<td>Ctrl+Y</td>
</tr>
</tbody>
</table>

Undo

The **Undo** command contains support for unlimited levels of Undo. Every action can be undone and it is possible to undo one command after another. The Undo history is retained after using the **Save** command, enabling you go back to the state the document was in before you saved your changes. You can step backwards and forwards through this history using the **Undo** and **Redo** commands (see **Redo command below**).

Redo

The **Redo** command allows you to redo previously undone commands, thereby giving you a complete history of work completed. You can step backwards and forwards through this history using the **Undo** and **Redo** commands.

23.2.2 Cut, Copy, Paste, Delete

Icons and shortcuts

<table>
<thead>
<tr>
<th>Command</th>
<th>Icon</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td><img src="image" alt="Cut Icon" /></td>
<td>Ctrl+X or Shift+Del</td>
</tr>
<tr>
<td>Copy</td>
<td><img src="image" alt="Copy Icon" /></td>
<td>Ctrl+C</td>
</tr>
<tr>
<td>Paste</td>
<td><img src="image" alt="Paste Icon" /></td>
<td>Ctrl+V</td>
</tr>
<tr>
<td>Delete</td>
<td><img src="image" alt="Delete Icon" /></td>
<td>Del</td>
</tr>
</tbody>
</table>

Cut

The **Cut** command copies the selected text or items to the clipboard and deletes them from their present location.

Copy

The **Copy** command copies the selected text or items to the clipboard. This can be used to duplicate data within XMLSpy or to move data to another application.

**Note:** When copying from Grid View, the selection is copied using one of two methods: **Copy as XML-Text** and **Copy as Structured Text**. The former copies the selection as XML text; the latter copies the selection as a table. Which of these two methods is used when...
the **Copy** command is invoked is specified in the [Editing section of the Tools | Options dialog](#).

**Paste**
The **Paste** command inserts the contents of the clipboard at the current cursor position.

**Delete**
The **Delete** command deletes the currently selected text or items without placing them in the clipboard.

### 23.2.3 Copy as XML Text

The **Copy as XML Text** command copies XML data from Grid View as XML text. Highlight any XML data in Standard Grid View or in Table View (which is another view available in Grid View), and select the **Copy As XML Text** command. The data is copied to the clipboard as XML text (as in the listing below). You can then paste this text in another document.

```
<row>
  <para align="left">
    <bold>Check the FAQ</bold>
  </para>
  <para>
    <link mode="internal">
      <link_section>support</link_section>
      <link_subsection>faq30</link_subsection>
      <link_text>XMLSPY 4.0 FAQ</link_text>
    </link>
    <link mode="internal">
      <link_section>support</link_section>
      <link_subsection>faq25</link_subsection>
      <link_text>XMLSPY 3.5 FAQ</link_text>
    </link>
  </para>
</row>
```

The **Copy as XML Text** command automatically formats text using the currently active settings for saving a file. These settings can be modified in the **Save File** section of the File section of the **Options** dialog (Tools | Options).

The same effect as that of the **Copy as XML Text** command can be obtained by switching to Text View and copying an XML text fragment with **Ctrl+C** (Edit | Copy). From Grid View, you can also use the **Copy as Structured Text** command to copy XML data in its tabular Grid View representation.

### 23.2.4 Copy as Structured Text

The **Copy as Structured Text** command copies elements to the clipboard as they appear on screen. This command is especially useful for copying table-like data from the Table View of Grid View into another application that holds data in tabular form (such as a spreadsheet application).
• **Copying individual rows from Table View**
• **Copying a whole table from Table View**
• **Copying from standard Grid View**

### Copying individual rows from Table View

In the screenshot below two rows in the **Table View** of Grid View are copied as structured text.

![Screenshot of Table View](image)

The next two screenshots below show the data that was copied as structured text in the Grid View screenshot shown above now pasted in a Microsoft Excel document and in a Notepad document.

![Screenshot of Excel](image)

![Screenshot of Notepad](image)

Notice that while Excel *(screenshot above)* automatically formats each piece of cell text on the basis of its lexical form of text, Notepad *(screenshot below)* pastes all cell text as strings.

### Copying a whole table from Table View

In the screenshot below, a whole table is selected (by clicking the **item (6)** node entry).
The result is as for individual rows above, except that the table's column titles are now also copied. The screenshot below shows the copied data pasted into a Microsoft Excel worksheet. Note that, in the XML document, the column headers are element or attribute names.

Copying from Standard Grid View

The examples shown above copied data from the Table View of Grid View (that is, with Table View toggled on in Grid View). In Standard Grid View (Table View toggled off), the Copy as Structured Text also copies the selected data as displayed in Grid View. However, the structure is not a table. For example, in the Standard Grid View screenshot below, the items node, which has two item children, is selected.
When the copied text is pasted in Microsoft Excel, the worksheet will look like the screenshot below.

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>items</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>item</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>partNum</td>
<td>833-AA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>productName</td>
<td>Lapis necklace</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>quantity</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>price</td>
<td>99.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>shipDate</td>
<td>1999-12-05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>po:comment</td>
<td>Need this for the holidays!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>item</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>partNum</td>
<td>748-OT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>productName</td>
<td>Diamond heart</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>quantity</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>price</td>
<td>248.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>shipDate</td>
<td>2000-02-14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>po:comment</td>
<td>Valentine's day packaging.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

In addition to copying the XML data as structured text with the Copy as Structured Text command, data can also be [copied from Grid View as XML text](#).
23.2.5 Copy XPath

The **Copy XPath** command is available in Text View and Grid View, and creates an XPath expression that locates the currently selected node/s in the document, and copies the XPath expression to the clipboard. This enables you to paste the XPath expression into a document (for example, in an XSLT document). All expressions start from the document root. For example, if an element called **LastName** of the third **Person** element of the second **Company** element is selected, the XPath expression that is copied would be: `/Companies/Company[2]/Person[3]/LastName`

**Note:** In Grid View the **Copy XPath** command can also be accessed via the context menu.

23.2.6 Copy XPointer

The **Copy XPointer** command is available in Text View and Grid View. It creates an `element()` scheme XPointer for the currently selected node/s and copies it to the clipboard. This enables you to paste the XPointer into a document (for example, in the `xpointer` attribute of an XInclude element in an XML document).

The `element()` scheme of XPointer returns results in the form `element(/1/3)`. This XPointer selects the third child of the document element (aka root element). You should note the following points:

- Attributes cannot be represented using the `element()` scheme. If an attribute is selected in XMLSpy, the following happens: In Grid View, the **Copy XPointer** command is disabled; in Text View, the XPointer of the parent element of that attribute is generated.
- XPointers for multiple elements cannot be generated. If multiple elements are selected, the situation is resolved as follows: In Grid View, the **Copy XPointer** command is disabled. In Text View, the XPointer of the parent element of the selection is generated.

**Note:** In Grid View the **Copy XPointer** command can also be accessed via the context menu.

23.2.7 Insert

Mousing over or selecting the **Insert** command rolls out a submenu with three commands (described below):

- **Insert File Path**
- **Insert XInclude**
- **Insert Encoded External File**

**Insert File Path**

The **File Path** command is enabled in the Text View and Grid View of documents of any file type. Using it, you can insert the path to a file at the cursor selection point. Clicking the command pops up a dialog (**screenshot below**) in which you select the required file.
The required file can be selected in one of the following ways: (i) by browsing for the file, URL, or global resource (use the **Browse** button); (ii) by selecting the window in which the file is open (the **Window** button). When done, click **OK**. The path to the selected file will be inserted in the active document at the cursor selection point.

### Insert XInclude

The **XInclude** command is available in Text View and Grid View, and enables you to insert a new XInclude element at the cursor selection point in Text View, or before the selected item in both Text View and Grid View. If in Grid View the current selection is an attribute, the XInclude element is inserted after the attribute and before the first child element of the attribute's parent element. Selecting this command pops up the XInclude dialog (**snapshot below**).

![XInclude dialog](image)

The XML file to be included is entered in the **href** text box (alternatively, you can browse for the file by clicking the **Browse (..)** button to the right of the text box). The filename will be entered in the XML document as the value of the **href** attribute. The **parse**, **xpointer**, and **encoding** attributes of the XInclude element (**xi:include**), and the **fallback** child element of **xi:include** can also be inserted via the dialog. Do this by first checking the appropriate check box and then...
selecting/entering the required values. In the case of the fallback element, checking its check box only inserts the empty element. The content of the fallback element must be added subsequently in one of the editing views.

The parse attribute determines whether the included document is to be parsed as XML or text. (XML is the default value and therefore need not be specified.) The xpointer attribute identifies a specific fragment of the document located with the href attribute; it is this fragment that will be included. The encoding attribute specifies the encoding of the included document so that XMLSpy can transcode this document (or the part of it to be included) into the encoding of the including document. The contents of the fallback child element replace the xi:include element if the document to be included cannot be located.

Here is an example of an XML document that uses XInclude to include two XML documents:

```xml
<?xml version="1.0" encoding="UTF-16"?>
<AddressBook xsi:schemaLocation="http://www.altova.com/sv/myaddresses AddressBook.xsd"
    xmlns="http://www.altova.com/stylevision/tutorials/myaddresses"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xi="http://www.w3.org/2001/XInclude">
    <xi:include href="BusinessAddressBook.xml"/>
    <xi:include href="PersonalAddressBook.xml"/>
</AddressBook>
```

When this XML document is parsed, it will replace the two XInclude elements with the files specified in the respective href attributes.

xml:base

When the XML validator of XMLSpy reads an XML document and encounters the include element in the XInclude namespace (hereafter xi:include), it replaces this element (xi:include) with the XML document named in the href attribute of the xi:include element. The document element (root element) of the included XML document (or the element identified by an XPointer) will be included with an attribute of xml:base in order to preserve the base URIs of the included element. If the resulting XML document (containing the included XML document/s or tree fragment/s) must be valid according to a schema, then the document element of the included document (or the top-level element of the tree fragment) must be created with a content model that allows an attribute of xml:base. If, according to the schema, the xml:base attribute is not allowed on this element, then the resulting document will be invalid. How to define an xml:base attribute in an element's content model using XMLSpy's Schema View is described in the xml:Prefixed Attributes section of the Schema View section of the documentation.

XPointers

XMLSpy supports XPointers in XInclude. The relevant W3C recommendations are the XPointer Framework and XPointer element() Scheme recommendations. The use of an XPointer in an XInclude element enables a specific part of the XML document to be included, instead of the entire XML document. XPointers are used within an XInclude element as follows:

```xml
<xi:include href="PersonalAddressBook.xml" xpointer="element(usa)"/>
<xi:include href="BusinessAddressBook.xml" xpointer="element(/1/1)"/>
<xi:include href="RobbsAddressBook.xml" xpointer="element(usa/3/1)"/>
<xi:include href="PatsAddressBook.xml" xpointer="element(usa)
```
In the element() scheme of XPointer, an NCName or a child sequence directed by integers may be used.

- In the first `xi:include` element listed above, the `xpointer` attribute uses the element scheme with an NCName of `usa`. According to the XPointer Framework, this NCName identifies the element that has an ID of `usa`.
- In the second `xi:include` listed above, the `xpointer` attribute with a value of `element(/1/1)` identifies, in the first step, the first child element of the document root (which, if the document is well-formed, will be its document (or root) element). In the second step, the first child element of the element located in the previous step is located; in our example, this would be the first child element of the document element.
- The `xpointer` attribute of the third `xi:include` listed above uses a combination of NCName and child sequence. This XPointer locates the first child element of the third child element of the element having an ID of `usa`.
- If you are not sure whether your first XPointer will work, you can back it up with a second one as shown in the fourth `xi:include` listed above: `xpointer="element(usa) element(/1/1)"`. Here, if there is no element with an ID of `usa`, the back-up XPointer specifies that the first child element of the document element is to be selected. Additional backups are also allowed. Individual XPointers may not be separated, or they may be separated by whitespace: for example, `xpointer="element(usa) element(addresses/1) element(/1/1)"`.

**Note:** The namespace binding context is not used in the element() scheme because the element() scheme does not support qualified names.

### Insert Encoded External File

The **Encoded External File** command is available in Text View and Grid View. It enables an external file to be included as encoded Base-16 or Base-64 text at any location in the XML document. This feature enables external files to be embedded in the XML document.

Clicking the **Insert | Encoded External File** command pops up the Insert Encoded External File dialog (screenshot below).

![Insert Encoded External File dialog](screenshot)

You can browse for or enter the name of the external file to be encoded and embedded. Either a Base-16 or Base-64 encoding must be specified. If you wish to enclose the encoded text in an element, then check the Create Element check box and specify the name of the desired element in the Create Element text box. If the Create Element check box is not checked, then the encoded text will be inserted directly at the cursor location.

On clicking **OK**, the encoded text of the selected file is inserted at the cursor location, with an
enclosing element if this has been specified.

```
<iVBORw0KGgoAAAANSUhEUgAAABAAAAAQAQMAAAAlPW0iAAAABlBMVEUAAAD/+l2Z/dAAAAM0lEQVR4nGP4/5/h/1+G/58ZDrAz3D/McH8yw83NDDeNGe4U9C9zwz3gLMDMA/A6P9/AFGGFyjOXZtQAAAAAElFTkSuQmCC</i>
```

The listing above shows the encoded text of a PNG image file. An `img` element was created around the encoded text.

### 23.2.8 Pretty-Print

**Icon**

![Icon](image.png)

**Description**

The **Pretty-Print** command re-formats your XML document in Text View. Two formatting options are available: which one is used depends upon whether the `Use Indentation` check box in the **View section of the Options dialog** *(Tools | Options)* is currently checked or not:

- **Use Indentation** checked: The document is re-formatted to give a structured display, indenting each deeper level in the hierarchy by an additional amount of the specified indentation space (this space being specified in the **Text View Settings dialog** *(View | Text View Settings)*). Indentation enables a clearer view of the document structure.
- **Use Indentation** unchecked: The document is re-formatted so that each new line is left-aligned.

**Procedure for using Pretty-Print**

To set up a structured, indented view of the XML document via Pretty-Print, do the following:

1. In the **View section of the Options dialog** *(Tools | Options)*, check the **Use Indentation** check box.
2. In the **Text View Settings dialog** *(View | Text View Settings)*, set the tab size you want for the indentation of the pretty-printed text.
3. In the **File section of the Options dialog** *(Tools | Options)*, enter the elements for which no output formatting (indentation) is wanted.
4. Click the **Pretty-Print** command (this command).

To re-format the document so that all lines are left-aligned, uncheck the **Use Indentation** check box.

**Note the following points**

- The XML document must be well-formed for this command to work.
- Pretty-printing adds spaces or tabs to the document when the document is saved.
- If pretty-printing has been switched on *(Tools | Options | View | Use Indentation)* and if you change from Text View to Grid View and back to Text View, then the document will...
be pretty-printed automatically. There is no need to select the Pretty-Print command.
- To remove all whitespace (new lines and indentation) created with the Pretty-Print command, use the **Strip Whitespaces** command.

23.2.9 **Strip Whitespaces**

**Icon**

![Icon]

**Description**

The **Strip Whitespaces** command strips all whitespace from the document. This can help reduce file size. This command can be useful if you wish to remove whitespace generated by the **Pretty-Print** command.

23.2.10 **Select All**

The **Select All** command (Ctrl+A) selects the contents of the entire document.

23.2.11 **Find, Find Next**

**Icons and shortcuts**

<table>
<thead>
<tr>
<th>Command</th>
<th>Icon</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find</td>
<td>![Icon]</td>
<td>Ctrl+F</td>
</tr>
<tr>
<td>Find Next</td>
<td>![Icon]</td>
<td>F3</td>
</tr>
</tbody>
</table>

**Find**

The **Find** command displays the Find dialog, in which you can specify the string you want to find and other options for the search. Depending on the current view, the Find dialog displays different options. To find text, enter the text in the Find field or use the combo box to select from one of the last 10 search criteria, and then specify the options for the search.

The **Find** and **Find Next** commands can also be used to find file and folder names when a project is selected in the Project window.

**Grid View**

In Grid View, the following dialog box appears. Select the options you require or select a radio button. The options available are described below.
The *Types* pane allows you to select what XML document nodes or components you wish to include in the search. This enables the search to skip particular node types and thus go faster. The **Set All** button selects all node types; the **Clear All** button deselects all node types.

The *Search In* pane allows you to define whether the names of a node, the contents of a node, or both should be searched for the input text string.

The *Settings* pane enables you to define whether the search should be case-sensitive and/or match the entire input string.

The *Where* pane allows you to define the scope of the search (the whole file or the selected text).

The *Direction* option specifies the search direction.

---

**Text View**

For a description of the Find function of editable Text Views, see the section *Text View*. The Find dialog of non-editable Text Views (such as in the XPath/XQuery window) is shown in the screenshot below.

The available options are as follows:
- **Match whole word only**: Only the exact words in the text will be matched. For example, for the input string `fit`, with **Match whole word only** checked, only the word `fit` will match the find string; the `fit` in `fitness`, for example, will not.
- **Match case**: Case-sensitive search (**Address** is not the same as **address**).
- **Regular expression**: Searches for text specified by the regular expression you enter in the text box. See [Regular expressions](#) below for a description of regular expressions.

Note the following points:

- The Find dialog is "modeless", which means that it can remain open while you continue to use Text View. Pressing **Enter** while the dialog box is open, closes the dialog box. If text is marked prior to opening the dialog box, then the marked text is automatically inserted into the Find What text box.
- Once the Find dialog is closed, you can repeat the current search by pressing **F3** for a forwards search, or **Shift+F3** for a backwards search.
- The **Unfold** button to the right of the **Find What** input box (the button marked `>`) opens a secondary window which you can use to enter regular expressions in the **Find What** input box.

### Regular expressions

You can use regular expressions to further refine your search criteria. A pop-up list is available to help you build regular expressions. To access this list, click the **Unfold** button to the right of the **Find What** input box (the button marked `>`).

- **Any Character**
- **Character in Range**
- **Character Not in Range**
- **Beginning of Word**
- **End of Word**
- **Beginning of Line**
- **End of Line**
- **Tagged Expression**
- **0 or More Matches**
- **1 or More Matches**

Clicking on the required expression description inserts the corresponding expression syntax in the input field. Given below is a list of regular expression syntax characters.

<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>
<tr>
<td>(</td>
<td>Marks the start of a region for tagging a match.</td>
</tr>
</tbody>
</table>
Marks the end of a tagged region.

\n
Where \( n \) is 1 through 9 refers to the first through ninth tagged region when replacing. For example, if the search string was Fred((1-9))XXX and the replace string was Sam \( \1YYY \), when applied to Fred2XXX this would generate Sam2YYY.

\<

Matches the start of a word.

\>

Matches the end of a word.

\x

Allows you to use a character x, which would otherwise have a special meaning. For example, \[ \] would be interpreted as [ ] and not as the start of a character set.

[...]

Indicates a set of characters. For example, \[abc\] means any of the characters a, b or c. You can also use ranges: for example \[a-z\] for any lower case character.

[^...]

The complement of the characters in the set. For example, \[^A-Za-z\] means 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. Example: A+$ to find one or more A's at end of line.

* Matches 0 or more times. For example, Sa*m matches Sm, Sam, Saam, Saaam and so on.

+ Matches 1 or more times. For example, Sa+m matches Sam, Saam, Saaam and so on.

**Note:** Regular expressions are not supported in the Replace field.

Find Next

The Find Next command repeats the last Find command. It searches for the next occurrence of the input text.

23.2.12 Replace

Icons and shortcuts

<table>
<thead>
<tr>
<th>Command</th>
<th>Icon</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replace</td>
<td>![Replace Icon]</td>
<td>Ctrl+H</td>
</tr>
</tbody>
</table>

**Description**

The Replace command enables you to find and replace one text string with another. It features the same options as the Find command. Depending on the view you are using, the Replace dialog displays different find options. You can replace each item individually, or you can use the Replace All button to perform a global find-and-replace operation.

**Grid View**

In Grid View, selecting the Replace command opens the Replace dialog (screenshot below). The screenshot shows the various Find options, which are described in the description of the Find command.
Text View

For a description of the Replace function of editable Text Views, see the section Text View. The Replace function of non-editable Text Views (such as in the XPath/XQuery window) is described below. In Text View, selecting the Replace command opens the Find & Replace dialog (screenshot below). The Find options are the same as for the Find command. The Replace in selection only option carries out the find-and-replace operation only within the text selection.

Note: When using the Replace All command, each replacement is recorded as a single operation, so Replace All can be undone step-by-step.

23.2.13 Find in Files

The Find in Files command is a powerful way to find and replace text quickly among a large number of files. Clicking the command pops up the Find in Files dialog (screenshot below). The Find in Files command is different from the Find command in that it searches all the specified locations for the Find string at once and executes replace actions immediately. A report is then
displayed in the **Find in Files output window.** In the case of the **Find** command, however, the user enters the search string and goes through the (single) active document one found item at a time.

### Find criteria

There are two broad find criteria: (i) what to find, and (ii) where to look? For a description of how to set the text that is to be searched (what to find), see the description of the **Find** command. If the text entered in the Find What text box is a regular expression, then the Regular Expression check box must be checked. An entry helper for regular expressions can be accessed by clicking the **Unfold** button to the right of the **Find What** input box (the button marked >) button. The use of regular expressions for searching is explained in the section about the **Find** command.

To specify what node types and parts of an XML document should be searched, check the **Advanced XML Search** check box and then select the required node types.

You can specify what files should be searched by checking either the **Only in Current File** check box or the **Search on Disk** check box. If you choose to search on disk, you can select a folder or a **project** to search (after checking the **Search On Disk** check box). When a project folder is selected, external folders added to the project can be skipped. The files to be searched can be filtered by file extension and a star (xml* or xsl*, for example). The separator between two file extensions can be a comma or a semi-colon (xml*;xsl*, for example). The star character can
also be used as a wildcard.

The instances of the Find string at all the search locations are listed in the Find in Files output bar. Clicking on one of the listed items opens that file in Text View and highlights the item.

Replace
The most important thing to note is that clicking the Replace button replaces all the instances of the Find string with the Replace string. If Open Files On Replacing was checked in the Find in Files dialog, then the file will be opened in Text View; otherwise the replacement is done silently. All the replaced strings are listed in the Find in Files output bar. Clicking on one of the listed items opens that file in Text View and highlights the item.

Note: Regular expressions are not supported in the Replace field.

23.2.14 Bookmark Commands

Icons and shortcuts

<table>
<thead>
<tr>
<th>Command</th>
<th>Icon</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert/Remove Bookmark</td>
<td></td>
<td>Ctrl+F2</td>
</tr>
<tr>
<td>Remove All Bookmarks</td>
<td></td>
<td>Ctrl+Shift+F2</td>
</tr>
<tr>
<td>Goto Next Bookmark</td>
<td></td>
<td>F2</td>
</tr>
<tr>
<td>Goto Previous Bookmark</td>
<td></td>
<td>Shift+F2</td>
</tr>
</tbody>
</table>

Insert/Remove Bookmark
The Insert/Remove Bookmark command inserts a bookmark at the current cursor position, or removes the bookmark if the cursor is in a line that has been bookmarked previously. This command is only available in Text View.

Bookmarked lines are displayed in one of the following ways:

- If the bookmarks margin has been enabled, then a solid blue ellipse appears to the left of the text in the bookmark margin.
- If the bookmarks margin has not been enabled, then the entire line containing the cursor is highlighted.

The F2 key cycles through all the bookmarks in the document.

Remove All Bookmarks
The Remove All Bookmarks command removes all the currently defined bookmarks. This command is only available in Text View. Note that the Undo command does not undo the effects of Remove All Bookmarks.
Goto Next Bookmark

The **Goto Next Bookmark** command places the text cursor at the beginning of the next bookmarked line. This command is only available in Text View.

Goto Previous Bookmark

The **Goto Previous Bookmark** command places the text cursor at the beginning of the previous bookmarked line. This command is only available in Text View.

23.2.15 **Comment In/Out**

The **Comment In/Out** command is available in Text View and is used to comment and uncomment XML text fragments. Text in an XML document can be commented out using the XML start-comment and end-comment delimiters, respectively `<!--` and `-->`. In XMLSpy, these comment delimiters can be inserted around a text selection by using the **Comment In/Out** menu command.

To comment out a block of text, select the text to be commented out and then select the command **Comment In/Out**, either from the **Edit** menu or the context menu you get on right-clicking the selected text. The commented text will be grayed out (see screenshot below).

```
<Department>
    <Name>Administration</Name>
    <Person>
    <Person>
    <Person>
        <!--<Person> -->
        <First>
        <Last>/Last>
        <PhoneExt>/PhoneExt>
        <EMail>/EMail>
        <LeaveTotal>/LeaveTotal>
        <LeaveUsed>/LeaveUsed>
        <LeaveLeTt>/LeaveLeTt>
    </Person>
</Department>
```

To uncomment a commented block of text, place the cursor in the commented block and select the command **Comment In/Out**, either from the **Edit** menu or the context menu you get on right-clicking within the commented-out text. The comment delimiters will be removed and the text will no longer be grayed out.
23.3 **Project Menu**

XMLSpy uses the familiar tree view to manage multiple files or URLs in XML projects. Files and URLs can be grouped into folders by common extension or any arbitrary criteria, allowing for easy structuring and batch manipulation.

Please note: Most project-related commands are also available in the context menu, which appears when you right-click any item in the project window.

**Absolute and relative paths**

Each project is saved as a project file, and has the `.spp` extension. These files are actually XML documents that you can edit like any regular XML File. In the project file, absolute paths are used for files/folders on the same level or higher, and relative paths for files/folders in the current folder or in sub-folders. For example, if your directory structure looks like this:

```
| Folder1
|   |
|   | Folder2
|   |
|   | Folder3
```
If your .spp file is located in Folder3, then references to files in Folder1 and Folder2 will look something like this:

```
c:\Folder1\NameOfFile.ext
c:\Folder1\Folder2\NameOfFile.ext
```

References to files in Folder3 and Folder4 will look something like this:

```
.\NameOfFile.ext
.\Folder4\NameOfFile.ext
```

If you wish to ensure that all paths will be relative, save the .spp files in the root directory of your working disk.

**Drag-and-drop**

In the Project window, a folder can be dragged to another folder or to another location within the same folder. A file can be dragged to another folder, but cannot be moved within the same folder (within which files are arranged alphabetically). Additionally, files and folders can be dragged from Windows File Explorer to the Project window.

**Find in project**

You can search for project files and folders using their names or a part of their name. If the search is successful, files or folders that are located are highlighted one by one.

To start a search, select the project folder in the Project sidebar that you wish to search, then select the command **Edit | Find** (or the shortcut **Ctrl+F**) in the Find dialog that pops up (screenshot below) enter the text string you wish to search for and select or deselect the search options (explained below) according to your requirements.

The following search options are available:

- Whole-word matching is more restricted since the entire string must match an entire word in the file or folder name. In file names, the parts before and after the dot (without the dot) are each treated as a word.
- It can be specified that casing in the search string must exactly match the text string in the file or folder name.
Folder names can be included in the search. Otherwise, only file names are searched.
- **External folders** can be included or excluded from the search. External folders are actual folders on the system or network, as opposed to project folders, which are created within the project and not on the system.

If the search is successful, the first matching item is highlighted in the Project sidebar. You can then browse through all the returned matching items by clicking the **Find Next** and **Find Prev** buttons in the Find dialog.

### Refreshing projects
If a change is made to an external folder, this change will not be reflected in the Project Window till the project is refreshed.

### Global resources in the context menu
When you right-click a folder in the Project window, in the context menu that appears, you can select the **Add Global Resource** menu item to add a global resource. The menu command itself pops up the Choose Global Resource dialog, which lists all the file-type and folder-type global resources in the currently active Global Resources XML File. Select the required global resource, and it will be added to the selected project folder.

### Projects and source control providers
If you intend to add an XMLSpy project to a source control repository, please ensure that the project files position in the hierarchical file system structure is one which enables you to add files only from below it (taking the root directory to be the top of the directory tree).

In other words, the directory where the **project file** is located, essentially represents the root directory of the project within the source control repository. Files added from above it (the project root directory) will be added to the XMLSpy project, but their location in the repository may be an unexpected one—if they are allowed to be placed there at all.

For example, given the directory structure show above, if a project file is saved in **Folder3** and placed under source control:

- Files added to **Folder1** may not be placed under source control,
- Files added to **Folder2** are added to the root directory of the repository, instead of to the project folder, but are still under source control,
- Files located in **Folder3** and **Folder4** work as expected, and are placed under source control.

### 23.3.1 New Project

The **New Project** command creates a new project in XMLSpy. If you are currently working with another project, a prompt appears asking if you want to close all documents belonging to the current project.
23.3.2 Open Project

The **Open Project**... command opens an existing project in XMLSpy. If you are currently working with another project, the previous project is closed first.

23.3.3 Reload Project

The **Reload Project** command reloads the current project from disk. If you are working in a multi-user environment, it can sometimes become necessary to reload the project from disk, because other users might have made changes to the project.

**Please note:** Project files (.spp files) are actually XML documents that you can edit like any regular XML File.

23.3.4 Close Project

The **Close Project** command **closes** the active project. If the project has been modified, you will be asked whether you want to save the project first. When a project is modified in any way, an asterisk is added to the project name in the Project Window.

23.3.5 Save Project, Save Project As

The **Save Project** command **saves** the current project. You can also save a project by making the project window active and clicking the icon.

The **Save Project As** command **saves** the current project with a new name that you can enter when prompted for one.

23.3.6 Source Control

Your Altova application supports Microsoft SourceSafe and other compatible repositories. A list of supported systems is given in the section, **Supported Source Control Systems**. This section describes the commands in the **Project | Source Control** submenu, which are used to work with the source control system from within your Altova application.

**Overview of the Source Control feature**

The mechanism for placing files in an application project under source control is as follows:
1. In XMLSpy, an application project folder containing the files to be placed under source control is created. Typically, the application project folder will correspond to a local folder in which the project files are located. The path to the local folder is referred to as the local path.

2. In the source control system's database (also referred to as source control or repository), a folder is created that will contain the files to be placed under source control.

3. Application project files are added to source control using the command **Project | Source Control | Add to Source Control**.

4. Source control actions, such as checking in to, checking out from, and removing files from source control, can be carried out by using the commands in the **Project | Source Control** submenu. The commands in this submenu are listed in the sub-sections of this section.

**Note:** If you wish to change the current source control provider, this can be done in any of two ways: (i) via the Source Control options (**Tools | Options | Source Control**), or (ii) in the Change Source Control dialog (**Project | Source Control | Change Source Control**).

**Note:** Note that a source control project is not the same as an application project. Source control projects are directory-dependent, while XMLSpy projects are logical constructions without direct directory dependence.

For additional information, see the section, **Source Control**.

### Open from Source Control

The **Open from Source Control** command creates a new application project from a project under source control.

Create the new project as follows:

1. Depending on the source control system used, it might be necessary, before you create a new project from source control, to make sure that no file from the project is checked out.

2. No project need be open in the application, but can be.

3. Select the command **Project | Source Control | Open from Source Control**.

4. The source control system that is currently set will pop up its verification and connection dialogs. Make the connection to the repository you want, that is, to the bound folder in the repository that corresponds to the local folder.

5. In the dialog that pops up (**screenshot below**), browse for the local folder to which the contents of the bound folder in the repository (that you have just connected to) must be copied. In the screenshot below the bound folder is called **MyProject** and is represented by the $ sign; the local folder is **C:\M20130326**.
6. Click **OK**. The contents of the bound folder (**MyProject**) will be copied to the local folder `C:\M20130326`, and a dialog pops up asking you to select the project file (**.spp** file) that is to be created as the new project.

7. Select the **.spp** file that will have been copied to the local folder. In our example, this will be **MyProject.spp** located in the `C:\M20130326` folder. A new project named **MyProject** will be created in the application and will be displayed in the Project window. The project's files will be in the folder `C:\M20130326`.

**Source control symbols**

Files and the project folder display certain symbols, the meanings of which are given below.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Checked in" /></td>
<td>Checked in. Available for check-out.</td>
</tr>
<tr>
<td><img src="image" alt="Checked out by another user" /></td>
<td>Checked out by another user. Not available for check-out.</td>
</tr>
<tr>
<td><img src="image" alt="Checked out locally" /></td>
<td>Checked out locally. Can be edited and checked-in.</td>
</tr>
</tbody>
</table>

**Enable Source Control**

The **Enable Source Control** command allows you to enable or disable source control for an application project. Selecting this option on any file or folder, enables/disables source control for the whole project. After source control is enabled, the check in/out status of the various files are retrieved and displayed in the Project window.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Checked in" /></td>
<td>Checked in. Available for check-out.</td>
</tr>
<tr>
<td><img src="image" alt="Checked out by another user" /></td>
<td>Checked out by another user. Not available for check-out.</td>
</tr>
</tbody>
</table>
Checked out locally. Can be edited and checked-in.

Get Latest Version

The Get Latest Version command (in the Project | Source Control menu) retrieves and places the latest source control version of the selected file(s) in the working directory. The files are retrieved as read-only and are not checked out. This command works like the Get command, but does not display the Get dialog.

If the selected files are currently checked out, then the action taken will depend on how your source control system handles such a situation. Typically, the source control system will ask whether you wish to replace, merge with, or leave the checked-out file as it is.

Note: This command is recursive when performed on a folder, that is, it affects all files below the current one in the folder hierarchy.

Get, Get Folders

The Get command (in the Project | Source Control menu) retrieves files from the repository as read-only files. (To be able to edit a file, you must check it out.) The Get dialog lists the files in the object (project or folder) on which the Get command was executed (see screenshot below). You can select the files to retrieve by checking them.

Note: The Get Folders command allows you to select individual sub-folders in the repository if this is allowed by your source control system.

You can choose to overwrite changed checked-out files by checking this option at the bottom of the Get dialog. On clicking OK, the files will be overwritten. If any of the overwritten files is currently open, a dialog pops up (screenshot below) asking whether you wish to reload the file/s (Reload button), close the file/s (Close), or retain the current view of the file (Cancel).
Advanced Get Options

The Advanced Get Options dialog (screenshot below) is accessed via the Advanced button in the Get dialog (see first screenshot in this section).

Here you can set options for (i) replacing writable files that are checked out, (ii) the timestamp, and (iii) whether the read-only property of the retrieved file should be changed so that it will be writable.

Check Out, Check In

After a project file has been placed under source control, it can be checked out or checked in by selecting the file (in the Project window) and clicking the respective command in the Project | Source Control menu: Check Out and Check In.

When a file is checked out, a copy from the repository is placed in the local folder. A file that is checked out can be edited. If a file that is under source control is not checked out, it cannot be edited. After a file has been edited, the changes can be saved to the repository by checking in the file. Even if the file is not saved, checking it in will save the changes to the repository. Whether a file is checked out or not is indicated with a tick or lock symbol in its icon.

Files and the project folder display certain symbols, the meanings of which are given below.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
</table>
Selecting the project or a folder within the project, selects all files in the selected object. To select multiple objects (files and folders), press the Ctrl key while clicking the objects. The screenshot below shows a project that has been checked out. The file *QuickStart.css* has subsequently been checked in.

![Project](image)

**Saving and rejecting editing changes**

Note that, when checking in a file, you can choose to leave the file checked out. What this does is save editing changes to the repository while continuing to keep the file checked out, which is useful if you wish to periodically save editing changes to the repository and then continue editing.

If you have checked out a file and made editing changes, and then wish to reject these changes, you can revert to the document version saved in the repository by selecting the command **Project | Source Control | Undo Check Out**.

**Checking out**

The Check Out dialog (*screenshot below*) allows you: (i) to select the files to check out, and (ii) to select whether the repository version or the local version should be checked out.
Checking in

The Check In dialog (screenshot below) allows you: (i) to select the files to check in, and (ii) if you wish, to keep the file checked out.

![Check In dialog](image_url)

**Note:** In both dialogs (Check Out and Check In), multiple files appear if the selected object (project or project folder/s) contain multiple files.

**Undo Check Out**

If you have checked out a file and made editing changes, and then wish to reject these changes, you can revert to the document version saved in the repository by selecting the command Project | Source Control | Undo Check Out.

Files and the project folder display certain symbols, the meanings of which are given below.
Add to Source Control

After a project has been added to source control, you can add files either singly or in groups to source control. Select the file in the Project window and then click the command **Project | Source Control | Add to Source Control**. To select multiple files, keep the **Ctrl** key pressed while clicking on the files you wish to add. Running the command on a (green) project folder (see *screenshot below*) adds all files in the folder and its sub-folders to source control.

When files are added to source control, the local folder hierarchy is replicated in the repository (not the project folder hierarchy). So, if a file is in a sub-folder X levels deep in the local folder, then the file's parent folder and all other ancestor folders are automatically created in the repository.

When the first file from a project is added to source control, the correct bindings are created in the repository and the project file (.spp file) is added automatically. For more details, see the section **Add to Source Control**.

Source control symbols

Files and the project folder display certain symbols, the meanings of which are given below.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Symbol" /></td>
<td>Checked in. Available for check-out.</td>
</tr>
<tr>
<td><img src="image2" alt="Symbol" /></td>
<td>Checked out by another user. Not available for check-out.</td>
</tr>
<tr>
<td><img src="image3" alt="Symbol" /></td>
<td>Checked out locally. Can be edited and checked-in.</td>
</tr>
</tbody>
</table>
**Remove from Source Control**

To remove a file from source control, select the file and click the command **Project | Source Control | Remove from Source Control**. You can also remove: (i) files in a project folder by executing the command on the folder, (ii) multiple files that you select while keeping the Ctrl key pressed, and (iii) the entire project by executing the command on the project.

**Share from Source Control**

The **Share from Source Control** command is supported when the source control system being used supports shares. You can share a file, so that it is available at multiple local locations. A change made to one of these local files will be reflected in all the other "shared" versions.

In the application's Project window first select the project (*highlighted in the screenshot below*). Then click the **Share from Source Control**.

![Project window screenshot](image)

The Share To [Folder] dialog (*screenshot below*) pops up.
To select the files to share, first choose, in the project tree in the right-hand pane, the folder in which the files are. The files in the chosen folder are displayed in the left-hand pane. Select the file you wish to share (multiple files by pressing the Ctrl key and clicking the files you want to share). The selected file/s will be displayed in the Files to Share text box (at top left). Click Share and then Close to copy the selected file/s to the local share folder.

The share folder is noted in the name of the Share to [Folder] dialog. In the screenshot above it is the local folder (since the $ sign is the folder in the repository to which the local folder is bound). You can see and set the share folder in the Change Source Control dialog (screenshot below, Change Source Control) by changing the local path and server binding.
For more details about sharing using your source control system, see the source control system's user documentation.

**Show History**

The **Show History** command activates the Show History feature of the active source control system. It displays the history of the file selected in the Project window. Select the project title to display the history of the project file (.spp file). You can view information about previous versions of a file and differences, as well as retrieve previous versions of the file.

The screenshot below shows the History dialog of the Visual SourceSafe source control system. It lists the various versions of the `MyProject.spp` file.

![History Dialog Example](image)

This History dialog provides various ways of comparing and getting specific versions of the file in question. Double-clicking an entry in the list opens the History Details dialog box for that file. The buttons in the History dialog provide the following functionality:

- **Close**: Closes this dialog box.
- **View**: Opens a dialog box in which you can select the type of file viewer.
- **Details**: Opens a dialog box in which you can see the properties of the currently active file.
- **Get**: Retrieves a previous file version and places it in the working directory.
- **Check Out**: Allows you to check out a previous version of the file.
- **Diff**: Opens the **Difference options** dialog box for differencing options between two file versions. Use **Ctrl+Click** to mark two file versions in this window, then click Diff to view the differences between them.
- **Pin**: Pins or unpins a version of the file, allowing you to define the specific file version to use when differencing two files.
- **Rollback**: Rolls back to the selected version of the file.
- **Report**: Generates a history report that you can send to a printer, file, or clipboard.
- **Help**: Opens the online help of the source control provider plugin.

### Show Differences

The **Show Differences** command is enabled when a file in the Project window is selected. To select the project file (.spp file), select the project title in the Project window. The **Show Differences** command starts the source control system's differencing tool so that differences between files can be directly checked from your Altova application.

The screenshot below shows the differencing tool of the Visual SourceSafe source control system.

The repository and local versions are shown by default in the **Compare** and **To** text fields respectively. You can browse for other files as follows:

1. From the **Browse** button dropdown list, select SourceSafe projects (for browsing repository files) or Windows folders (for browsing local folders).
2. Browse for the files you want and select them.

Select the options you want and click **OK** to run the check. The differencing results are displayed in a separate window. The screenshots below show the results of a check in two formats.
The screenshot above shows the Visual SourceSafe differencing result in Visual format (see Options dialog above), while the screenshot below shows the result in Unix format. In both, there are two differences, each of which is a change of the grade from C to B.

For a detailed description of how your source control system handles differencing, see the product's user documentation.

Show Properties

The Show Properties command displays the properties of the currently selected file (screenshot below). What properties are displayed depends on the source control system you are using. The screenshot below shows properties when Visual SourceSafe is the active source control system.

Note that this command is enabled only for single files.
Refresh Status
The Refresh Status command refreshes the status of all project files independent of their current status.

Source Control Manager
The Source Control Manager command starts your source control software with its native user interface.

Change Source Control
The current binding is what the active application project will use to connect to the source control database, so the current binding must be correct. By this is meant that the application project file (.spp file) must be in the local path folder and the bound folder on the repository must be the database where this project's files are stored. Typically the bound folder and its sub-structure will correspond with the local workspace folder and its sub-structure.

In the Change Source Control dialog (screenshot below), you can change the source control system (SCC Provider), the local folder (Local Path), and the repository binding (Server Name and Server Binding).
Only after unbinding the current binding can the settings be changed. Unbind the current binding with the **Unbind** button. All the settings are now editable.

![Change Source Control Dialog](image)

Change source control settings as follows:

1. Use the **Browse** button to browse for the local folder and the **Select** button to select from among the installed source control systems.
2. After doing this you can bind the local folder to a repository database. Click the **Bind** button to do this. This pops up the connection dialog of your source control system.
3. If you have entered a **Logon ID**, this will be passed to the source control system; otherwise you might have to enter your logon details in the connection dialog.
4. Select the database in the repository that you wish to bind to this local folder. This setting might be spread over more than one dialog.
5. After the setting has been created, click **OK** in the Change Source Control dialog.

### 23.3.7 Add Files to Project

The **Project** | **Add Files to Project** command adds files to the current project. Use this command to add files to any folder in your project. You can either select a single file or any group of files (using **Ctrl+ click**) in the Open dialog box. If you are adding files to the project, they will be distributed among the respective folders based on the File Type Extensions defined in the **Project Properties** dialog box.

### 23.3.8 Add Global Resource to Project

The **Project** | **Add Global Resource to Project** command pops up the Choose Global Resource dialog, in which you can select a global resource of file or folder type to add to the project. If a file-type global resource is selected, then the file is added to the appropriate folder based on the File Type Extensions defined in the **Project Properties** dialog box. If a folder-type global resource is selected, that folder will be opened in a file-open dialog and you will be prompted to select a file;
the selected file is added to the appropriate folder based on the File Type Extensions defined in the Project Properties dialog box. For a description of global resources, see the Global Resources section in this documentation.

23.3.9 Add URL to Project

The Project | Add URL to Project command adds a URL to the current project. URLs in a project cause the target object of the URL to be included in the project. Whenever a batch operation is performed on a URL or on a folder that contains a URL object, XMLSpy retrieves the document from the URL, and performs the requested operation.

23.3.10 Add Active File to Project

The Project | Add Active File to Project command adds the active file to the current project. If you have just opened a file from your hard disk or through an URL, you can add the file to the current project using this command.

23.3.11 Add Active And Related Files to Project

The Project | Add Active and Related Files to Project command adds the currently active XML document and all related files to the project. When working on an XML document that is based on a DTD or Schema, this command adds not only the XML document but also all related files (for example, the DTD and all external parsed entities to which the DTD refers) to the current project.

Please note: Files referenced by processing instructions (such as XSLT files) are not considered to be related files.

23.3.12 Add Project Folder to Project

The Project | Add Project Folder to Project command adds a new folder to the current project. Use this command to add a new folder to the current project or a sub-folder to a project folder. You can also access this command from the context-menu when you right-click on a folder in the project window.
Note: A project folder can be dragged and dropped into another project folder or to any other location in the project. Also, a folder can be dragged from Windows (File) Explorer and dropped into any project folder.

Note: Project folders are green, while external folders are yellow.

23.3.13 Add External Folder to Project

The Project | Add External Folder to Project command adds a new external folder to the current project. Use this command to add a local or network folder to the current project. You can also access this command from the context-menu when you right-click a folder in the project window.

Note: External folders are yellow, while project folders are green.

Note: Files contained in external folders cannot be placed under source control.

Adding external folders to projects

To add an external folder to the project:

1. Select the menu option Project | Add External Folder to Project.
2. Select the folder you want to include from the Browse for Folder dialog box, and click OK to confirm.

The selected folder now appears in the project window.

3. Click the plus icon to view the folder contents.
Filtering contents of folders

To filter the contents of the folder:

1. Right-click the local folder, and select the popup menu option **Properties**. This opens the Properties dialog box.

2. Click in the **File extensions** field and enter the file extensions of the file types you want to see. You can separate each file type with a **semicolon** to define multiple types (XML and Schema XSDs in this example).

3. Click **OK** to confirm.
The Project window now only shows the XML and XSD files of the tutorial folder.

**Validating external folders**
To validate and check an external folder for well-formedness:

1. Select the file types you want to see or check from the external folder,

2. Click the folder and click the Check well-formedness or Validate icon (hotkeys F7 or F8). All the files visible under the folder are checked. If a file is malformed or invalid, then this file is opened in the Main Window, allowing you to edit it.

3. Correct the error and run the validation process once more to recheck.

**Updating a project folder**
You might add or delete files in the local or network directory at any time. To update the folder view, right-click the external folder, and select the popup menu option Refresh external folder.

**Deleting external folders and files in them**
Select an external folder and press the Delete key to delete the folder from the Project window. Alternatively, right-click the external folder and select the Delete command. Each of these actions only deletes the external folder from the Project window. The external folder is not deleted from the hard disk or network.

To delete a file in an external folder, you have to delete it physically from the hard disk or network. To see the change in the project, refresh the external folder contents (right-click the external folder and select Refresh).

**Note:** An external folder can be dragged and dropped into a project folder or to any other location in the project (but not into another external folder). Also, an external folder can be dragged from Windows (File) Explorer and dropped into any location in the project window except into another external folder.

### 23.3.14 Add External Web Folder to Project
This command adds a new external web folder to the current project. You can also access this command from the context-menu when you right-click a folder in the project window. Note that files contained in external folders cannot be placed under source control.

**Adding an external web folder to the project**
To add an external web folder to the project, do the following:

1. Select the menu option Project | Add External Web Folder to Project. This opens the Add Web Folder to Project dialog box (screenshot below).
2. Click in the Server URL field and enter the URL of the server URL. If the server is a Microsoft® SharePoint® Server, check this option. See the Folders on a Microsoft® SharePoint® Server section below for further information about working with files on this type of server.

3. If the server is password-protected, enter your User ID and password in the User and Password fields.

4. Click Browse to connect to the server and view the available folders.

5. Click the folder you want to add to the project view. The Open button only becomes active once you do this. The URL of the folder now appears in the File URL field.

6. Click Open to add the folder to the project.
7. Click the plus icon to view the folder contents.

Filtering folder contents
To filter the contents of a folder, right-click the folder and select Properties from the context menu. In the Properties dialog that pops up, click in the File Extensions field and enter the file extensions of the file types you want to see (for example, XML and XSD files). Separate each file type with a semicolon (for example: xml; xsd; sps). The Project window will now show that folder only with files having the specified extension.

Validating and checking a folder for well-formedness
To check the files in a folder for well-formedness or to validate them, select the folder and then click the Check well-formedness or Validate icon (hotkeys F7 or F8, respectively). All the files that are visible in the folder are checked. If a file is malformed or invalid, then this file is opened in the main window, allowing you to edit it. Correct the error and restart the process to recheck the rest of the folder. Note that you can select discontinuous files in the folder by holding Ctrl and clicking the files singly. Only these files are then checked when you press F7 or F8.

Updating the contents of the project folder
Files may be added or deleted from the web folder at any time. To update the folder view, right-click the external folder and select the context menu option Refresh.
Deleting folders and files
Since it is the Web folder that has been added to the project, it is only the Web folder (and not files within it) that can be deleted from the project. You can delete a Web folder from a project, by either (i) right-clicking the folder and selecting **Delete**, or (ii) selecting the folder and pressing the **Delete** key. This only deletes the folder from the Project view; it does not delete anything on the web server.

**Note:** Right-clicking a single file and pressing the **Delete** key does not delete a file from the Project window. You have to delete it physically on the server and then refresh the contents of the external folder.

Folders on a Microsoft® SharePoint® Server
When a folder on a Microsoft® SharePoint® Server has been added to a project, files in the folder can be checked out and checked in via commands in the context menu of the file listing in the Project window (*see screenshot below*). To access these commands, right-click the file you wish to work with and select the command you want (**Check Out**, **Check In**, **Undo Check Out**).

The User ID and password can be saved in the properties of individual folders in the project, thereby enabling you to skip the verification process each time the server is accessed.

In the Project window (*screenshot below*), file icons have symbols that indicate the check-in/check-out status of files. The various file icons are shown below:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Icon]</td>
<td>Checked in. Available for check-out.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Checked out by another user. Not available for check-out.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Checked out locally. Can be edited and checked-in.</td>
</tr>
</tbody>
</table>

The following points should be noted:

- After you check out a file, you can edit it in your Altova application and save it using **File | Save (Ctrl+S)**.
- You can check-in the edited file via the context menu in the Project window (*see screenshot above*), or via the context menu that pops up when you right-click the file tab.
in the Main Window of your application (screenshot below).

- When a file is checked out by another user, it is not available for check out.
- When a file is checked out locally by you, you can undo the check-out with the Undo Check-Out command in the context menu. This has the effect of returning the file unchanged to the server.
- If you check out a file in one Altova application, you cannot check it out in another Altova application. The file is considered to be already checked out to you. The available commands at this point in any Altova application supporting Microsoft® SharePoint® Server will be: Check In and Undo Check Out.

### 23.3.15 Script Settings

A scripting project is assigned to an XMLSpy project as follows:

1. In the XMLSpy GUI, open the required application project.
2. Select the menu command **Project | Script Settings**. The Scripting dialog (screenshot below) opens.

   ![Scripting Dialog](image)

3. Check the **Activate Project Scripts** check box and select the required scripting project (.asprj file). If you wish to run Auto-Macros when the XMLSpy project is loaded, check the **Run Auto-Macros when XML Spy project is loaded** check box.
4. Click **OK** to finish.

   **Note:** To deactivate (that is, unassign) the scripting project of an XMLSpy project, uncheck the **Activate Project Scripts** check box.
23.3.16 Properties

The Project > Project Properties command opens the Properties dialog (screenshot below) of the active project. If you right-click a folder in the Project window (as opposed to the project folder itself) and select Properties, the Properties dialog of that folder is opened. The dialog settings are described below.

**Note:** If your project file is under source control, a prompt appears asking if you want to check out the project (.spp) file. Click OK if you want to edit settings and be able to save them.

![Properties dialog screenshot]

### Settings

**File extensions**
The *File extensions* setting is enabled for individual folders, and not for the project folder. When a file is added to a project, it will be added to the folder on which its file extension has been defined. For example, say a file named *MyReport.xml* is added to the project. If *xml* file extensions have been set on the *Invoices-EU* folder (as shown in the screenshot above), then *MyReport.xml* will be added to the *Invoices-EU* folder. If there is more than one folder to which you wish to add XML
files, then you should add individual XML files directly to the folder (instead of to the project).

**User ID and password for external folders**
On external folders (including external Web folders), you can save the user ID and password that might be required for accessing the server.

**Validation**
The DTD, XML Schema, or JSON schema that should be used to validate the files in the current folder (or entire project if the properties are those of the project).

**XSL transformation of XML files**
The XSLT stylesheet to be used for XSLT transformation of XML files in the folder.

**XSL-FO transformation of XML files**
The XSLT stylesheet to transform XML files in the folder to XSL-FO.

**XSL transformation of XSL files**
The XML file to be used for XSLT transformation with XSLT files in the folder.

**XQuery transformation of XML files**
The XQuery file to be used for XQuery executions of XML files in the folder.

**Destination files of XSL transformation**
The destination directory of XSLT transformations, and, optionally, the file extension of the result document.

**Authentic View**
The Use config specifies the StyleVision Power Stylesheet (SPS file) to use for the Authentic View display of XML files in the folder. Note that the XML file must be valid against the same schema used for the SPS.

**Notes about project properties**
Note the following points about precedence:

- When validations or XSLT/XQuery transformations are carried out via project folder context menus, then the validation or transformation files specified in this dialog take precedence over any assignment in the XML file. Also, settings specified for individual project folders take precedence over settings specified for ancestor folders.
- If one file is present in multiple folders of the project and has been assigned different validation or transformation files in the different folders, then you can set which assignment to use when the file is processed outside the project. Specify this as follows: Locate the file in the project folder whose assignment/s you wish to use. Right-click the file in that project folder, and select Properties. In the dialog that appears (screenshot below), select Use settings in current folder as default. (The current folder is the project
folder in which the file is located.) If the option is disabled, it means that the settings of the current folder are already selected as the default settings to use. If you select a file instance that is in a project folder that is not the default, then the option is enabled, and you can switch the default settings to be this folder’s settings. Note that, if the file has a local assignment (that is, an assignment within the file itself), then the local assignment will be used, and the default folder settings will be ignored.

![Properties dialog box](image)

### 23.3.17 Most Recently Used Projects

This command displays the file name and path for the nine most recently used projects, allowing quick access to these files.

Also note, that XMLSpy can automatically open the last project that you used, whenever you start XMLSpy. (Tools | Options | File section, Project | Open last project on program start).
23.4 XML Menu

The XML menu contains commands commonly used when working with XML documents. You will find commands to insert or append elements, modify the element hierarchy, set a namespace prefix, as well as to evaluate XPaths in the context of individual XML documents.

Among the most frequently used XML tasks are checks for the well-formedness of documents and validity of XML documents. Commands for these tasks are in this menu.

23.4.1 Insert

The XML | Insert command, though enabled in all views, can be used in Grid View only. It has a submenu (see screenshot) with which you can insert:

- The XML declaration and node types (Attribute, Element, Text, CDATA, Comment, Processing Instruction) in XML documents;
- DOCTYPE declarations and external DTD declarations in XML documents;
- DTD declarations (ELEMENT, ATTLIST, ENTITY, and NOTATION) in DTD documents and internal DTD declarations of XML documents.
The XML | Insert | Attribute command is available in Grid View only, and inserts a new attribute before the selected item. An inserted attribute may appear a few lines before the current item in Grid View. This is because attributes immediately follow their parent element in Grid View and precede all child elements of that parent element.

The XML | Insert | Element command is available in Grid View only, and inserts a new element before the selected item. If the current selection is an attribute, the new element is before the first child element of the attribute's parent element.
Insert Text

![Insert Text Icon] Ctrl+Shift+T

The XML | Insert | Text command is available in Grid View only, and inserts a new text row before the selected item. If the current selection is an attribute, the text row is inserted after the attribute and before the first child element of the attribute's parent element.

Insert CDATA

![Insert CDATA Icon] Ctrl+Shift+D

The XML | Insert | Cdata command is available in Grid View only, and inserts a new CDATA block before the selected item. If the current selection is an attribute, the CDATA block is inserted after the attribute and before the first child element of the attribute's parent element.

Insert Comment

![Insert Comment Icon] Ctrl+Shift+M

The XML | Insert | Comment command is available in Grid View only, and inserts a new comment before the selected item. If the current selection is an attribute, the new comment row is inserted after the attribute and before the first child element of the attribute's parent element.

Insert XML

![Insert XML Icon]

The XML | Insert | XML command is available in Grid View only, and inserts a row for the XML declaration before the selected item. You must insert the child attributes of the XML declaration and the values of this attribute. An XML declaration must look something like this:

```xml
<?xml version="1.0" encoding="UTF-8"?>
```

**Please note:** Since an XML document may only contain one XML declaration at the very top of the file, this command should only be used with the topmost row selected and if an XML declaration does not already exist.
Insert Processing Instruction

The XML | Insert | Processing Instruction command is available in Grid View only, and inserts a new processing instruction (PI) before the selected item. If the current selection is an attribute, the PI is inserted after the attribute and before the first child element of the attribute's parent element.

Insert XInclude

The XML | Insert | XInclude command is available in Grid View only, and enables you to insert a new XInclude element before the selected item. If the current selection is an attribute, the XInclude element is inserted after the attribute and before the first child element of the attribute's parent element. Selecting this command pops up the XInclude dialog (screenshot below).

The XML file to be included is entered in the href text box (alternatively, you can browse for the file by clicking the Browse (...) button to the right of the text box). The filename will be entered in the XML document as the value of the href attribute. The parse, xpointer, and encoding attributes of the XInclude element (xi:include), and the fallback child element of xi:include can also be inserted via the dialog. Do this by first checking the appropriate check box and then selecting/entering the required values. In the case of the fallback element, checking its check box only inserts the empty element. The content of the fallback element must be added.
subsequently in one of the editing views.

The `parse` attribute determines whether the included document is to be parsed as XML or text. (XML is the default value and therefore need not be specified.) The `xpointer` attribute identifies a specific fragment of the document located with the `href` attribute; it is this fragment that will be included. The `encoding` attribute specifies the encoding of the included document so that XMLSpy can transcode this document (or the part of it to be included) into the encoding of the including document. The contents of the `fallback` child element replace the `xi:include` element if the document to be included cannot be located.

Here is an example of an XML document that uses XInclude to include two XML documents:

```xml
<?xml version="1.0" encoding="UTF-16"?>
<AddressBook xsi:schemaLocation="http://www.altova.com/sv/myaddresses AddressBook.xsd"
    xmlns="http://www.altova.com/stylevision/tutorials/myaddresses"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xi="http://www.w3.org/2001/XInclude">
    <xi:include href="BusinessAddressBook.xml"/>
    <xi:include href="PersonalAddressBook.xml"/>
</AddressBook>
```

When this XML document is parsed, it will replace the two XInclude elements with the files specified in the respective `href` attributes.

xml:base

When the XML validator of XMLSpy reads an XML document and encounters the `include` element in the XInclude namespace (hereafter `xi:include`), it replaces this `element (xi:include)` with the XML document named in the `href` attribute of the `xi:include` element. The document element (root element) of the included XML document (or the element identified by an XPointer) will be included with an attribute of `xml:base` in order to preserve the base URIs of the included element. If the resulting XML document (containing the included XML document/s or tree fragment/s) must be valid according to a schema, then the document element of the included document (or the top-level element of the tree fragment) must be created with a content model that allows an attribute of `xml:base`. If, according to the schema, the `xml:base` attribute is not allowed on this element, then the resulting document will be invalid. How to define an `xml:base` attribute in an element's content model using XMLSpy's Schema View is described in the `xml:Prefixed Attributes` section of the Schema View section of the documentation.

XPointers

XMLSpy supports XPointers in XInclude. The relevant W3C recommendations are the XPointer Framework and XPointer element() Scheme recommendations. The use of an XPointer in an XInclude element enables a specific part of the XML document to be included, instead of the entire XML document. XPointers are used within an XInclude element as follows:

```xml
<xi:include href="PersonalAddressBook.xml" xpointer="element(usa)/"/>
<xi:include href="BusinessAddressBook.xml" xpointer="element(/1/1)/"/>
<xi:include href="BobsAddressBook.xml" xpointer="element(usa/3/1)/"/>
<xi:include href="PatsAddressBook.xml" xpointer="element(usa)
element(/1/1)/"/>
```
In the element() scheme of XPointer, an NCName or a child sequence directed by integers may be used.

- In the first `xi:include` element listed above, the `xpointer` attribute uses the element scheme with an NCName of `usa`. According to the XPointer Framework, this NCName identifies the element that has an ID of `usa`.
- In the second `xi:include` listed above, the `xpointer` attribute with a value of `element(/1/1)` identifies, in the first step, the first child element of the document root (which, if the document is well-formed, will be its document (or root) element). In the second step, the first child element of the element located in the previous step is located; in our example, this would be the first child element of the document element.
- The `xpointer` attribute of the third `xi:include` listed above uses a combination of NCName and child sequence. This XPointer locates the first child element of the third child element of the element having an ID of `usa`.
- If you are not sure whether your first XPointer will work, you can back it up with a second one as shown in the fourth `xi:include` listed above: `xpointer="element(usa) element(/1/1)"`. Here, if there is no element with an ID of `usa`, the back-up XPointer specifies that the first child element of the document element is to be selected. Additional backups are also allowed. Individual XPointers may not be separated, or they may be separated by whitespace: for example, `xpointer="element(usa) element(addresses/1) element(/1/1)"`.

**Note:** The namespace binding context is not used in the element() scheme because the element() scheme does not support qualified names.

### Insert DOCTYPE

The XML | Insert | DOCTYPE command is available in the Grid View of an XML file when a top-level node is selected. It appends a DOCTYPE declaration at the top of the XML document. You must enter the name of the DOCTYPE, and this name must be the same as the name of the document element.

After you have entered the name of the DOCTYPE, you can enter the declarations you wish to use in the internal DTD subset.

**Please note:**

- A DOCTYPE declaration may only appear between the XML declaration and the XML document element.
- You could use the Assign DTD command instead to create a DOCTYPE statement that refers to an external DTD document.
**Insert ExternalID**

A DOCTYPE declaration in an XML file can contain a reference to an external resource containing DTD declarations. This resource is referenced either through a public or system identifier. For example:

```
<!DOCTYPE doc_element_name PUBLIC "publicID" "systemID">
<!DOCTYPE doc_element_name SYSTEM "systemID">
```

A system identifier is a URI that identifies the external resource. A public identifier is location-independent and can be used to dereference the location of an external resource. For example, in your XMLSpy installation, URIs for popular DTDs and XML Schemas are listed in the catalog files named `catalog.xml` in the various schema folders in `C:\Program Files\Altova\Common2019\Schemas\`. A public identifier in an XML document can be used to dereference a DTD listed in these catalog files.

The XML | Insert | ExternalID command is available when a "child" item of the DOCTYPE declaration in an XML file is selected in Grid View. This command inserts a Grid View row for an external identifier (PUBLIC or SYSTEM). You must enter the type of identifier and its value.

![DOCTYPE OrgChart]

```
<!DOCTYPE OrgChart SYSTEM "orgchart.dtd" [ 
  <!ELEMENT name (#PCDATA)>
]
```

Please note: A row for External-ID can be added as a child when the DOCTYPE item is selected, or it can be inserted or appended when one of the child items of the DOCTYPE item is selected, for example, the ELEMENT declaration `name` in the example above.

**Insert ELEMENT**

The XML | Insert | ELEMENT command is available in Grid View only, for DTD documents or when an item in the DOCTYPE declaration of an XML document is selected. It inserts an ELEMENT declaration before the selected declaration.
Insert ATTLIST

The XML | Insert | ATTLIST command is available in Grid View only, for DTD documents or when an item in the DOCTYPE declaration of an XML document is selected. It inserts an ATTLIST declaration before the selected declaration.

Insert ENTITY

The XML | Insert | ENTITY command is available in Grid View only, for DTD documents or when an item in the DOCTYPE declaration of an XML document is selected. It inserts an ENTITY declaration before the selected declaration.

Insert NOTATION

The XML | Insert | NOTATION command is available in Grid View only, for DTD documents or when an item in the DOCTYPE declaration of an XML document is selected. It inserts a NOTATION declaration before the selected declaration.

Insert Encoded External File

The XML | Insert | Encoded External File command is available in Grid View only. It inserts a binary encoded file, such as an image file, as encoded characters. The encoded external file is inserted before the Grid View selection.

On clicking the command, the Insert Encoded External File dialog (screenshot below) pops up. In it you enter the path to the file, select the encoding you want, and specify whether the encoded file is to be inserted in an element or not.

You can browse for or enter the name of the external file to be encoded and embedded. Either a
Base-16 or Base-64 encoding must be specified. If you wish to enclose the encoded text in an element, then check the Create Element check box and specify the name of the desired element in the Create Element text box. If the Create Element check box is not checked, then the encoded text will be inserted directly at the cursor location.

On clicking OK, the encoded text of the selected file is inserted at the cursor location, with an enclosing element if this has been specified.

The encoded file is inserted in Grid View (the highlighted element in the screenshot below).

In Text View, the file will be inserted as below.

<img ext="jpg" encoding="xs:base64Binary">  iVBORw0KGgoAAAANSUhEUgAAABAAAAAQAQMAAAAlPW0iAAAABlBMVEUAAAD/  //+l2Z/dAAAAM0lEQVR4nGP4/5/h/1+G/58ZDrAz3D/McH8yw83NDDeNGe4U  g9C9zwz3gVLMDA/A6P9/AFGGFyjOXZtQAAAAAElFTkSuQmCC</img>

The listing above shows the encoded text of a JPG image file. An img element was created around the encoded text.

### 23.4.2 Append

The XML | Append command, though enabled in all views, can be used in Grid View only. It opens a submenu (see screenshot) with which you can append:

- The XML declaration and node types (Attribute, Element, Text, CDATA, Comment, Processing Instruction) in XML documents;
- DOCTYPE declarations and external DTD declarations in XML documents
- DTD declarations (ELEMENT, ATTLIST, ENTITY, and NOTATION) in DTD documents and internal DTD declarations of XML documents.
Append Attribute

Ctrl+I

The XML | Append | Attribute command is available in Grid View only, and appends a new attribute.

Append Element

Ctrl+E

The XML | Append | Element command is available in Grid View only, and appends an element node after the last sibling element of the selected element. If an attribute node is selected, then the element node is appended after the last child of the selected attribute's parent element.
Append Text

Ctrl+T

The XML | Append | Text command is available in Grid View only, and appends a text block after the last sibling element of the selected element. If an attribute node is selected, then the text block is appended after the last child of the selected attribute's parent element.

Append CDATA

Ctrl+D

The XML | Append | Cdata command is available in Grid View only, and appends a CDATA node after the last sibling of any selected node other than an attribute node. If an attribute node is selected, then the CDATA section is appended after the last child of the selected attribute's parent element.

Append Comment

Ctrl+M

The XML | Append | Comment command is available in Grid View only, and appends a comment node after the last sibling of any selected node other than an attribute node. If an attribute node is selected, then the comment node is appended after the last child of the selected attribute's parent element.

Append XML

The XML | Append | XML command inserts a new XML declaration `<?xml version="1.0" encoding="UTF-8"?>` as the first item in a document.

Please note: An XML document may contain only one XML declaration, which must appear at the very top of the file.
Append Processing Instruction

The **XML | Append | Processing Instruction** command is available in Grid View only, and appends a processing instruction node after the last sibling of any selected node other than an attribute node. If an attribute node is selected, then the processing instruction node is appended after the last child of the selected attribute's parent element.

Append XInclude

The **XML | Append | XInclude** command is available in Grid View only, and enables you to append an XInclude element after the last sibling of any selected node other than an attribute node. If the current selection is an attribute, the XInclude element is appended after the last child of the selected attribute's parent element. Selecting this command pops up the XInclude dialog (screenshot below).

The XML file to be included is entered in the **href** text box (alternatively, you can browse for the file by clicking the **Browse (...)** button to the right of the text box). The filename will be entered in the XML document as the value of the **href** attribute. The **parse**, **xpointer**, and **encoding** attributes of the **XInclude** element (**xi:include**), and the **fallback** child element of **xi:include** can also be inserted via the dialog. Do this by first checking the appropriate check box and then selecting/entering the required values. In the case of the **fallback** element, checking its check
box only inserts the empty element. The content of the fallback element must be added subsequently in one of the editing views.

The parse attribute determines whether the included document is to be parsed as XML or text. (XML is the default value and therefore need not be specified.) The xpointer attribute identifies a specific fragment of the document located with the href attribute; it is this fragment that will be included. The encoding attribute specifies the encoding of the included document so that XMLSpy can transcode this document (or the part of it to be included) into the encoding of the including document. The contents of the fallback child element replace the xi:include element if the document to be included cannot be located.

Here is an example of an XML document that uses XInclude to include two XML documents:

```xml
<?xml version="1.0" encoding="UTF-16"?>
<AddressBook xsi:schemaLocation="http://www.altova.com/sv/myaddresses
  AddressBook.xsd"
  xmlns="http://www.altova.com/stylevision/tutorials/myaddresses"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xi="http://www.w3.org/2001/XInclude">
  <xi:include href="BusinessAddressBook.xml"/>
  <xi:include href="PersonalAddressBook.xml"/>
</AddressBook>
```

When this XML document is parsed, it will replace the two XInclude elements with the files specified in the respective href attributes.

**xml:base**

When the XML validator of XMLSpy reads an XML document and encounters the include element in the XInclude namespace (hereafter xi:include), it replaces this element (xi:include) with the XML document named in the href attribute of the xi:include element. The document element (root element) of the included XML document (or the element identified by an XPointer) will be included with an attribute of xml:base in order to preserve the base URIs of the included element. If the resulting XML document (containing the included XML document/s or tree fragment/s) must be valid according to a schema, then the document element of the included document (or the top-level element of the tree fragment) must be created with a content model that allows an attribute of xml:base. If, according to the schema, the xml:base attribute is not allowed on this element, then the resulting document will be invalid. How to define an xml:base attribute in an element's content model using XMLSpy's Schema View is described in the xml:Prefixed Attributes section of the Schema View section of the documentation.

**XPointers**

XMLSpy supports XPointers in XInclude. The relevant W3C recommendations are the XPointer Framework and XPointer element() Scheme recommendations. The use of an XPointer in an XInclude element enables a specific part of the XML document to be included, instead of the entire XML document. XPointers are used within an XInclude element as follows:

```xml
<xi:include href="PersonalAddressBook.xml" xpointer="element(usa)"/>
<xi:include href="BusinessAddressBook.xml" xpointer="element(/1/1)"/>
<xi:include href="BobsAddressBook.xml" xpointer="element(usa/3/1)"/>
<xi:include href="PatsAddressBook.xml" xpointer="element(usa)
  element(/1/1)"/>
```
In the element() scheme of XPointer, an NCName or a child sequence directed by integers may be used.

- In the first \texttt{xi:include} element listed above, the \texttt{xpointer} attribute uses the element scheme with an NCName of \texttt{usa}. According to the XPointer Framework, this NCName identifies the element that has an ID of \texttt{usa}.
- In the second \texttt{xi:include} listed above, the \texttt{xpointer} attribute with a value of \texttt{element(/1/1)} identifies, in the first step, the first child element of the document root (which, if the document is well-formed, will be its document (or root) element). In the second step, the first child element of the element located in the previous step is located; in our example, this would be the first child element of the document element.
- The \texttt{xpointer} attribute of the third \texttt{xi:include} listed above uses a combination of NCName and child sequence. This XPointer locates the first child element of the third child element of the element having an ID of \texttt{usa}.
- If you are not sure whether your first XPointer will work, you can back it up with a second one as shown in the fourth \texttt{xi:include} listed above: \texttt{xpointer="element(usa) element(/1/1)"}. Here, if there is no element with an ID of \texttt{usa}, the back-up XPointer specifies that the first child element of the document element is to be selected. Additional backups are also allowed. Individual XPointers may not be separated, or they may be separated by whitespace: for example, \texttt{xpointer="element(usa) element(addresses/1) element(/1/1)"}.

\textbf{Note:} The namespace binding context is not used in the element() scheme because the element() scheme does not support qualified names.

\section{Append DOCTYPE}

The \texttt{XML \ Append \ DOCTYPE} command is available in the Grid View of an XML file when a top-level node is selected. It appends a DOCTYPE declaration at the top of the XML document. You must enter the name of the DOCTYPE, and this name must be the same as the name of the document element.

After you have entered the name of the DOCTYPE, you can enter the declarations you wish to use in the internal DTD subset.

\textbf{Please note:}

- A DOCTYPE declaration may only appear between the XML declaration and the XML document element.
- You could use the \texttt{Assign DTD} command instead to create a DOCTYPE statement that refers to an external DTD document.
Append ExternalID

A DOCTYPE declaration in an XML file can contain a reference to an external resource containing DTD declarations. This resource is referenced either through a public or system identifier. For example:

```
<!DOCTYPE doc_element_name PUBLIC "publicID" "systemID">
<!DOCTYPE doc_element_name SYSTEM "systemID">
```

A system identifier is a URI that identifies the external resource. A public identifier is location-independent and can be used to dereference the location of an external resource. For example, in your XMLSpy installation, URIs for popular DTDs and XML Schemas are listed in the catalog files named `catalog.xml` in the various schema folders in \C:\Program Files\Altova\Common2019\Schemas\`. A public identifier in an XML document can be used to dereference a DTD listed in these catalog files.

The XML | Append | ExternalID command is available when a "child" item of the DOCTYPE declaration in an XML file is selected in Grid View. This command inserts a Grid View row for an external identifier (PUBLIC or SYSTEM). You must enter the type of identifier and its value.

```
<!DOCTYPE OrgChart SYSTEM "orgchart.dtd" [
  <!ELEMENT name (#PCDATA)>
]>
```

Please note: A row for External-ID can be added as a child when the DOCTYPE item is selected, or it can be inserted or appended when one of the child items of the DOCTYPE item is selected, for example, the ELEMENT declaration `name` in the example above.

Append ELEMENT

The XML | Append | ELEMENT command is available in Grid View only, for DTD documents or when an item in the DOCTYPE declaration of an XML document is selected. It appends an ELEMENT declaration to the list of declarations.
Append ATTLIST

The XML | Append | ATTLIST command is available in Grid View only, for DTD documents or when an item in the DOCTYPE declaration of an XML document is selected. It appends an ATTLIST declaration to the list of declarations.

Append ENTITY

The XML | Append | ENTITY command is available in Grid View only, for DTD documents or when an item in the DOCTYPE declaration of an XML document is selected. It appends an ENTITY declaration to the list of declarations.

Append NOTATION

The XML | Append | NOTATION command is available in Grid View only, for DTD documents or when an item in the DOCTYPE declaration of an XML document is selected. It appends a NOTATION declaration to the list of declarations.

Append Encoded External File

The XML | Append | Encoded External File command is available in Grid View only. It appends a binary encoded file, such as an image file, as encoded characters. The encoded external file is appended after the Grid View selection.

On clicking the command, the Insert Encoded External File dialog (screenshot below) pops up. In it you enter the path to the file, select the encoding you want, and specify whether the encoded file is to be inserted in an element or not.

You can browse for or enter the name of the external file to be encoded and embedded. Either a
Base-16 or Base-64 encoding must be specified. If you wish to enclose the encoded text in an element, then check the Create Element check box and specify the name of the desired element in the Create Element text box. If the Create Element check box is not checked, then the encoded text will be inserted directly at the cursor location.

On clicking OK, the encoded text of the selected file is inserted at the cursor location, with an enclosing element if this has been specified.

The encoded file is appended in Grid View.

23.4.3 Add Child

The XML | Add Child command, though enabled in all views, can be used in Grid View only. It opens a submenu (see screenshot) with which you can add the following child items to the currently selected element:

- The XML declaration and node types (Attribute, Element, Text, CDATA, Comment, Processing Instruction) in XML documents;
- DOCTYPE declarations and external DTD declarations in XML documents
- DTD declarations (ELEMENT, ATTLIST, ENTITY, and NOTATION) in DTD documents and internal DTD declarations of XML documents.
**Attribute**

Ctrl+Alt+I

The XML | Add Child | Attribute command is available in Grid View only and when an element node is selected. It inserts a new attribute as a child of the selected element node.

**Element**

Ctrl+Alt+E

The XML | Add Child | Element command is available in Grid View only. It inserts a new element as a child of the selected node.

**Text**

Ctrl+Alt+T

The XML | Add Child | Text command is available in Grid View only, and inserts new text content as a child of the selected item.

**CDATA**

Ctrl+Alt+D

The XML | Add Child | Cdata command is available in Grid View only, and inserts a new CDATA section as a child of the selected item.

**Comment**

Ctrl+Alt+M

The XML | Add Child | Comment command is available in Grid View only, and inserts a new Comment node as a child of the selected item.
The XML | Add Child | XML command is available in Grid View only and when the file is empty. It inserts a new XML declaration `<xml version="1.0" encoding="UTF-8"?>` as the first item in a document.

Please note: An XML document may contain only one XML declaration, which must appear at the very top of the file.

Processing Instruction

The XML | Add Child | Processing Instruction command is available in Grid View only and inserts a new Processing Instruction (PI) as a child of the selected item.

XInclude

The XML | Add Child | XInclude command is available in Grid View only, and enables you to insert an XInclude element as a child of the selected item. Selecting this command pops up the XInclude dialog (screenshot below).
The XML file to be included is entered in the href text box (alternatively, you can browse for the file by clicking the Browse (...) button to the right of the text box). The filename will be entered in the XML document as the value of the href attribute. The parse, xpointer, and encoding attributes of the XInclude element (xi:include), and the fallback child element of xi:include can also be inserted via the dialog. Do this by first checking the appropriate check box and then selecting/entering the required values. In the case of the fallback element, checking its check box only inserts the empty element. The content of the fallback element must be added subsequently in one of the editing views.

The parse attribute determines whether the included document is to be parsed as XML or text. (XML is the default value and therefore need not be specified.) The xpointer attribute identifies a specific fragment of the document located with the href attribute; it is this fragment that will be included. The encoding attribute specifies the encoding of the included document so that XMLSpy can transcode this document (or the part of it to be included) into the encoding of the including document. The contents of the fallback child element replace the xi:include element if the document to be included cannot be located.

Here is an example of an XML document that uses XInclude to include two XML documents:

```xml
<?xml version="1.0" encoding="UTF-16"?>
<AddressBook xsi:schemaLocation="http://www.altova.com/sv/myaddresses AddressBook.xsd"
    xmlns="http://www.altova.com/stylevision/tutorials/myaddresses"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xi="http://www.w3.org/2001/XInclude">
    <xi:include href="BusinessAddressBook.xml"/>
    <xi:include href="PersonalAddressBook.xml"/>
</AddressBook>
```

When this XML document is parsed, it will replace the two XInclude elements with the files
xml:base
When the XML validator of XMLSpy reads an XML document and encounters the include element in the XInclude namespace (hereafter xi:include), it replaces this element (xi:include) with the XML document named in the href attribute of the xi:include element. The document element (root element) of the included XML document (or the element identified by an XPointer) will be included with an attribute of xml:base in order to preserve the base URIs of the included element. If the resulting XML document (containing the included XML document/s or tree fragment/s) must be valid according to a schema, then the document element of the included document (or the top-level element of the tree fragment) must be created with a content model that allows an attribute of xml:base. If, according to the schema, the xml:base attribute is not allowed on this element, then the resulting document will be invalid. How to define an xml:base attribute in an element's content model using XMLSpy's Schema View is described in the xml:Prefixed Attributes section of the Schema View section of the documentation.

XPointers
XMLSpy supports XPointers in XInclude. The relevant W3C recommendations are the XPointer Framework and XPointer element() Scheme recommendations. The use of an XPointer in an XInclude element enables a specific part of the XML document to be included, instead of the entire XML document. XPointers are used within an XInclude element as follows:

```xml
<xi:include href="PersonalAddressBook.xml" xpointer="element(usa)"/>
<xi:include href="BusinessAddressBook.xml" xpointer="element(/1/1)"/>
<xi:include href="BobsAddressBook.xml" xpointer="element(/1/1)"/>
<xi:include href="PatsAddressBook.xml" xpointer="element(/1/1)"/>
```

In the element() scheme of XPointer, an NCName or a child sequence directed by integers may be used.

- In the first xi:include element listed above, the xpointer attribute uses the element scheme with an NCName of usa. According to the XPointer Framework, this NCName identifies the element that has an ID of usa.
- In the second xi:include listed above, the xpointer attribute with a value of element(/1/1) identifies, in the first step, the first child element of the document root (which, if the document is well-formed, will be its document (or root) element). In the second step, the first child element of the element located in the previous step is located; in our example, this would be the first child element of the document element.
- The xpointer attribute of the third xi:include listed above uses a combination of NCName and child sequence. This XPointer locates the first child element of the third child element of the element having an ID of usa.
- If you are not sure whether your first XPointer will work, you can back it up with a second one as shown in the fourth xi:include listed above: xpointer="element(usa) element(/1/1)". Here, if there is no element with an ID of usa, the back-up XPointer specifies that the first child element of the document element is to be selected. Additional backups are also allowed. Individual XPointers may not be separated, or they may be separated by whitespace: for example, xpointer="element(usa) element(addresses/1) element(/1/1)".

**Note:** The namespace binding context is not used in the element() scheme because the
element() scheme does not support qualified names.

**DOCTYPE**

The XML | Add Child | DOCTYPE command is only available in the Grid View of an empty document. It inserts a DOCTYPE declaration in an XML document. The DOCTYPE declaration can be used to declare an internal DTD subset.

**ExternalID**

The XML | Add Child | ExternalID command is available only when the DOCTYPE declaration of an XML file is selected in Grid View. This command inserts a Grid View row for an external identifier (PUBLIC or SYSTEM). You must enter the type of identifier and its value.

The Text View corresponding to the screenshot of the Grid View shown above looks something like this:

```xml
<!DOCTYPE OrgChart SYSTEM "orgchart.dtd" [
  <!ELEMENT name (#PCDATA)>
]>
```

Please note: A row for External-ID can be added as a child when the DOCTYPE item is selected, or it can be inserted or appended when one of the child items of the DOCTYPE item is selected, for example, the ELEMENT declaration name in the example above.

**ELEMENT**

The XML | Add Child | ELEMENT command is available in Grid View only, for DTD documents, or when the DOCTYPE declaration of an XML document is selected. It appends an ELEMENT declaration to the list of declarations.
ATTLIST

The XML | Add Child | ATTLIST command is available in Grid View only, for DTD documents, or when the DOCTYPE declaration of an XML document is selected. It appends an ATTLIST declaration to the list of declarations.

ENTITY

The XML | Add Child | ENTITY command is available in Grid View only, for DTD documents, or when the DOCTYPE declaration of an XML document is selected. It appends an ENTITY declaration to the list of declarations.

NOTATION

The XML | Add Child | NOTATION command is available in Grid View only, for DTD documents, or when the DOCTYPE declaration of an XML document is selected. It appends an NOTATION declaration to the list of declarations.

Encoded External File

The XML | Add Child | Encoded External File command is available in Grid View only. It adds a binary encoded file as a child node. The encoded external file is inserted as a child of the Grid View selection.

On clicking the command, the Insert Encoded External File dialog (screenshot below) pops up. In it you enter the path to the file, select the encoding you want, and specify whether the encoded file is to be inserted in an element or not.

On clicking OK, the encoded text of the selected file is inserted at the cursor location, with an
enclosing element if this has been specified.

The encoded file is added as a child in Grid View.

23.4.4 Convert To

The XML | Convert to command converts a selected item in Grid View to a different item type. This operation is available only in Grid View on individual items that do not contain any child node. Placing the cursor over the Convert to command displays a submenu (see screenshot) which contains the items to which the selected item can be converted.

<table>
<thead>
<tr>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
</tr>
<tr>
<td>Text</td>
</tr>
<tr>
<td>CDATA</td>
</tr>
<tr>
<td>Comment</td>
</tr>
<tr>
<td>XML</td>
</tr>
<tr>
<td>Processing Instruction</td>
</tr>
<tr>
<td>DOCTYPE</td>
</tr>
<tr>
<td>ExternalID</td>
</tr>
<tr>
<td>ELEMENT</td>
</tr>
<tr>
<td>ATTLIST</td>
</tr>
<tr>
<td>ENTITY</td>
</tr>
<tr>
<td>NOTATION</td>
</tr>
</tbody>
</table>

Please note: If the operation you select would result in a loss of data (for example, converting an attribute to a comment would result in a loss of the attribute name), a warning dialog box will appear.

Attribute

The XML | Convert to | Attribute command converts the selected item to a attribute.

Element

The XML | Convert to | Element command converts the selected item to an element.

Text

The XML | Convert to | Text command converts the selected item into text content.
CDATA
The XML | Convert to | Cdata command converts the selected item into a CDATA segment.

Comment
The XML | Convert to | Comment command converts the selected item into a comment.

XML
The XML | Convert to | XML command converts the selected item to an XML declaration:

<?xml version="1.0" encoding="UTF-8"?>.

Please note: Each XML document may only contain one XML declaration and it must appear at the very top of the file.

Processing Instruction
The XML | Convert to | Processing Instruction command converts the selected item to a new Processing Instruction (PI).

DOCTYPE
The XML | Convert to | DOCTYPE command converts the selected item to a DOCTYPE declaration (in an XML file).

Please note: A DOCTYPE declaration may only appear at the top of an XML instance document between the XML Declaration and the document element of the XML document.

ExternalID
The XML | Convert to | ExternalID command converts the selected item to an external DTD reference in a DOCTYPE declaration (PUBLIC or SYSTEM identifier).

ELEMENT
The XML | Convert to | ELEMENT command converts the selected item to an element declaration in a DOCTYPE declaration or in an external DTD.

ATTLIST
The XML | Convert to | ATTLIST command converts the selected item to an attribute list declaration in a DOCTYPE declaration or in an external DTD.
ENTITY

The XML | Convert to | ENTITY command converts the selected item to an entity declaration in a DOCTYPE declaration or in an external DTD.

NOTATION

The XML | Convert to | NOTATION command converts the selected item to a notation declaration in a DOCTYPE declaration or in an external DTD.

23.4.5 Table

The XML | Table command, though enabled in all views, can be used only in Grid View. It displays a submenu with all the commands relevant to the Database/Table View of Grid View.

<table>
<thead>
<tr>
<th>Command</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display as Table</td>
<td>F12</td>
</tr>
<tr>
<td>Insert Row</td>
<td>Shift+F12</td>
</tr>
<tr>
<td>Append Row</td>
<td>Ctrl+F12</td>
</tr>
<tr>
<td>Ascending Sort</td>
<td></td>
</tr>
<tr>
<td>Descending Sort</td>
<td></td>
</tr>
</tbody>
</table>

Display as Table

Display as Table (F12) menu command and toolbar icon

The XML | Table | Display as Table command allows you to switch between the standard Grid View and Database/Table View (or Table View) of a document element. The Table View enables you to view repeated elements as a table in which the rows represent the occurrences while the columns represent child nodes (including comments, CDATA sections, and PIs).

To switch to Table View:

1. Select any one occurrence of the repeating element you wish to view as a table.
2. Click **XML | Table | Display as Table** or **F12** or the toolbar icon. The element is displayed as a table and the **Display as Table** toolbar icon is toggled on.

To switch from the Table View of a document element to the normal Grid View of that element, select the table or any of its rows or columns, and click the **Display as Table** toolbar icon. That table element switches to Grid View, and the icon is toggled off.

**Note:** Table View colors can be set in the Colors section of the Options dialog (**Tools | Options | Colors**)

**Insert Row**

**Shift+F12**

The **XML | Table | Insert Row** command is enabled in **Database/Table View** when a row or cell is selected. It inserts a new row before the selected row. The new row corresponds to an occurrence of the table element. Mandatory child elements are created for the new element.

**Append Row**

**Ctrl+F12**
The **XML | Table | Append Row** command is enabled in **Database/Table View** when a row or cell is selected. It appends a new row after the last row of the table. The new row corresponds to an occurrence of the table element. Mandatory child elements are created for the new element.

### Ascending Sort

The **XML | Table | Ascending Sort** command is enabled in **Database/Table View** when a column or cell is selected. It sorts the column in either alphabetic or numeric ascending order. XMLSpy tries to automatically determine what kind of data is used in the column, and sorts on alphabetic or numeric order, as required. In case of uncertainty, you will be prompted for the sort method to use (see screenshot).

![Sort dialog](image)

### Descending Sort

The **XML | Table | Descending Sort** command is enabled in **Database/Table View** when a column or cell is selected. It sorts the column in either alphabetic or numeric descending order. XMLSpy tries to automatically determine what kind of data is used in the column, and sorts on alphabetic or numeric order, as required. In case of uncertainty, you will be prompted for the sort method to use (see screenshot).

![Sort dialog](image)
23.4.6  **Move Left**

![Left Arrow] Ctrl+L

The **XML | Move Left** command is available in Grid View only. It moves the selected node to the left by one level, thereby changing a child element into a sibling of its parent. This command is often referred to as the **Promote** command.

23.4.7  **Move Right**

![Right Arrow] Ctrl+R

The **XML | Move Right** command is available in Grid View only. It moves the selected node to the right by one level, thereby turning it into a child element of the preceding sibling element. This command is often referred to as the **Demote** command.

23.4.8  **Enclose in Element**

The **XML | Enclose in Element** command is enabled in Grid View only. It encloses a selected text range in a new element. The new element is created inline around the selected text. If you are editing a document based on a Schema or DTD, you will automatically be presented with a list of valid choices for the name of the element in which the text is to be enclosed.

For example, in the screenshot below, the text **Nanonull** in the **para** element is highlighted.

```
<para>
  Due to the fact that nanoelectronic software components are new and that sales are restricted to corporate customers, **Nanonull** and its product line have not received much media publicity in the company’s early years. This has however changed in recent months as trade journals have realized the importance of this revolutionary technology.
</para>
```

When you select the command **XML | Enclose in Element**, the text **Nanonull** is enclosed in a newly created inline element and a list appears offering a choice of **bold** or **italic** for the name of the element. These elements are defined in the schema as children of **para**.

```
<p>Due to the fact that nanoelectronic software components are new and that sales are restricted to corporate customers, **Nanonull** and its product line have not received much media publicity in the company’s early years. This has however changed in recent months as trade journals have realized the importance of this revolutionary technology.</p>
```

The selection you make will be the name of the new element. Alternatively, you can enter some other name for the element.
23.4.9 Evaluate XPath

The XML | Evaluate XPath command opens the Output Windows if these are not open and activates the XPath tab in the Output Windows. In the XPath tab, you can evaluate an XPath expression on the active document and see the results in the Output Window.

23.4.10 Check Well-Formedness

The XML | Check well-formedness (F7) command checks the active document for well-formedness by the definitions of the XML 1.0 specification. Every XML document must be well-formed. XMLSpy checks for well-formedness whenever a document is opened or saved, or when the view is changed from Text View to any other view. You can also check for well-formedness at any time while editing by using this command.

If the well-formedness check succeeds, a message is displayed in the Messages window (screenshot below).

If an error is encountered during the well-formedness check, a corresponding error message is displayed (screenshot below).

Based on information in the schema, options for a smart fix are also suggested if the well-formedness check was carried out in Text View or Grid View. To view a list of smart fix options,
click the **Show Smart Fix** button (see screenshot above). A pane with suggested smart fix options appears in the Messages window (see screenshot below). Note that errors in the Messages window are displayed one error at a time.

![Screenshot of Messages window with Smart Fix pane](image)

In the Smart Fix pane, select one of the suggested smart fixes and click either the **Fix + Validate** button or the **Fix** button (see screenshot above). The error will be corrected with the smart fix. Alternatively, you can double-click the smart fix you want. This action either fixes the error, or fixes the error and validates the document, according to the option selected in the dropdown Options list. The **Fix + Validate** command is useful because the validator will now be able to validate beyond the fixed error and pick up the next well-formed error or validation error, if there is any.

To hide the Smart Fix pane, click the **Hide Smart Fix** button (see screenshot above).

**Note:** The Messages window has nine tabs. The validation result is always displayed in the active tab. So you can validate one XML document in Tab-1 and retain the result in that tab. To validate a second document, switch to Tab-2 (or Tab-3 if you like) before running the check. If you do not switch tabs, Tab-1 (or the active tab) will be overwritten with the results of the latest validation.

### Validating from the Project window

The **Validate** command can also be applied to a file, folder, or group of files in the active project. Select the required file or folder in the Project Window (by clicking on it). Then click **XML | Validate** or **F8**. Invalid files in a project will be opened and made active in the Main Window, and the **File is not valid** error message will be displayed.

**Note:** The Messages window has nine tabs. The result of the well-formed check is always displayed in the active tab. So you can check the well-formedness of one XML document in Tab-1 and retain the result in that tab. To check the well-formedness of a second document, switch to Tab-2 (or Tab-3 if you like) before running the check. If you do not switch tabs, Tab-1 (or the active tab) will be overwritten with the results of the latest check.

It is generally not permitted to save a malformed XML document, but XMLSpy gives you a **Save Anyway** option. This is useful when you want to suspend your work temporarily (in a not well-formed condition) and resume it later.

**Note:** You can also use the **Check well-formedness** command on any file, folder, or group of
files in the active project window. Click on the respective item, and then on the Check Well-Formedness icon.

23.4.11 Validate XML

The XML | Validate (F8) command enables you to validate XML documents against DTDs, XML Schemas, and other schemas. Validation is automatically carried out when you switch from Text View to any other view. You can specify that a document be automatically validated when a file is opened or saved (Tools | Options | File). The Validate command also carries out a well-formedness check before checking validity, so there is no need to use the Check Well-Formedness command before using the Validate command.

If a document is valid, a successful validation message is displayed in the Messages window.

Otherwise, a message that describes the error is displayed. You can click on the links in the error message to jump to the node in the XML document where the error was found. See the next section below for a description of the error message and how to fix validation errors with the smart fixes of XMLSpy.

Validation errors and their fixes

When a validation error is displayed in the Messages window, the causes of the error are displayed in the left-hand pane (see screenshot below). If a cause is selected in the left-hand pane, then smart fixes for it, if available, are displayed in the right-hand pane. Smart fix suggestions are available in Text View and Grid View, and are based on information in the associated schema. To view smart fixes, click the Show Smart Fix button. Click Hide Smart Fix if you do not want these suggestions to be displayed. Note that errors of well-formedness (such as mismatched start and end tags), if such exist, are displayed prior to validation errors being displayed. So the Show/Hide Smart Fix button will be enabled only when a validation error is reached (that is, after all well-formedness errors have been corrected).
To apply a smart fix, either (i) double-click it, or (ii) select it and click either the **Fix** or **Fix + Validate** options (see screenshot below). The **Fix + Validate** command will validate beyond the fixed error and pick up the next error, if there is any.

In Text View, there are two additional indicators of a validation error (see screenshot below): (i) a red exclamation-mark icon in the line-numbering margin, and (ii) a red marker-square in the scroll bar (on the right of the window).

The light-bulb icon next to the exclamation-mark icon (see screenshot above) is the smart-fix icon. If you hover over it, all smart fixes across all causes of the error are displayed (see screenshot below). Select a smart fix to apply it.
Note: The validation error indicators and smart fixes described above are refreshed only when the XML | Validate (F8) command is executed; they are not updated in the background. So, after correcting an error, you must run the Validate (F8) command again to make sure that the error has indeed been fixed.

Note: The Messages window has nine tabs. The validation result is always displayed in the active tab. So you can validate one XML document in Tab-1 and retain the result in that tab. To validate a second document, switch to Tab-2 (or Tab-3 if you like) before running the check. If you do not switch tabs, Tab-1 (or the active tab) will be overwritten with the results of the latest validation.

Validating from the Project window
The Validate command can also be applied to a file, folder, or group of files in the active project. Select the required file or folder in the Project Window (by clicking on it). Then click XML | Validate or F8. Invalid files in a project will be opened and made active in the Main Window, and the File is not valid error message will be displayed.

Validating XML documents
To validate an XML file, make the XML document active in the Main Window, and click XML | Validate or F8. The XML document is validated against the schema referenced in the XML file. If no reference exists, an error message is displayed in the Messages window. As long as the XML document is open, the schema is kept in memory (see Flush Memory Cache in the DTD/Schema menu).

Validating schema documents (DTDs and XML Schema)
XMLSpy supports major schema dialects, including DTD and XML Schema. To validate a schema document, make the document active in the Main Window, and click XML | Validate or F8.

Validation messages
There are two kinds of messages:

- If the schema (DTD or XML Schema) is valid, a successful validation message is displayed in the Messages window.
- If the schema is not valid, an error message is displayed in the Messages window (screenshot below).
An error message shows each possible cause of that error separately. For example, in the screenshot above, four possible causes of the validation error are reported; the first one is expanded, the other three are collapsed. Each cause is divided into three parts:

1. A description of the possible cause. The description contains links to the relevant definition in the associated schema document. You can quickly go to the specific schema definition to see why exactly the document is invalid.
2. The location path to the node in the XML document that has caused the error. Clicking any node in this location path highlights that node in the document.
3. Detailed information about the error, as well as a link to the relevant paragraph in the schema specification. This is where the schema rules that specify the relevant legality are specified.

Note: If the validation is done in Text View, then clicking a link in the Messages window will highlight the corresponding definition in Text View. If the validation is done in Schema View, then clicking a definition link will open the definition in Schema View and allow you to edit the component directly.

Catalogs

XMLSpy supports a subset of the OASIS XML catalogs mechanism. The catalog mechanism enables XMLSpy 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 need to be changed in the catalog files only.) The catalog mechanism in XMLSpy works as follows:

- XMLSpy loads a file called `RootCatalog.xml`, which contains a list of catalog files that will be looked up. You can enter as many catalog files to look up, each in a `nextCatalog` element in `RootCatalog.xml`.
- The catalog files included in `RootCatalog.xml` are looked up and the URIs are resolved according to the mappings specified in the catalog files. You should take care not to duplicate mappings, as this could lead to errors.
- Two catalog files are supplied with XMLSpy. How these work is described in the section `Catalogs in XMLSpy`.
- The `PUBLIC` or `SYSTEM` identifier in the `DOCTYPE` statement of your XML file will be used for the catalog lookup. For popular schemas, the `PUBLIC` identifier is usually pre-defined, thus requiring only the URI in the catalog file to be changed when XML documents are used on multiple machines.

When writing your `CustomCatalog.xml` file (or other custom catalog file), use only the following
subset of the OASIS catalog in order for XMLSpy to process the catalog correctly. Each of the elements in the supported subset can take the xml:base attribute, which is used to specify the base URI of that element.

```xml
<catalog...>
  ...
  <public publicId="PublicID of Resource" uri="URL of local file"/>
  <system systemId="SystemID of Resource" uri="URL of local file"/>
  <rewriteURI uriStartString="StartString of URI to rewrite" rewritePrefix="String to replace StartString"/>
  <rewriteSystem systemIdStartString="StartString of SystemID" rewritePrefix="Replacement string to locate resource locally"/>
  <uri name="filename" uri="URL of file identified by filename"/>
  ...
</catalog>
```

**Note:**

- 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 any of these schemas. The referenced DTD files are included solely to provide XMLSpy with entry helper info for editing purposes should you wish to create documents according to these older recommendations. Also see next point.
- If you create a custom file extension for a particular schema (for example, the .myhtml extension 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 a line of text to CustomCatalog.xml. For the example extension mentioned, you should add the element `<spy:fileExtHelper ext="myhtml" uri="schemas/xhtml/xhtml1-transitional.dtd"/>` as a child of the `<catalog>` element. This would enable intelligent editing (auto-completion, entry helpers, etc) of .myhtml files in XMLSpy according to the XHTML 1.0 Transitional DTD.
- For more information on catalogs, see the XML Catalogs specification.

### Automating validation with RaptorXML 2019

**RaptorXML** is Altova's standalone application for XML validation, XSLT transformation, and XQuery transformation. It can be used from the command line, via a COM interface, in Java programs, and in .NET applications. Validation tasks can therefore be automated with the use of RaptorXML. For example, you can create a batch file that calls RaptorXML to perform validation on a set of documents and sends the output to a text file. See the RaptorXML documentation for details.

### 23.4.12 Validate XML on Server (high-performance)

**Ctrl+F8**

The XML | Validate on Server (high-performance) (Ctrl+F8) command validates the active XML document on the currently active RaptorXML Server using the active configuration. The command immediately carries out the validation and displays the results in the Messages window.
Note: The actual performance depends on the number of PC processor cores used by RaptorXML Server for the validation: The higher the number of cores used, the faster will be the processing.

If you have defined multiple configurations on multiple servers, you can select a server and one of its configurations as the active configuration. The active configuration will be used for subsequent validations. On placing the cursor over the Tools | Raptor Servers and Configurations command (see screenshot below), a submenu appears that contains all the added servers, together with the configuration of each. Select the server configuration you want to make the active configuration. In the screenshot below, the xbrl configuration of the server named Raptor-01 has been selected as the active configuration (indicated by the green arrow).

Note: You can also select the active configuration in the dropdown menu of the Validate on Server icon. This menu also has a command to validate EDGAR on the active server.

The Validate XML on Server (high-performance) (Ctrl+F8) command is also available in the Project entry helper. Right-click the project, a folder, or a file, and select Validate XML on Server to validate XML or XBRL data in the selected object.

Note: Raptor validation is available in Text View, Grid View, and XBRL View.

23.4.13 Update Entry Helpers

The XML | Update Entry Helpers command updates the Entry Helper windows by reloading the underlying DTD or Schema. If you have modified the XML Schema or DTD that an open XML document is based upon, it is advisable to update the Entry Helpers so that the intelligent editing information reflects the changes in the schema.

23.4.14 Namespace Prefix

The XML | Namespace Prefix command is available in Grid View and opens a dialog box in which you can set the namespace prefix of the selected element or attribute, and, in the case of elements, of its descendants as well.
You can choose to set the namespace prefix on either elements, attributes, or both. The namespace prefix is applied to the selected element or attribute, and, if an element is selected, to descendant nodes of the selected element.
23.5 DTD/Schema Menu

The DTD/Schema menu (screenshot below) contains commands to work with DTDs and XML Schemas.

23.5.1 Assign DTD

The DTD/Schema | Assign DTD... command is enabled when an XML file is active. It assigns a DTD to an XML document, thus allowing the document to be validated and enabling intelligent editing for the document. The command opens the Assign File dialog to let you specify the DTD file you wish to assign. You can also select a file via a global resource or a URL (click the Browse button) or a file in one of the open windows in XMLSpy (click the Window button). Note that you can make the path of the assigned DTD file relative by clicking the Make Path Relative To... check box. When you are done, your XML document will contain a DOCTYPE declaration that references the assigned DTD. The DOCTYPE declaration will look something like this:
Please note: A DTD can be assigned to a new XML file at the time the file is created.

23.5.2 Assign Schema

The DTD/Schema | Assign Schema... command is enabled when an XML document is active. It assigns an XML Schema to an XML document, thus allowing the document to be validated and enabling intelligent editing for the document. The command opens the Assign File dialog to let you specify the XML Schema file you wish to assign. You can also select a file via a global resource or a URL (click the Browse button) or a file in one of the open windows in XMLSpy (click the Window button). Note that you can make the path of the assigned file relative by clicking the Make Path Relative To... check box. When you are done, your XML document will contain an XML Schema assignment with the required namespaces. The schema assignment will look something like this:

```
xmlns="http://www.xmlspy.com/schemas/icon/orgchart"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xsi:schemaLocation="http://www.xmlspy.com/schemas/icon/orgchart
http://schema.xmlspy.com/schemas/icon/orgchart.xsd"
```

23.5.3 Include Another DTD

The DTD/Schema | Include another DTD... command allows you to include another Document Type Definition (DTD) or external parsed entity into the internal subset of a document type definition, or in any DTD document. This is done by defining a corresponding external parsed entity declaration and using that entity in the following line:

```
<!ENTITY % navigation.dtd SYSTEM "S:\xml\navigation.dtd">
%navigation.dtd;
```

The command opens the Assign File dialog to let you specify the DTD file you want to include in your DTD.

Please note: This command is enabled in Grid View only.

23.5.4 Go to DTD

The DTD/Schema | Go to DTD command opens the DTD on which the active XML document is based. If no DTD is assigned, then an error message is displayed.
23.5.5  Go to Schema

The DTD/Schema | Go to Schema command opens the XML Schema on which the active XML document is based. If no XML Schema is assigned, then an error message is displayed.

23.5.6  Go to Definition

The DTD/Schema | Go to Definition command displays the exact definition of an element or attribute in the corresponding Document Type Definition or Schema document.

*To see the item definition in Grid View*
1. Click left on the item.
2. Select the menu item DTD/Schema | Go to Definition, or click on the icon.

*To see the item definition in Schema View*
- Use CTRL + Double click on the item you want to see the definition of, or
- Click the item and select menu option DTD/Schema | Go to Definition, or click on the icon.

In both cases, the corresponding DTD or Schema file is opened, and the item definition is highlighted.

23.5.7  Generate DTD/Schema

The DTD/Schema | Generate DTD/Schema command generates a new DTD or W3C XML Schema from an XML document (or from a set of XML documents contained in a folder in the project window). This command is useful when you want to generate a DTD or XML Schema from XML documents.
If you generate an XML Schema, the following options are available:

- **Elements**: The type of elements can be defined locally or globally (*Define types for elements*). If elements have the same name, a common type can be declared for use in the definition of these elements (*Generate one shared type*).

- **Attributes**: The simple types of attributes (*Define simple types for attributes*) can be defined as (i) common global types; (ii) distinct global types; (iii) local types. Attributes with the same name and type can be defined either locally or globally.

- **Simple type recognition**: The recognition of types (*Simple type recognition*) can be set to: (i) best possible; (ii) recognition of number datatypes only; (iii) no datatype recognition, in which case all datatypes are set to `xs:string`.

- **Entity resolution**: In the XML document, entities may appear in element content and attribute values. Whether they are resolved or not (*Validate and resolve entities*) is therefore significant for enumeration values. Furthermore, some entities (especially parsed entities that contain markup) can affect the content model differently depending on whether they are resolved or not. Note that the XML document will be validated for being
correct XML before the schema is generated. If the document is invalid, the schema generation process will be discontinued.

- **Enumerations**: All types of values, or string values only, can be enumerated.

If you generate a DTD, the entity resolution and enumeration options are available.

The Generate DTD/Schema command normally operates on the active main window, but you can also use the Generate DTD/Schema command on any file, folder, or group of files in the active project window.

If elements or attributes in more than one namespace are present, XMLSpy generates a separate XML Schema for each distinct namespace; therefore, multiple files may be created on the disk.

### 23.5.8 Flatten DTD

The Flatten DTD command is enabled when a DTD is the active document. It creates a new flat DTD, removing parameter entities and producing a single DTD from a collection of modules. It also suppresses sections marked `IGNORE` and deletes unused parameter entities.

The command pops up a Save dialog, in which you select a location at which to save the generated DTD file. Click **Save** to carry out the conversion. The flattened DTD file is generated and opened in XMLSpy.

### 23.5.9 Convert DTD to Schema

The Convert DTD to Schema command is enabled when a DTD is the active document. It converts a DTD into an XML Schema document (XSD).

The command pops up the Convert DTD to W3C Schema dialog (screenshot below), in which you can select whether complex elements should be converted into elements or complex types. On clicking **OK**, you are prompted to select a location at which to save the generated XSD file. Click **Save** to carry out the conversion. The XSD file is generated and opened in XMLSpy.

When you convert a DTD to XML Schema, XMLSpy makes a few assumptions because of the limited information available. Most notably, the values of certain DTD components are treated literally rather than having their semantics parsed. This is because the program cannot know which of several possible usages is intended. In these cases, you should modify the generated conversion.

In any case, you should carefully examine the generated conversion to see if you can enhance it. A few areas in which improvements may be required are listed below.
Attribute Datatyping

DTDs allow for only 10 attribute datatypes, whereas XML Schemas, for instance, allow for more than 40 datatypes plus derived datatypes. You may wish to enhance a generated XML Schema, for example, by using a more restrictive datatype. Note that when an XML Schema is converted to DTD datatype information will be lost.

Namespaces

DTDs are not namespace-aware. As a result, if namespaces are to be specified in a DTD they must be hard-coded into element and attribute names. This could pose challenging problems when converting from one schema to another.

Entities

XML Schema does not have equivalents for the general entity declarations of DTDs. When XMLSpy converts a DTD to an XML Schema, it ignores entity declarations.

Unparsed data declarations

DTDs and XML Schemas use different mechanisms for handling unparsed data. This is explained in more detail below.

DTDs use the following mechanism:

- A notation is declared consisting of a name and an identifier, for example:
  ```xml
  <!NOTATION gif SYSTEM "image/gif">
  ```
- You declare the entity, for example:
  ```xml
  <!ENTITY cover_img SYSTEM "graphics/cover_img.gif" NDATA gif>
  ```
- Typically, you specify an attribute type of ENTITY on the relevant attribute, for example:
  ```xml
  <!ELEMENT img EMPTY>
  <!ATTLIST img format ENTITY #REQUIRED>
  ```

In XML Schema, the corresponding mechanism is as follows:

- Declare a notation. This functions in the same way as for the DTD.
  ```xml
  <xs:notation name="gif" public="image/gif"/>
  ```
  Note that the public attribute is mandatory and holds the identifier. An optional system attribute holds the system identifier and is usually an executable that can deal with resources of the notation type.
- You associate the notation declaration with a given attribute value using the NOTATION datatype. You cannot, however, use the NOTATION datatype directly, but must derive another datatype from the NOTATION datatype.
  ```xml
  <xs:simpleType name="formatType">
    <xs:restriction base="xs:NOTATION">
      <xs:enumeration value="gif"/>
      <xs:enumeration value="jpeg"/>
    </xs:restriction>
  </xs:simpleType>
  ```
- You associate the attribute with the datatype derived from the NOTATION datatype, e.g.
  ```xml
  <xs:complexType name="imgType">
    <xs:attribute name="height"/>
    <xs:attribute name="width"/>
  </xs:complexType>
  ```
When you convert a DTD to an XML Schema, XMLSpy does the following:

- Something like
  ```xml
  <!ATTLIST image format ENTITY #REQUIRED
  ...
  ```
  is converted to
  ```xml
  <xs:attribute name="format" type="xs:ENTITY" use="required"/>
  ```

- And something like
  ```xml
  <!NOTATION gif SYSTEM "image/gif">
  ```
  is converted to
  ```xml
  <xs:notation name="gif" system="image/gif"/>
  ```

You should therefore make the following modifications:

1. In notations like `<xs:notation name="gif" system="image/gif"/>` replace `system` with `public`, and add an optional system identifier if required.
2. Derive a datatype from the `NOTATION` datatype as described above for `formatType`.
3. Associate the derived datatype with the relevant attribute.

**Note:** According to the XML Schema specification, you do not need to—or cannot, depending on your viewpoint—declare an external entity.

### 23.5.10 Flatten Schema

The **Flatten Schema** command is enabled when an XML Schema is the active document. It generates a new flat XSD by (i) adding the components of all included schemas as global components of the active schema, and (ii) deleting the included schemas.

The command redirects to the **Flatten Schema** command of the Schema Design menu. Since the **Flatten Schema** command is available in Schema View only, you will be prompted about whether you wish to switch to Schema View or not. For more information, see the **Flatten Schema** command.

### 23.5.11 Convert Schema to DTD

The **Convert Schema to DTD** command is enabled when an XML Schema is the active document. It converts an XML Schema document (XSD) into a DTD.

The command pops up a Save dialog, in which you select a location at which to save the generated DTD file. Click **Save** to carry out the conversion. The DTD file is generated and opened in XMLSpy.
Note the following points:

1. When you convert an XML Schema to a DTD, the namespace prefixes used in the XML Schema—not the namespace URIs or the namespace declarations—are carried through to the names of the corresponding elements and attributes in the DTD.
2. Since XML parsers ignore namespaces when validating an XML document against a DTD, the namespace declarations themselves are not converted.
3. The `elementFormDefault` and `attributeFormDefault` attributes of the `xs:schema` element determine what elements and attributes have their prefixes included in the conversion process. If set to unqualified, then only globally declared elements and attributes, respectively, include prefixes in the conversion. If set to qualified, all element and attribute names have their prefixes included in the conversion.
4. Prefixes are converted to their corresponding string value plus a colon. Elements and attributes in default namespaces are converted to elements and attributes with names that begin with the string: `default_NS_X`, where X is an integer (starting with 1 and having a maximum value equal to the number of default namespaces used in the XML Schema).
5. In the DTD, element names are composed of parameter entities. This enables you to easily change the prefix in the DTD should the prefix in the XML document ever need to change. Parameter entity definitions can be changed either in the DTD document itself or by overriding the parameter entity definitions in the XML document's internal DTD subset.

**Note:** Namespaces have no semantic value in DTDs, and namespace prefixes carried over from the XML Schema become merely a lexical part of the name of the element or attribute defined in the DTD.

### 23.5.12 Convert to UML

The **DTD/Schema | Convert to UML** command converts a W3C XML Schema to an Altova UModel Project (.ump) document (hereafter UModel project). UMP is the native format of Altova UModel, Altova's UML modeling application. UMP files can then be viewed and edited in Altova UModel.

To convert a schema to UML, do the following:

1. With the schema open, click the **Convert to UML** command. This pops up the Convert to UML dialog (**screenshot below**).
2. In the Content Diagrams tab, select the option Generate Diagrams for XSD Globals. This will generate, in the UModel project, a content model diagram for each global component.
3. Select the required options from those available in the dialog. These options are explained below.
4. If you wish to view the created project in UModel immediately, select the option to open the project in UModel. Otherwise leave this option unselected.
5. Click **OK**.
6. In the Save As dialog that appears, browse for the destination folder, then enter the name of the UMP file, and click **Save**.

### Convert to UML options

The following options are available in the Convert to UML dialog.

In the **Content Diagrams** tab:

- **Hyperlink diagrams** creates in each diagram a link to the entry of that global component in the Model Tree view, thus enabling the component to be quickly located in the schema hierarchy.
- In the **Style** pane, the show compartments options enables various compartments to be either shown or hidden.

In the **Package Dependency Diagram** tab:

- The **Generate Diagram** option determines whether a package dependency diagram is generated. A package dependency diagram provides an overview of the entire package, showing the relationships of package components to one another. Note that the other options in this tab will be enabled only if the Generate Diagram option is selected.
- Selecting the **Hyperlink Package to Diagram** option creates a link from the package diagram to the Model Tree View.
- Four options are available for the layout of the package dependencies diagram: (i) unorganized layout (Autolayout option unselected); (ii) hierarchical layout (Autolayout and Hierarchical options selected); iii) block (Autolayout and Block options selected); and (iv) evenly spaced (Autolayout and Force Directed options selected). The layout can be modified by editing the diagram in UModel.
23.5.13 **Generate XML from DB, Excel, EDI with MapForce**

The DTD/Schema | Generate XML from DB, Excel, EDI with MapForce command launches Altova's MapForce if the application is installed. MapForce enables you to map a schema to another DTD, XML Schema, or database and to generate XML.

23.5.14 **Design HTML/PDF/Word Output with StyleVision...**

The DTD/Schema | Design HTML/PDF Output in StyleVision... command launches Altova's StyleVision if the application is installed. StyleVision enables you to design stylesheets for HTML, PDF, and RTF output.

23.5.15 **Generate Sample XML/JSON File**

The Generate Sample XML/JSON File command is enabled in Text View, Grid View, and Schema View, and generates an XML or JSON instance based on the currently active schema file. If the currently active file is a DTD or XML Schema, then an XML instance file is generated. If the currently active file is a JSON schema, then a JSON instance file is generated.

**Generating sample XML files**

With a DTD or XML Schema active, you can generate a sample XML instance based on the schema. On clicking the Generate Sample XML/JSON File command, the Generate Sample XML File dialog (screenshot below) appears, in which you can specify the options for the sample generation.

**Note:** The Convert to UML feature supports W3C XML Schemas only.
Elements of choice groups
A choice group is a group of elements from which one may be used. For example, if an element called *items* is defined as having a choice group consisting of the three elements: *cd*, *dvd*, *book*, then *items* can validly have any one of these three elements as a child element (with a maximum number of occurrences as specified in that element's *maxOccurs* attribute).

In the Generate Sample XML File dialog, you can select whether (i) the first branch (element) of the choice group, (ii) all branches, or (iii) the branch with the smallest number of descendant elements is generated. Note that the All branches selection could generate an invalid document since only one branch from a choice group is allowed.

If any of the choice group's branches are repeatable (that is, it has a *maxOccurs* value of greater than 1), then you specify, in the first text box of the dialog, how many of the repeatable elements to generate, up to a maximum of 99. If the *maxOccurs* attribute of the choice group is defined as unbounded or as a large number and All branches is selected in the Generate Sample XML File dialog, then the *maxOccurs* of the choice group is also limited by the number of repeatable
elements you specify in the first text box of the dialog.

**Generate non-mandatory elements**  
Activating this option generates both the mandatory and non-mandatory elements defined in the schema.

**Generate non-mandatory attributes**  
Activating this option generates both the mandatory and non-mandatory attributes defined in the schema.

**Generate X elements if marked repeatable in Schema/DTD**  
Activating this option generates the number of repeatable elements you enter in the text box. This applies to all elements, including those in choice groups.

**Fill elements and attributes with data**  
Activating this option inserts the datatype values of the respective elements and attributes. For example if an element is defined as being of datatype `string`, then the element is given a dummy value of `string`.

**Nillable elements and abstract types**  
The contents of nillable elements can be treated as non-mandatory, and elements with an abstract type can use a non-abstract type for its `xsi:type` attribute.

**Schema assignment for the generated XML file**  
The schema used to generate the XML file can be assigned to the generated XML file with a relative or absolute path.

**Use manually added sample values if available**  
If the schema component has sample values assigned to it, these will be used as the value or content of that component. For individual components, sample values are assigned in the Facets Entry Helper, in the Samples tab. Which value from the available sample values is selected for a single file generation can be specified:

- A random selection.
- Each sample value in turn for each instance of the component. For each file generation, the cycle starts anew.
- The first value always.

**Root element**  
If the schema contains more than one global element, these are listed, and the root element required for the sample XML file can be selected from the list.

**Namespaces**  
Click the **Namespaces** button to open the Edit Namespaces dialog *(screenshot below)*, The
namespaces that are defined in the schema, plus any standard XML Schema namespaces that are required in the sample XML file, will appear in this dialog.

You can edit the following:

- The namespace prefix that is bound to any of the document's namespaces. The namespace prefixes that are set in this dialog will be used (in the generated XML file) to prefix nodes that are in the corresponding namespace. For example, the screenshot indicates that nodes in the http://www.xmlspy.com/schemas/textstate namespace will be prefixed with ts: in the sample file.
- You can set one of the document's namespaces to be the default namespace (xmlns=) by selecting, in the xmlns= combo box, the namespace that you want. Nodes in the namespace that is selected as the default namespace will then be generated without a namespace prefix.

Generating sample JSON files

With a JSON schema active, you can generate a sample JSON instance based on the JSON schema. On clicking the command, the Generate Sample JSON File dialog (screenshot below) appears, in which you can specify the options for the sample generation.

You can choose whether to generate non-mandatory object properties, non-mandatory pattern properties, the length of arrays, and repetitions of recursive definitions. You can also specify whether the active JSON schema should be automatically assigned to the generated JSON sample file or not.
23.5.16 Flush Memory Cache

The DTD/Schema | Flush Memory Cache command flushes all cached schema (DTD and XML Schema) documents from memory. To speed up validation and intelligent editing, XMLSpy caches recently used schema documents and external parsed entities in memory. Information from these cached documents is also displayed when the Go to Definition command is invoked.

Flush the memory cache if memory is tight on your system, or if you have used documents based on different schemas recently.
23.6 **Schema Design Menu**

The **Schema Design** menu enables you to configure the Schema View of XMLSpy. This view enables you to design XML Schemas in a GUI. It is available when an XML Schema document is active in Schema View.

The commands available in this menu are described in this section.

23.6.1 **Schema Settings**

The **Schema Design | Schema Settings** command is enabled in Schema View and lets you define global settings for the active schema. These settings are the attributes of the `xs:schema` element.
The settings defined in the Schema Settings dialog above (when XSD mode is set to 1.1) will create the following `xs:schema` element.

```xml
<xs:schema xmlns="http://www.altova.com/schemas/org"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.altova.com/schemas/org"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified"
           xpathDefaultNamespace="##targetNamespace"
           version="1.1"
           defaultAttributes="Contact">
</xs:schema>
```

**Note:** What's in the Schema Settings dialog will differ according to the active XSD mode. When XSD 1.0 is the active mode, XSD 1.1 attributes are not present in the dialog.

The `defaultAttributes` and `xpathDefaultNamespace` attributes are XML Schema 1.1 features and will be available only in **XSD 1.1 mode**. The other attributes are available in both XSD 1.0 and XSD 1.1.
The `version` attribute

The `version` attribute is the document version. It is not the XSD version of the document.

The `defaultAttributes` attribute

The `defaultAttributes` attribute enables you to select an attribute group as the default attribute group of all complex types in the schema. The default attribute group is displayed in the content model of these complex types. In the screenshot below, the `Contact` attribute group is the default attribute group (also see screenshot above, where this has been set), and is automatically available on the `group` element. To disable the attribute group, set the complex type’s `defaultAttributesApply` to `false`. You can do this via the `defAttrs` property in the Details entry helper of the complex type (see screenshot below).

![Schematic Diagram of defaultAttributes Attribute](image)

The `xpathDefaultNamespace` attribute

The `xpathDefaultNamespace` attribute sets the default namespace for elements in XPath expressions used in the schema. If set in the Schema Settings dialog, the attribute is applied to the top-level `xs:schema` element. So the scope of the declaration will be the entire document. You can override the declaration on `xs:schema` with declarations on elements where the attribute is allowed:

- `xs:assert` and `xs:assertion`
- `xs:alternative`
- `xs:selector` and `xs:field` (in identity constraints)

You can change the XPath default namespace in the Details entry helper of the elements listed above.
The `xpathDefaultNamespace` attribute can have one of three allowed values:

- `##targetNamespace`: The XPath default namespace will be the same as the target namespace of the schema
- `##defaultNamespace`: The XPath default namespace will be the same as the default namespace of the schema
- `##local`: There is no XPath default namespace

If no XPath default namespace is declared in the document, unprefixed elements in XPath expressions will be in no namespace. The XPath default namespace declaration does not apply to attributes.

### 23.6.2 Save Diagram

The **Schema Design | Save Diagram...** command saves the diagram of the Content Model currently displayed in the Main Window in PNG format to any desired location.

![Save As dialog](image)

### 23.6.3 Generate Documentation

The **Schema Design | Generate Documentation** command generates detailed documentation about your XML or JSON schema (see screenshot below) 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 (JSON) Schema Documentation dialog (which appears when you select the **Generate Documentation** command). Related elements (child elements, complex types, etc.) are typically hyperlinked in the onscreen output, enabling you to navigate from component to component. Components with a content model also have links to the content model definitions. Note that...
schema documentation is also generated for **included and imported schema components**. The various documentation-generation options for XML Schema are described in the section **Documentation Options**. JSON schema documentation options are described in the section **Generating JSON Schema Documentation**.

Note that the **Documentation Options** are applied on top of the settings you specify in the **Schema Display Configuration dialog**.

**Note:** In order to generate documentation in MS Word format, you must have MS Word (version 2000 or later) installed.

You can either use XMLSpy's fixed standard design for the generated document, or you can use a StyleVision SPS for the design. Using a StyleVision SPS enables you to customize the design of the generated documentation as well as to generate PDF as an additional output format. How to work with an SPS is explained in the section, **User-Defined Design**.

**Note:** In order to use an SPS to generate schema documentation, you must have StyleVision installed on your machine.

The screenshot above shows generated schema documentation with an index (all related schemas with their global components organized by component type) at the top of the document.

**Note:** When generating documentation for W3C schema documents, XMLSpy uses application-
internal versions of these documents. Consequently, other locations of these documents are not considered, and redefinitions and other schema modifications will not be reflected in the documentation.

Documentation Options

The Schema Design | Generate Documentation command generates detailed documentation of the active schema: XML Schema or JSON schema. This section describes the generation of XML Schema documentation. The procedure for generating JSON schema documentation is similar. For details about generating JSON schema documentation and a description of documentation generation settings, see the section Generating JSON Schema Documentation.

Generating XML Schema documentation

If an XML Schema document is active and you click the Generate Documentation command, the Schema Documentation dialog (screenshot below) is displayed. In this dialog, you can select options for the documentation.

In the Documentation Design pane of the dialog you can select whether to use the fixed XMLSpy design for the generated documentation or whether to use a customized design created in a StyleVision SPS. Select the option you want. Note that PDF output is available only for documentation generated with a StyleVision SPS, not for documentation generated using a fixed design. How to work with a user-defined design is described in the section, User-Defined Design.
The other options in the Schema Documentation dialog are explained below:

- The required format is specified in the Output Format pane: either HTML, Microsoft Word, RTF, or PDF. (The PDF output format is only available if you use a StyleVision SPS to generate the documentation.) On clicking OK, you will be prompted for the name of the output file and the location to which it should be saved.
- 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 documentation can be generated either as a single file or be split into multiple files. When multiple files are generated, each file corresponds to a component. What components are included in the output is specified using the check boxes in the Include pane. In fixed designs, links between multiple documents are created automatically.
- For HTML output, the CSS style definitions can be either saved in a separate CSS file or embedded in the HTML file (in the <head> element). If a separate CSS file is created, it will be given the same name as the HTML file, but will have a .css extension. Check or uncheck the Embed CSS in HTML check box to set the required option.
- The Embed Diagrams option is enabled for the MS Word, RTF, and PDF output options. When this option is checked, diagrams are embedded in the result file, in PNG format. Otherwise diagrams are created as PNG files, which are displayed in the result file via

\[\text{Diagram}\,\text{Namespace}\,\text{Type}\,\text{Substitution Group}\,\text{Attributes}\,\text{Identity Constraints}\,\text{Annotations}\,\text{Source code}\]
When the output is HTML, all diagrams are created as external PNG files. If the Create folder for diagrams check box is checked, then a folder will be created in the same folder as the HTML file, and the PNG files will be saved inside it. This folder will have a name of the format HTMLFilename_diagrams. If the Create folder for diagrams check box is unchecked, the PNG files will be saved in the same folder as the HTML file.

Links to local files (such as diagram image files and external CSS file) can be relative or absolute. In the Generate links to local files pane, select the appropriate radio button according to the option you prefer.

In the Include pane, you select which item types you want to include in the documentation. Each item of the selected types will be displayed in the generated documentation. For example, if Local Attributes is checked, then the description of each local attribute is displayed as a separate entry. The Index option lists all related schemas at the top of the file, with their global components organized by component type. The Check All and Uncheck All buttons enable you to quickly select or deselect all the options in the pane. Note that the Include option does not affect the display of an item type within the graphical definitions. That display is controlled by the settings you make in the Schema Display Configuration dialog. So if you wish to disable the display of attributes within the graphical representation of a schema item, then uncheck the Attributes option in the Schema Display Configuration dialog.

The Details pane lists the details that may be included for each component. Select the details you wish to include in the documentation. The Check All and Uncheck All buttons enable you to quickly select or deselect all the options in the pane.

The Show Result File option is enabled for all output options. When this option is checked, the result files are displayed in Browser View (HTML output), MS Word (MS Word output), and the default applications for .rtf files (RTF output) and .pdf files (PDF output).

Parameter values
If the StyleVision SPS contains one or more parameter definitions, then on clicking OK, a dialog pops up listing all the parameters defined in the SPS. You can enter parameter values in this dialog to override the default parameter values that were assigned in the SPS.

User-Defined Design
Instead of the fixed standard XMLSpy design, you can create a customized design for schema documentation. The customized design is created in a StyleVision SPS, which is a design template for the output document.

Creating the SPS
A StyleVision Power Stylesheet (or SPS) is created using Altova’s StyleVision product. An SPS for generating schema documentation must be based on an XML Schema that specifies the structure of the schema documentation. This schema is called SchemaDocumentation.xsd, and it is delivered with your XMLSpy package. It is stored in the folder: C:\Documents and Settings \<username>\My Documents\Altova\XMLSpy2019\Documentation\Schema.

When creating the SPS design in StyleVision, nodes from the SchemaDocumentation.xsd schema are placed in the design template and assigned styles and properties. Additional components, like links, tables and images, can also be added to the SPS design. In this way, the entire output document can be designed in the SPS. 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 schema documentation is that you have complete control over the schema documentation design. Note also that PDF output of the schema documentation is available only if a user-defined SPS is used; PDF output is not available if the fixed XMLSpy design is used.

Specifying the SPS to use for schema documentation

After an SPS has been created, it can be used to generate schema documentation. The SPS you wish to use for generating the schema documentation is selected in the Schema Documentation dialog (accessed via the Schema Design | Generate Documentation command). In the Documentation Design pane of this dialog (see screenshot below), select the Use User-Defined Design radio button. You can then click the Browse button and browse for the SPS you want. Click the dialog’s OK button, and, in the Save dialog that pops up, select the folder for, and enter the name of, the output file.

Note: The SPS file must correctly locate the schema on which it is based: SchemaDocumentation.xsd (see above).

The following editable SPS designs for schema documentation generation are delivered with XMLSpy. They are in the Altova\XMLSpy2019\Documentation\Schema\ subfolder of the (My) Documents folder:

- OverallDocumentation.sps, which generates full documentation about the schema
- Statistics.sps, which lists the number of global and local elements, attributes and attribute groups, and simple and complex types for the main schema and for each schema file independently
- StructureOverview.sps, which outputs a structure of global elements and complex types up to a configurable depth
- TypeDerivations.sps, which lists simple and complex types and all their directly and indirectly derived types in the form of a tree
- XPathGenerator.sps, which generates all possible XPath statements up to a configurable depth

These files, together with other SPS files you have recently browsed for, will be available in the combo box of the Use User-Defined option (see screenshot above).

Clicking the Edit button in the Documentation Design pane launches StyleVision and opens the
selected SPS in a StyleVision window. In order to preview the result document in StyleVision, you will need a Working XML file. The SPS designs listed above have already been assigned a sample XML file named Sample.xml, which is located in the (My) Documents folder, in the following subfolder:

Altova\XMLSpy2019\Documentation\Schema\SampleData

Note: In order to use an SPS to generate schema documentation, you must have StyleVision installed on your machine.

### 23.6.4 Configure View

The **Schema Design | Configure view** command is active in Content Model View and allows you to configure the Content Model View. Clicking the command opens the Schema Display Configuration dialog at the bottom right of the XMLSpy window, enabling you to see the effect of your settings as you enter them in the dialog. The settings take effect when you click the **OK** button of the dialog, and apply to the Content Model View of all XML Schema files that are opened subsequently. These settings also apply to the schema documentation output and printer output. For example, if you wish to disable the display of attributes within the graphical representation of a schema item in the schema documentation output, then uncheck the Attributes option in the Schema Display Configuration dialog (screenshot below).

Note: For a description of how to configure JSON Schema Design View, see the section Configuring JSON Schema Design View.

### Defining property descriptor lines for the content model

You can define what properties of elements and attributes are displayed in the Content Model View. These properties appear as grid lines in component boxes.

To define property descriptor lines:

1. Select **Schema Design | Configure view**. The Schema display configuration dialog appears.
2. In the **Element** or **Attribute** tab, click the Append or Insert icon to add a property descriptor line. The line is added in the dialog and to element boxes in the Content Model View.
3. From the combo box, select the property you want to display. See screenshot.
4. Repeat steps 2 and 3 for as many properties as required.
The Content Model View is updated, showing the defined property descriptor lines for all elements for which they exist.

**Note:** For attributes, the configuration you define appears only when attributes are displayed in the diagram (as opposed to them being displayed in a pane below the Content Model View). The configured view applies to all Content Model Views opened after the configuration is defined.

**Deleting a property descriptor line from the Content Model View**

To delete individual property descriptor lines, in the Schema Display Configuration dialog, select the property descriptor line you want to delete, and click the Delete icon \(\times\).

**Settings for configuring the Content Model View**

The Content Model View can be configured using settings in the Schema Display Configuration dialog. How to define what property descriptor lines are displayed in Content Model View has been described above. The other settings are described below.
You can define whether a property descriptor line is to contain single or double content, and whether individual lines must appear for every element or only for elements that contain that property. Use the appropriate radio buttons to define your settings. Note that these two settings can be set for individual lines separately (select the required line and make the setting).

**Common line settings**
This option toggles the line descriptions (i.e. the name of the property) on and off.
Widths
These sliders enable you to set the minimum and maximum size of the element rectangles in Content Model View. Change the sizes if line descriptor text is not fully visible or if you want to standardize your display.

Distances
These sliders let you define the horizontal and vertical distances between various elements onscreen.

Show in diagram
The Annotations check box toggles the display of annotation text on or off, as well as the annotation text width with the slider. You can also toggle the display of the substitution groups on or off. The Attributes and Identity Constraints appear in the Content Model diagram if their check boxes are selected; otherwise they appear as tabs in a pane at the bottom of the Content Model window.

Draw direction
These options define the orientation of the element tree on screen, horizontal or vertical.

Editing the content model in the diagram itself
You can change element properties directly in the content model diagram. To do this, double-click the property you wish to edit and start entering data. If a selection is available, a drop-down list appears, from which you can select an option. Otherwise, enter a value and confirm with Enter.

Buttons in the Schema display configuration dialog
This dialog has the following buttons:

- The Load/Save button allows you to load and save the settings you make here.
- The Predefined button, resets the display configuration to default values.
- The Clear all button empties the list box of all entries.

Enabling smart restrictions
To enable smart restrictions, check the Enable Schema Restrictions check box.

23.6.5 Zoom
The Schema Design | Zoom command controls the zoom factor of the Content Model View. This feature is useful if you have a large content model and wish to zoom out so that the entire content model fits in the Main Window. You can zoom between 10% and 200% of actual size.

To zoom in and out, either drag the slider or click in the entry box and enter a percentage value.
23.6.6 Display All Globals

The **Schema Design | Display AllGlobals** command switches from **Content Model View** to **Schema Overview** to display all global components in the schema. It is a toggle with the Display Diagram command. The currently selected toggle is indicated with a check mark to its left (see screenshot).

Alternatively, you could use the **Display All Globals** icon at the top of the Content Model View to switch to the Schema Overview.

23.6.7 Display Diagram

The **Schema Design | Display Diagram** command switches to the **Content Model View** of the selected global component—if the selected component has a content model. Global components that have a content model (complex types, elements, and element groups) are indicated with the icon to its left. The Display Diagram command is a toggle with the Display All Globals command. The currently selected toggle is indicated with a check mark to its left (screenshot below).

Alternatively, you could use the following methods to switch to Content Model View:

- Click the icon next to the component, the content model of which you want to display.
- Double-click a component name in the Component Navigator Entry Helper (at top right).

23.6.8 Schema Extensions for Databases

This menu item pops out a sub-menu containing commands for Oracle and MS SQL Server schema extensions.

- **Enable Oracle Schema Extensions**
- **Oracle Schema Settings**
- **Enable Microsoft SQL Server Schema Extensions**
- **Named Schema Relationships**
- **Unnamed Element Relationships**

**Enable Oracle Schema Extensions**

XMLSpy provides support for Oracle schema extensions for use with Oracle 9i Project XDB. Using these schema extensions allows you to configure and customize how Oracle 9i Project XDB...
stores XML documents. These XML documents are then accessible through SQL queries and legacy tools. Please see the Oracle Website for more information.

When you select the **Enable Oracle Schema Extensions** command, the following occurs:

- An Oracle tab is created in the Details Entry Helper, enabling you to add attributes—including XDB-specific attributes—to schema elements such as `xsd:complexType` and `xsd:element`.

Oracle extensions can be defined for complex types, elements, and attributes. Use the Entry Helper as you normally would in XMLSpy.

**Please note:** This menu command can be toggled on and off, that is, extensions can be enabled or disabled. When Oracle extensions are enabled, the command is displayed with a check mark to its left. Disabling Oracle extensions (by clicking the enabled command) deletes the XDB namespace declaration and all XDB extensions in the file. A warning message appears since this action cannot be undone.

**Oracle Schema Settings**

The **Oracle Schema Settings** command allows you to define global settings for Oracle schema extensions.

In order to access this dialog, Oracle schema extensions must be enabled (using the **Enable Oracle Schema Extensions** command).
Enable Microsoft SQL Server Schema Extensions

XMLSpy provides support for Microsoft SQL Server 2000 schema extensions for use with Microsoft SQL Server. Using these schema extensions allows you to configure and customize how Microsoft SQL Server stores XML documents. These XML documents are then accessible through SQL queries and legacy tools. Please see the Microsoft Website for more information.

When you select the Enable Microsoft SQL Server Schema Extensions command, the following occurs:

- The SQL Server namespace is declared on the schema element:
  `xmlns:sql="urn:schemas-microsoft-com:mapping-schema"`.
- An SQL Server tab is created in the Details Entry Helper, enabling you to add attributes to schema elements such as `xsd:element`.

Where SQL Server extensions can be defined for a schema component, the SQL Server tab is available in the Details Entry Helper when the component is selected. Use the Entry Helper as you normally would in XMLSpy.

Please note: This menu command can be toggled on and off, that is, extensions can be enabled or disabled. When SQL Server extensions are enabled, the command is displayed with a check mark to its left. Disabling SQL Server extensions (by clicking the enabled command) deletes the SQL Server namespace declaration and all SQL extensions in the file. A warning message appears since this action cannot be undone.

Named Schema Relationships

The Named Schema Relationships command allows the definition of named relationships to provide the information needed to create the document hierarchy. You have to have previously enabled the SQL Server schema extensions, using the menu option "Enable SQL Server Schema Extensions", to be able to access this menu option.

To create a named schema relationship:

1. Click the insert or append icon, to add a new row to the dialog box.
2. Click the field and enter the corresponding relationship name.
3. Click **OK** to confirm.

![Microsoft SQL Server Relationship Definitions](image)

This generates a SQL relationship element, placing it just after the namespace declaration.

**Please note:** Click the delete icon ✗, to delete a row from the dialog box.

### Unnamed Element Relationships

The **Unnamed Element Relationships** command allows the definition of unnamed relationships to provide the information needed to create the document hierarchy. You have to have previously enabled the SQL Server schema extensions, using the menu option **Enable Microsoft SQL Server Schema Extensions**, to be able to access this menu option.

To create an unnamed schema relationship:

1. Click the insert or append icon ‾, to add a new row to the dialog box.
2. Click the field and enter the corresponding name.
3. Click **OK** to confirm.

![Microsoft SQL Server Relationship Definitions](image)

This generates a SQL relationship element for the currently selected schema element.

**Please note:** Click the delete icon ✗, to delete a row from the dialog box.
23.6.9 Connect to SchemaAgent Server

The **Schema Design | Connect to SchemaAgent Server** command is enabled when an XML Schema document is active and it enables you to connect to a SchemaAgent Server. You are able to connect to a SchemaAgent server only if a licensed Altova SchemaAgent product is installed on your machine. When you click this command, the Connect to SchemaAgent Server dialog *(screenshot below)* opens:

![Connect to SchemaAgent Server dialog](image)

You can use either the local server (the SchemaAgent server that is packaged with Altova SchemaAgent) or a network server (the Altova SchemaAgent Server product, which is available free of charge). If you select **Work Locally**, the local server of SchemaAgent will be started when you click **OK** and a connection with it will be established. If you select **Connect to Network Server**, the selected SchemaAgent Server must be running in order for a connection to be made.

When connected to SchemaAgent Server, XMLSpy acts as a SchemaAgent client, and provides powerful and enhanced schema editing and management functionality. For details about SchemaAgent, the installation of SchemaAgent Server, and how to connect to SchemaAgent Server, see **SchemaAgent** in the DTD and XML Schema section of this user manual. For more information about installing and working with these two products, see the SchemaAgent user manual that is delivered with these products.

After you connect to SchemaAgent Server, a message appears in the bar at the top of the Main Window with information about the connection. You now have full access to all schemas and schema components in the search path/s (folder/s) defined for the SchemaAgent server to which XMLSpy is connected.

**Please note:** In order for the connection to succeed, you must have Altova's SchemaAgent Client product installed with a valid license on the same machine as that on which XMLSpy is installed.

23.6.10 Disconnect from SchemaAgent Server

![Disconnect from SchemaAgent Server](image)
The **Disconnect from SchemaAgent Server** command is enabled when a connection to a SchemaAgent Server has been made successfully. Selecting this command disconnects XMLSpy from the SchemaAgent Server.

### 23.6.11 Show in SchemaAgent

The **Show in SchemaAgent** menu item causes the active schema and, optionally, linked schemas to be displayed in the Altova product SchemaAgent. (This product must be installed on the same machine as XMLSpy if you wish to use SchemaAgent functionality). The schema/s are opened in a new SchemaAgent Design in SchemaAgent.

Mousing over the **Show in SchemaAgent** menu item pops out a submenu with options about what schemas to show in SchemaAgent. These options are described in **SchemaAgent** in the DTD and XML Schema section of this user manual.

### 23.6.12 SchemaAgent Validation

The **SchemaAgent Validation** command enables you to validate the currently active schema as well as schemas related to the currently active schema. This feature is described in detail in the **SchemaAgent Validation** section in the Schema View section of this user manual.

### 23.6.13 Create Schema Subset

The **Create Schema Subset** command pops up the Select Schema Components dialog (*screenshot below*). In this dialog, you check the component or components you wish to create as a single schema subset, then click **Next**. (Note that a check box below the pane enables components from all referenced files to also be listed for selection.)
In the Schema Subset Generation dialog that now appears (screenshot below), enter the name/s you want the file/s of the schema subset package to have. You must also specify the folder in which the new schema subset files are to be saved. A schema subset package could have multiple files if one or more of the components being created is an imported component in the original schema. A separate schema file is created for each namespace in the schema subset. The filenames displayed in the dialog are, by default, the names of the original files. But since you are not allowed to overwrite the original files, use new filenames if you wish to save the files in the same folder as the original files.
On clicking **OK**, the schema subset file with the namespace corresponding to that of the active file is opened in Schema View. Any other files in the package are created but not opened in Schema View.

### 23.6.14 Flatten Schema

Flattening the active schema in Schema View is the process of: (i) adding the components of all included schemas as global components of the active schema, and (ii) deleting the included schemas.

To flatten the active schema, select the command **Schema Design | Flatten Schema**. This pops up the Flatten Schema dialog (*screenshot below*), which contains the names of separate files, one for each namespace that will be in the flattened schema. These default names are the same as the original filenames. But since you are not allowed to overwrite the original files, the filenames must be changed if you wish to save in the same folder as the active file. You can browse for a folder in which the flattened schema and its associated files will be saved.
On clicking **OK**, the flattened schema file will be opened in Schema View.
The XSL Transformation language lets you specify how an XML document should be converted into other XML documents or text files. One kind of XML document that is generated with an XSLT document is an FO document, which can then be further processed to generate PDF output. XMLSpy contains built-in XSLT processors (for XSLT 1.0, XSLT 2.0, and XSLT 3.0) and can link to an FO processor on your system to transform XML files and generate various kinds of outputs. The location of the FO processor must be specified in the XSL section of the Options dialog (Tools | Options) in order to be able to use it directly from within the XMLSpy interface.

XMLSpy also has a built-in XQuery engine, which can be used to execute XQuery documents (with or without reference to an XML document).

Commands to deal with all the above transformations are accessible in the XSL/XQuery menu. In addition, this menu also contains commands to work with the Altova XSLT/XQuery Debugger.
23.7.1 XSL Transformation

The XSL/XQuery | XSL Transformation command transforms an XML document using an assigned XSLT stylesheet. The transformation can be carried out using the appropriate built-in Altova XSLT Engine (Altova XSLT 1.0 Engine for XSLT 1.0 stylesheets; Altova XSLT 2.0 Engine for XSLT 2.0 stylesheets; Altova XSLT 3.0 Engine for XSLT 3.0 stylesheets), the Microsoft-supplied MSXML module, or an external XSLT processor. The processor that is used in conjunction with this command is specified in the XSL section of the Options dialog (Tools | Options).

If your XML document contains a reference to an XSLT stylesheet, then this stylesheet is used for the transformation. (An XSLT stylesheet can be assigned to an XML document using the Assign XSL command. If the XML document is part of a project, an XSLT stylesheet can be specified on a per-folder basis in the Project Properties dialog. Right-click the project folder/s or file/s you wish to transform and select XSL Transformation.) If an XSLT stylesheet has not been assigned to an XML file, you are prompted for the XSLT stylesheet to use. You can also select a file via a global resource or a URL (click the Browse button) or a file in one of the open windows in XMLSpy (click the Window button).

Automating validation with RaptorXML 2019

RaptorXML is Altova's standalone application for XML validation, XSLT transformation, and XQuery transformation. It can be used from the command line, via a COM interface, in Java programs, and in .NET applications. XSLT transformation tasks can therefore be automated with the use of RaptorXML. For example, you can create a batch file that calls RaptorXML to run XSLT transformations on a set of documents and sends the output to a text file. See the RaptorXML documentation for details.

Transformations to ZIP files

In order to enforce output to a ZIP file, including Open Office XML (OOXML) files such as .docx, one must specify the ZIP protocol in the file path of the output file. For example:

```
filename.zip|zip/filename.xxx
filename.docx|zip/filename.xxx
```

Note: The directory structure might need to be created before running the transformation. If you are generating files for an Open Office XML archive, you would need to zip the archive files in order to create the top-level OOXML file (for example, .docx).

23.7.2 XSL-FO Transformation

FO is an XML format that describes paged documents. An FO processor, such as the Apache XML Project's FOP, takes an FO file as input and generates PDF as output. The production of a
PDF document from an XML document is, therefore, a two-step process.

1. The XML document is transformed to an FO document using an XSLT stylesheet.
2. The FO document is processed by an FO processor to generate PDF (or some alternative output).

The **XSL/XQuery | XSL:FO Transformation** command transforms an XML document or an FO document to PDF.

- If the **XSL:FO Transformation** command is executed on a source XML document, then both of the steps listed above are executed, in sequence, one after the other. If the XSLT stylesheet required to transform to FO is not referenced in the XML document, you are prompted to assign one for the transformation. Note that you can also select a file via a global resource or a URL (click the **Browse** button) or a file in one of the open windows in XMLSpy (click the **Window** button). The transformation from XML to XSL-FO is carried out by the XSLT processor specified in the **XSL section** of the Options dialog (**Tools | Options**). By default the selected XSLT processor is XMLSpy's built-in XSLT processor. The resultant FO document is directly processed with the FO processor specified in the **XSL section** of the Options dialog (**Tools | Options**).
- If the **XSL:FO Transformation** command is executed on an FO document, then the document is processed with the FO processor specified in the **XSL section** of the Options dialog (**Tools | Options**).

**XSL:FO Transformation output**

The **XSL:FO Transformation** command pops up the Choose XSL:FO Output dialog (screenshot below). (If the active document is an XML document without an XSLT assignment, you are first prompted for an XSLT file.)

You can view the output of the FO processor directly on screen using FOP viewer or you can generate an output file in any one of the following formats: PDF, text, an XML area tree, MIF PCL, or PostScript. You can also switch on messages from the FO processor to show (i) the
processor's standard output message in the Messages window; and (ii) the processor's error messages in the Messages window. To switch on either these two options, check the appropriate check box at the bottom of the dialog.

Note:

- Unless you deselected the option to install the FOP processor of the [Apache XML Project](http://xml.apache.org), it will have been installed in the folder `C:\ProgramData\Altova\SharedBetweenVersions`. If installed, the path to it will automatically have been entered in the XSL section of the Options dialog (Tools | Options) as the FO processor to use. You can set the path to any FO processor you wish to use.

- The XSL:FO Transformation command can not only be used on the active file in the Main Window but also on any file or folder you select in the active project. To do this, right-click and select XSL:FO Transformation. The XSLT stylesheet assigned to the selected project folder is used.

### 23.7.3 XSL Parameters / XQuery Variables

The XSL/XQuery | XSL Parameters/XQuery Variables command opens the XSLT Input Parameters/XQuery External Variables dialog (see screenshot). You can enter the name of one or more parameters you wish to pass to the XSLT stylesheet, or one or more external XQuery variables you wish to pass to the XQuery document, and their respective values. These parameters are used as follows in XMLSpy:

- When the XSL Transformation command in the XSL/XQuery menu is used to transform an XML document, the parameter values currently saved in the dialog are passed to the selected XSLT document and used for the transformation.

- When the XQuery Execution command in the XSL/XQuery menu is used to process an XQuery document, the XQuery external variable values currently saved in the dialog are passed to the XQuery document for the execution.

**Note:** Parameters or variables that you enter in the XSLT Input Parameters/XQuery External Variables dialog are only passed on to the built-in Altova XSLT engine. Therefore, if you are using MSXML or another external engine that you have configured, these parameters are not passed to this engine.

**Note:** It is not an error if an XSLT parameter or external XQuery variable is defined in the XSLT Input Parameters/XQuery External Variables dialog but is not used in the XSLT/XQuery document or the transformation.

#### Using XSLT Parameters

The value you enter for the parameter can be an XPath expression without quotes or a text string delimited by quotes. If the active document is an XSLT document, the Get from XSL button will be enabled. Clicking this button inserts parameters declared in the XSLT into the dialog together with their default values. This enables you to quickly include declared parameters and then change their default values as required.
Note: Once a set of parameter-values is entered in the dialog, it is used for all subsequent transformations until it is explicitly deleted or the application is restarted. Parameters entered in the dialog are specified at the application-level for that session, and will be passed to the respective XSLT document for every transformation that is carried out via the IDE from that moment onward. This means that:

- parameters are not associated with any particular document
- any parameter entered in the dialog is erased once XMLSpy has been closed.

Usage example for XSLT parameters

We have an XML document that contains the names of countries and their respective capitals:

```xml
<document>
  <countries>
    <country name="USA" capital="Washington DC"/>
    <country name="UK" capital="London"/>
    <country name="France" capital="Paris"/>
    <country name="Russia" capital="Moscow"/>
    <country name="China" capital="Beijing"/>
  </countries>
</document>
```

The following XSLT document will generate an XML document that displays one country from the XML file together with that country’s capital. The country is selected by entering its name as the value of the parameter named country (shown highlighted in yellow below).

```xml
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:param name="country" select="'USA'"/>
  <xsl:template match="countries">
    <xsl:for-each select="country[@name=$country]">
      <country>
        <name><xsl:value-of select="$country"/></name>
        <capital><xsl:value-of select="@capital"/></capital>
      </country>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

When this XSLT document is run on the XML document listed above, the result will be this:
<country><name>USA</name><capital>Washington DC</capital></country>

Now, if in the XSLT Input Parameters/XQuery External Variables dialog you create a parameter named `country` and give it a value (see screenshot above), then this value will be passed to the parameter `country` in the XSLT stylesheet for the transformation. In this way, you can pass different values to different parameters at run time.

**Note:**

- If you use the `XSL:FO Transformation` command (XSL/XQuery | XSL:FO Transformation), then parameters entered in the XSLT Input Parameters/XQuery External Variables dialog are **not** passed to the stylesheet. In order for these parameters to be used in PDF output, first transform from XML to FO using the XSLT Transformation command (XSL/XQuery | XSL Transformation), and then transform the FO to PDF using the `XSL:FO Transformation` command (XSL/XQuery | XSL:FO Transformation).
- If you use an XSLT processor other than the built-in Altova XSLT Engines, parameters you enter using the Input Parameters dialog will not be passed to the external processor.

**Using external XQuery variables**

The value you enter for an external XQuery variable could be an XPath expression without quotes or a text string delimited by quotes. The datatype of the external variable is specified in the variable declaration in the XQuery document.

![XSLT Input Parameters/XQuery External Variables dialog](image)

**Note:** Once a set of external XQuery variables are entered in the dialog, they are used for all subsequent executions until they are explicitly deleted or the application is restarted. Variables entered in the dialog are specified at the application-level, and will be passed to the respective XQuery document for every execution that is carried out via the IDE from that moment onward. This means that:

- Variables are not associated with any particular document
- Any variable entered in the dialog is erased once the application (XMLSpy) has been closed down.
Usage example for external XQuery variables

In the following example, a variable $first is declared in the XQuery document and is then used in the return clause of the FLWOR expression:

```xquery
xquery version "1.0";
declare variable $first as xs:string external;
let $last := "Jones",
return concat($first, " ", $last )
```

This XQuery returns Peter Jones, if the value of the external variable (entered in the XSLT Input Parameters/XQuery External Variables dialog) is Peter. Note the following:

- The external keyword in the variable declaration in the XQuery document indicates that this variable is an external variable.
- Defining the static type of the variable is optional. If a datatype for the variable is not specified in the variable declaration, then the variable value is assigned the type xs:untypedAtomic.
- If an external variable is declared in the XQuery document, but no external variable of that name is passed to the XQuery document, then an error is reported.
- If an external variable is declared and is entered in the XSLT Input Parameters/XQuery External Variables dialog, then it is considered to be in scope for the XQuery document being executed. If a new variable with that name is declared within the XQuery document, the new variable temporarily overrides the in-scope external variable. For example, the XQuery document below returns Paul Jones even though the in-scope external variable $first has a value of Peter.

```xquery
xquery version "1.0";
declare variable $first as xs:string external;
let $first := "Paul"
let $last := "Jones"
return concat($first, " ", $last )
```

23.7.4 XQuery/Update Execution

The XSL/XQuery | XQuery/ Update Execution command executes an XQuery (1.0/3.1) or XQuery Update (1.0/3.0) document. Depending on whether the selected file is an XQuery or XQuery Update file, either an XQuery execution or an XQuery update is carried out. XMLSpy recognizes the type of document (XQuery or XQuery Update) on the basis of the document's file type association (defined in the File types section of the Options dialog).

The XQuery Engine to use (1.0 or 3.1) is selected automatically on the basis of the version declaration in the document. If there is no version declaration in the document, then the default version specified in the XQuery section of the Options dialog is used. The XQuery/ Update Execution command can be invoked when an XQuery, XQuery Update, or XML file is active. When invoked from an XML file, it opens a dialog asking for an XQuery file to associate with the XML file. You can also select a file via a global resource or a URL (click the Browse button) or a file in one of the open windows in XMLSpy (click the Window button).
Note: The command is also available in the context menu of Project window items.

Automating validation with RaptorXML  2019

RaptorXML is Altova's standalone application for XML validation, XSLT transformation, and XQuery transformation. It can be used from the command line, via a COM interface, in Java programs, and in .NET applications. XQuery execution tasks can therefore be automated with the use of RaptorXML. For example, you can create a batch file that calls RaptorXML to run XQuery executions on a set of documents and sends the output to a text file. See the RaptorXML documentation for details.

23.7.5 Assign XSL

The XSL/XQuery | Assign XSL... command assigns an XSLT stylesheet to an XML document. Clicking the command opens a dialog to let you specify the XSLT file you want to assign. You can also select a file via a global resource or a URL (click the Browse button) or a file in one of the open windows in XMLSpy (click the Window button).

An xml-stylesheet processing instruction is inserted in the XML document:

```xml
<?xml-stylesheet type="text/xsl" href="C:\workarea\recursion\recursion.xslt"
>
```

Please note: You can make the path of the assigned file relative by clicking the Make Path Relative To... check box.

23.7.6 Assign XSL-FO

The XSL/XQuery | Assign XSL:FO command assigns an XSLT stylesheet for transformation to FO to an XML document. The command opens a dialog to let you specify the XSL or XSLT file you want to assign and inserts the required processing instruction into your XML document.

You can make the path of the assigned file relative by clicking the Make Path Relative To check box. You can also select a file via a global resource or a URL (click the Browse button) or a file in one of the open windows in XMLSpy (click the Window button).

Please note: An XML document may have two XSLT files assigned to it: one for standard XSLT transformations, a second for an XSLT transformation to FO.

23.7.7 Assign Sample XML File

The XSL/XQuery | Assign Sample XML File command assigns an XML file to an XSLT
document. The command inserts a processing instruction naming an XML file to be processed with this XSLT file when the XSL Transformation is executed on the XSLT file:

```xml
<?altova_samplexml C:\workarea\html2xml\article.xml?>
```

Please note: You can make the path of the assigned file relative by clicking the Make Path Relative To... check box. You can also select a file via a global resource or a URL (click the Browse button) or a file in one of the open windows in XMLSpy (click the Window button).

### 23.7.8 Go to XSL

The XSL/XQuery | Go to XSL command opens the associated XSLT document. If your XML document contains a stylesheet processing instruction (i.e. an XSLT assignment) such as this:

```xml
<?xml-stylesheet type="text/xsl" href="Company.xsl"?>
```

then the Go to XSL command opens the XSLT document in XMLSpy.

### 23.7.9 Start Debugger / Go

The XSL/XQuery | Start Debugger/Go command starts or continues processing the XSLT/XQuery document till the end. If breakpoints have been set, then processing will pause at that point. If tracepoints have been set, output for these statements will be displayed in the Trace window when the closing node of the statement with the tracepoint has been reached. If the debugger session has not been started, then this button will start the session and stop at the first node to be processed. If the session is running, then the XSLT/XQuery document will be processed to the end, or until the next breakpoint is encountered.

### 23.7.10 Stop Debugger

The XSL/XQuery | Stop Debugger command stops the debugger. This is not the same as stopping the debugger session in which the debugger is running. This is convenient if you wish to edit a document in the middle of a debugging session or to use alternative files within the same debugging session. After stopping the debugger, you must restart the debugger to start from the beginning of the XSLT/XQuery document.
23.7.11 Restart Debugger

The XSL/XQuery | Restart Debugger command clears the output window and restarts the debugging session with the currently selected files.

23.7.12 End Debugger Session

The XSL/XQuery | End Debugger Session command ends the debugging session and returns you to the normal XMLSpy view that was active before you started the debugging session. Whether the output documents that were opened for the debugging session stay open depends on a setting you make in the XSLT/XQuery Debugger Settings dialog.

23.7.13 Step Into

F11

The XSL/XQuery | Step Into command proceeds in single steps through all nodes and XPath expressions in the stylesheet. This command is also used to re-start the debugger after it has been stopped.

23.7.14 Step Out

Shift+F11

The XSL/XQuery | Step Out command steps out of the current node to the next sibling of the parent node, or to the next node at the next higher level from that of the parent node.

23.7.15 Step Over

Ctrl+F11

The XSL/XQuery | Step Over command steps over the current node to the next node at the same level, or to the next node at the next higher level from that of the current node. This command is also used to re-start the debugger after it has been stopped.
23.7.16  **Show Current Execution Node**

The **XSL/XQuery | Show Current Execution Node** command displays/selects the current execution node in the XSLT/XQuery document and the corresponding context node in the XML document. This is useful when you have clicked in other tabs which show or mark specific code in the XSLT stylesheet or XML file, and you want to return to where you were before you did this.

23.7.17  **Insert/Remove Breakpoint**

The **XSL/XQuery | Insert/Remove Breakpoint** command inserts or removes a breakpoint at the current cursor position. Inline breakpoints can be defined for nodes in both the XSLT/XQuery and XML documents, and determine where the processing should pause. A dashed red line appears above the node when you set a breakpoint. Breakpoints cannot be defined on closing nodes, and breakpoints on attributes in XSLT documents will be ignored. This command is also available by right-clicking at the breakpoint location.

23.7.18  **Insert/Remove Tracepoint**

The **XSL/XQuery | Insert/Remove Tracepoint** command inserts or removes a tracepoint at the current cursor position in an XSLT/XQuery document. For statements with a tracepoint, during debugging, the value of the statement is displayed in the Trace window when the closing node of that statement is reached. A dashed blue line appears above the node when you set a tracepoint. Tracepoints cannot be defined on closing nodes. This command is also available by right-clicking at the tracepoint location.

23.7.19  **Enable/Disable Breakpoint**

The **XSL/XQuery | Enable/Disable Breakpoint** command enables or disables already defined breakpoints. The red breakpoint highlight turns to gray when the breakpoint is disabled. The debugger does not stop at disabled breakpoints. To disable/enable a breakpoint, place the cursor in that node name and click the **Enable/Disable Breakpoint** command. This command is also available by right-clicking at the location where you want to enable/disable the breakpoint.
23.7.20 Enable/Disable Tracepoint

Ctrl+Shift+F9

The XSL/XQuery | Enable/Disable Tracepoint command enables or disables already defined tracepoints. The blue tracepoint highlight turns to gray when the tracepoint is disabled. No output is displayed for statements with disabled tracepoints. To disable/enable a tracepoint, place the cursor in that node name and click the Enable/Disable Tracepoint command. This command is also available by right-clicking at the location where you want to enable/disable the tracepoint.

23.7.21 Breakpoints/Tracepoints

The XSL/XQuery | Breakpoints/Tracepoints... command opens the XSLT Breakpoints / Tracepoints dialog, which displays a list of all currently defined breakpoints and tracepoints (including disabled breakpoints and tracepoints) in all files in the current debugging session.

The check boxes indicate whether a breakpoint or tracepoint is enabled (checked) or disabled. You can remove the highlighted breakpoint or tracepoint by clicking the corresponding Remove button, and remove all breakpoints by clicking the corresponding Remove All button. The Edit Code button takes you directly to that breakpoint/tracepoint in the file.

Use the down arrow to move the highlighted breakpoint to the Tracepoints pane and the up
To move the highlighted tracepoint to the Breakpoints pane.
In the XPath column in the Tracepoints pane, you can set an XPath for each tracepoint.

23.7.22 Debug Windows

Placing the cursor over the XSL/XQuery | Debug Windows command pops out a submenu with a list of the various Information Windows of the XSLT/XQuery Debugger. Selecting an Information Window from this list shows/hides that Information Window in the XSLT/XQuery Debugger interface. This command can be used to effect only when a debugging session is in progress.

23.7.23 Debug Settings

The XSL/XQuery | Debug Settings command opens the Debug Settings dialog, which enables you to set user options for the Debugger. See the XSLT/XQuery Debugger section for details.
23.8 **Authentic Menu**

Authentic View enables you to edit XML documents based on StyleVision Power Stylesheets (.sps files) created in Altova’s StyleVision product! These stylesheets contain information that enables an XML file to be displayed graphically in Authentic View. In addition to containing display information, StyleVision Power Stylesheets also allow you to write data to the XML file. This data is dynamically processed using all the capability available to XSLT stylesheets and instantly produces the output in Authentic View.

Additionally, StyleVision Power Stylesheets can be created to display an editable XML view of a database. The StyleVision Power Stylesheet contains information for connecting to the database, displaying the data from the database in Authentic View, and writing back to the database.

The **Authentic** menu contains commands relevant to editing XML documents in Authentic View. For a tutorial on Authentic View, see the [Authentic View Tutorials](Authentic View Tutorials) section.
23.8.1 **New Document**

This command enables you to open a new XML document template in Authentic View. The XML document template is based on a StyleVision Power Stylesheet (.sps file), and is opened by selecting the StyleVision Power Stylesheet (SPS file) in the Create New Document dialog (screen shot below). On selecting an SPS and clicking OK, the XML document template defined for that SPS file is opened in Authentic View.

The Create New Document dialog offers a choice of XML document templates that are based on popular DTDs or schemas. Alternatively, you can browse for a custom-made SPS file that has a Template XML File assigned to it. SPS files are created using Altova StyleVision, an application that enables you to design XML document templates based on a DTD or XML Schema. After designing the required SPS in StyleVision, an XML file is assigned (in StyleVision) as a Template XML File to the SPS. The data in this XML file provides the starting data of the new document template that is opened in the Authentic View of XMLSpy.

The new XML document template will therefore have the documentation presentation properties defined in the SPS and the data of the XML file that was selected as the Template XML File. The Authentic View user can now edit the XML document template in a graphical WYSIWYG interface, and save it as an XML document.

23.8.2 **Edit Database Data**

The Authentic | Edit Database Data... command enables you to open an editable view of a database (DB) in Authentic View. All the information about connecting to the DB and how to display the DB and accept changes to it in Authentic View is contained in a StyleVision Power Stylesheet. It is such a DB-based StyleVision Power Stylesheet that you open with the Edit Database Data... command. This sets up a connection to the DB and displays the DB data (through an XML lens) in Authentic View.

Clicking the Edit Database Data... command opens the Edit Database Data dialog.
Browse for the required SPS file, and select it. This connects to the DB and opens an editable view of the DB in Authentic View. The design of the DB view displayed in Authentic View is contained in the StyleVision Power Stylesheet.

Please note: If, with the Edit Database Data... command, you attempt to open a StyleVision Power Stylesheet that is not based on a DB or to open a DB-based StyleVision Power Stylesheet that was created in a version of StyleVision prior to the StyleVision 2005 release, you will receive an error.

Please note: StyleVision Power Stylesheets are created using Altova StyleVision.

23.8.3 Assign a StyleVision Stylesheet

The Assign a StyleVision Stylesheet command assigns a StyleVision Power Stylesheet (SPS) to an XML document to enable the viewing and editing of that XML document in Authentic View. The StyleVision Power Stylesheet that is to be assigned to the XML file must be based on the same schema as that on which the XML file is based.

To assign a StyleVision Power Stylesheet to an XML file:

1. Make the XML file the active file and select the Authentic | Assign a StyleVision Stylesheet... command.
2. The command opens a dialog box in which you specify the StyleVision Power Stylesheet file you wish to assign to the XML.
3. Click OK to insert the required SPS statement into your XML document. Note that you can make the path to the assigned file relative by clicking the Make path relative to... check box. You can also select a file via a global resource or a URL (click the Browse button) or a file in one of the open windows in XMLSpy (click the Window button).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?altova_sps HTML-Orgchart.sps?>
```
In the example above, the StyleVision Power Stylesheet is called `HTML_Orgchart.sps`, and it is located in the same directory as the XML file.

**Note:** Previous versions of Altova products used a processing instruction with a target or name of `xmlspysps`, so a processing instruction would look something like `<?xmlspysps HTML-Orgchart.sps?>`. These older processing instructions are still valid with Authentic View in current versions of Altova products.

### 23.8.4 Edit StyleVision Stylesheet

The **Edit StyleVision Stylesheet** command starts StyleVision and allows you to edit the StyleVision Power Stylesheet immediately in StyleVision. The command is enabled only if a StyleVision Power Stylesheet has been assigned to the XML document.

### 23.8.5 Select New Row with XML Data for Editing

The **Select New Row with XML Data for Editing** command enables you to select a new row from the relevant table in an XML DB, such as IBM DB2. This row appears in Authentic View, can be edited there, and then saved back to the DB.

When an XML DB is used as the XML data source, the XML data that is displayed in Authentic View is the XML document contained in one of the cells of the XML data column. The **Select New Row with XML Data for Editing** command enables you to select an XML document from another cell (or row) of that XML column. Selecting the **Select New Row...** command pops up the Choose XML Field dialog (screenshot below), which displays the table containing the XML column.

![Choose XML field](image)

You can enter a filter for this table. The filter should be an SQL `WHERE` clause (just the condition,
without the WHERE keyword, for example: CID>1002). Click Update to refresh the dialog. In the screenshot above, you can see the result of a filtered view. Next, select the cell containing the required XML document and click OK. The XML document in the selected cell (row) is loaded into Authentic View.

23.8.6 XML Signature

The XML Signature command is available in Authentic View when the associated SPS has XML Signatures enabled. The XML Signature command is also available as the XML Signature toolbar icon in the Authentic toolbar.

Verification and own certificate/password

Clicking the XML Signature command starts the signature verification process. If no signature is present in the document, a message to that effect is displayed in the XML Signature dialog (see screenshot below), and the dialog will have a button that enables the Authentic View user to sign the document.

If the Select Own Certificate or Select Own Password button is present in this dialog, it means that the Authentic View has been given the option of selecting an own certificate/password. (Whether a certificate or password is to be chosen has been decided by the SPS designer at the time the signature was configured. The signature will be either certificate-based or password-based.) Clicking either of these buttons, if present in the dialog, enables the Authentic View user to browse for a certificate or to enter a password. The Authentic View user's selection is stored in memory and is valid for the current session only. If, after selecting a certificate or password, the document or application is closed, the certificate/password setting reverts to the setting originally saved with the SPS.

Verification and authentication information

If the verification process is run on a signed document, two general situations are possible. First: If the authentication information is available (in the signature or the SPS), then the verification process is executed directly and the result is displayed (screenshot below).
Authentication information is either the signing certificate's key information or the signing password. The SPS designer will have specified whether the certificate's key information is saved in the signature when the XML document is signed, or, in the case of a password-based signature, whether the password is saved in the SPS. In either of these cases, the authentication is available. Consequently the verification process will be run directly, without requiring any input from the Authentic View user.

The second possible general situation occurs when authentication information is not available in the signature (certificate's key information) or SPS file (password). In this situation, the Authentic View user will be asked to supply the authentication information: a password (see screenshot below) or the location of a certificate.

23.8.7 Define XML Entities

You can define entities for use in Authentic View, whether your document is based on a DTD or an XML Schema. Once defined, these entities are displayed in the Entities Entry Helper and in the Insert Entity submenu of the context menu. When you double-click on an entity in the Entities Entry Helper, that entity is inserted at the cursor insertion point.

An entity is useful if you will be using a text string, XML fragment, or some other external resource in multiple locations in your document. You define the entity, which is basically a short name that stands in for the required data, in the Define Entities dialog. After defining an entity you can use it at multiple locations in your document. This helps you save time and greatly enhances maintenance.

There are two broad types of entities you can use in your document: a parsed entity, which is XML data (either a text string or a fragment of an XML document), or an unparsed entity, which is non-XML data such as a binary file (usually a graphic, sound, or multimedia object). Each entity
has a name and a value. In the case of parsed entities the entity is a placeholder for the XML data. The value of the entity is either the XML data itself or a URI that points to a .xml file that contains the XML data. In the case of unparsed entities, the value of the entity is a URI that points to the non-XML data file.

To define an entity:

1. Click **Authentic | Define XML Entities...** This opens the Define Entities dialog.

2. Enter the name of your entity in the **Name** field. This is the name that will appear in the Entities Entry Helper.

3. Enter the type of entity from the drop-down list in the **Type** field. Three types are possible. An **Internal** entity is one for which the text to be used is stored in the XML document itself. Selecting **PUBLIC** or **SYSTEM** specifies that the resource is located outside the XML file, and will be located with the use of a public identifier or a system identifier, respectively. A system identifier is a URI that gives the location of the resource. A public identifier is a location-independent identifier, which enables some processors to identify the resource. If you specify both a public and system identifier, the public identifier resolves to the system identifier, and the system identifier is used.

4. If you have selected **PUBLIC** as the Type, enter the public identifier of your resource in the **PUBLIC** field. If you have selected **Internal** or **SYSTEM** as your Type, the **PUBLIC** field is disabled.

5. In the **Value/Path** field, you can enter any one of the following:

   - If the entity type is Internal, enter the text string you want as the value of your entity. Do not enter quotes to delimit the entry. Any quotes that you enter will be treated as part of the text string.
   - If the entity type is System, enter the URI of the resource or select a resource on your local network by using the **Browse** button. If the resource contains parsed data, it must be an XML file (i.e. it must have a .xml extension). Alternatively, the resource can be a binary file, such as a GIF file.
   - If the entity type is PUBLIC, you must additionally enter a system identifier in this field.

6. The **NDATA** entry tells the processor that this entity is not to be parsed but to be sent to the appropriate processor. The **NDATA** field should therefore be used with unparsed entities only.
Dialog features
You can append, insert, and delete entities by clicking the appropriate buttons. You can also sort entities on the alphabetical value of any column by clicking the column header; clicking once sorts in ascending order, twice in descending order. You can also resize the dialog box and the width of columns.

Once an entity is used in the XML document, it is locked and cannot be edited in the Define Entities dialog. Locked entities are indicated by a lock symbol in the first column. Locking an entity ensures that the XML document valid with respect to entities. (The document would be invalid if an entity is referenced but not defined.)

Duplicate entities are flagged.

Limitations

- An entity contained within another entity is not resolved, either in the dialog, Authentic View, or XSLT output, and the ampersand character of such an entity is displayed in its escaped form, i.e. \&amp;.
- External entities are not resolved in Authentic View, except in the case where an entity is an image file and it is entered as the value of an attribute which has been defined in the schema as being of type ENTITY or ENTITIES. Such entities are resolved when the document is processed with an XSLT generated from the SPS.

23.8.8 View Markup
The View Markup command has a submenu with options to control markup in the Authentic XML document. These options are described below.

The Hide Markup command hides markup symbols in Authentic View.

The Show Small Markup command shows small markup symbols in Authentic View.

The Show Large Markup command shows large markup symbols in Authentic View.

The Show Mixed Markup command shows mixed markup symbols in Authentic View. The person who designs the StyleVision Power Stylesheet can specify either large markup, small
markup, or no markup for individual elements/attributes in the document. The Authentic View user sees this customized markup in mixed markup viewing mode.

### 23.8.9 RichEdit

Mousing over the **RichEdit** command pops out a submenu containing the RichEdit markup commands (*screenshot below*). The menu commands in this submenu are enabled only in Authentic View and when the cursor is placed inside an element that has been created as a RichEdit component in the SPS design.

- **B**: Toggle Bold
- **I**: Toggle Italic
- **U**: Toggle Underline
- **S**: Toggle Strikethrough
- **Foreground Color**: Set foreground color
- **Background Color**: Set background color
- **Align Left**
- **Center**
- **Align Right**

The text-styling properties of the RichEdit menu will be applied to the selected text when a RichEdit markup command is clicked. The Authentic View user can, in addition to the font and font-size specified in the Authentic toolbar, additionally specify the font-weight, font-style, font-decoration, color, background color and alignment of the selected text.

### 23.8.10 Append/Insert/Duplicate/Delete Row

- **Append Row**: Appends a row to the current table in Authentic View.

- **Insert Row**: Inserts a row into the current table in Authentic View.

- **Duplicate Row**: Duplicates the current table row in Authentic View.

- **Delete Row**: Deletes the current table row in Authentic View.
23.8.11 Move Row Up/Down

The Move Row Up command moves the current table row up by one row in Authentic View.

The Move Row Down command moves the current table row down by one row in Authentic View.

23.8.12 Generate HTML, RTF, PDF, Word 2007+ Document

These four commands generate output documents from the Authentic View XML document stored in a PXF file:

- Generate an HTML Document
- Generate an RTF Document
- Generate a PDF Document
- Generate a Word 2007+ Document

They are also available in the Portable XML Form (PXF) toolbar (screenshot below).

Clicking the individual command or buttons generates HTML, RTF, PDF, or DocX output, respectively.

These buttons are enabled when a PXF file is opened in Authentic View. Individual commands and buttons are enabled if the PXF file was configured to contain the XSLT stylesheets for that specific output format. For example, if the PXF file was configured to contain the XSLT stylesheets for HTML and RTF, then only the commands and toolbar buttons for HTML and RTF output will be enabled while those for PDF and DocX (Word 2007+) output will be disabled.

23.8.13 Trusted Locations

The Trusted Locations command pops up the Trusted Locations dialog (screenshot below), in which you can specify the security settings for scripts in an SPS. When an XML file based on a script-containing SPS is switched to Authentic View, the script will be allowed to run or not depending on the settings you make in this dialog.
The three available options are:

- Authentic scripts are always run when a file is opened in Authentic View.
- Authentic scripts are never run when a file is opened in Authentic View.
- Only Authentic scripts in trusted locations are run. The list of trusted (folder) locations is shown in the bottom pane. Use the Add button to browse for a folder and add it to the list. To remove an entry from the list, select an entry in the Trusted Locations list and click Remove.
23.9 **DB Menu**

The **DB** menu is the menu for database (DB) operations. It is shown in the screenshot below and contains the menu items listed below. Descriptions of commands in the sub-menus of the DB menu are in the sub-sections of this section.

![DB Menu Screenshot](image)

- **Query Database**, which enables you to query a variety of databases.
- **IBM DB2**, which contains commands that provide support for IBM DB2-specific functionality.
- **SQL Server**, which contains commands for managing SQL Server databases.

The operations described in this section require a connection to a database (for instructions, see [Connecting to a Database](#)).

### 23.9.1 Query Database

The **Query Database** command opens the Database Query window *(screenshot below)*. Once the Query Window is open, its display can be toggled on and off by clicking either the **DB | Query Database** command or the Query Database toolbar icon.

![Query Database Screenshot](image)
Overview of the Database Query window

The Database Query window consists of three parts:

- A Browser pane at top left, which displays connection info and database tables.
- A Query pane at top right, in which the query is entered.
- A tabbed Results/Messages pane. The Results pane displays the query results in what we call the Result Grid. The Messages pane displays messages about the query execution, including warnings and errors.

The Database Query window has a toolbar at the top. At this point, take note of the two toolbar icons below. The other toolbar icons are described in the section, Query Pane: Description and Features.

Toggle the Browser pane on and off.

Toggle the Results/Messages pane on and off.
Overview of the Query Database mechanism

The Query Database mechanism is as follows. It is described in detail in the sub-sections of this section.

1. A connection to the database is established via the Database Query window. Supported databases include: MS Access 2000 and 2003; Microsoft SQL Server; Oracle; MySQL; Sybase; and IBM DB2.
2. The connected database or parts of it are displayed in the Browser pane, which can be configured to suit viewing requirements.
3. A query written in a syntax appropriate to the database to be queried is entered in the Query pane, and the query is executed.
4. The results of the query can be viewed through various filters, edited, and saved back to the DB.

Data Sources

In order to query a database, you have to first connect to the required database. This section describes how to:

- Connect to a database, and
- Select the required data source and root object from among multiple existing connections.

Connecting to a database

When you click the Query Database command for the first time in a session (or when no database connection exists), the Quick Connect dialog (screenshot below) pops up to enable you to connect to a database. To make connections subsequently, click the Quick Connect icon in the Database Query window.
How to connect to a database via the Quick Connect dialog is described in the section [Connecting to a Data Source](#).

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></td>
</tr>
<tr>
<td>MariaDB 10.2, 10.3</td>
<td></td>
</tr>
</tbody>
</table>
## Database Connections

<table>
<thead>
<tr>
<th>Database</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<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. In Authentic view, data coming from a SQLite database is not editable. When you attempt to save SQLite data from the Authentic view, a message box will inform you of this known limitation.</td>
</tr>
<tr>
<td>Sybase ASE 15, 16</td>
<td></td>
</tr>
<tr>
<td>Teradata 16</td>
<td></td>
</tr>
</tbody>
</table>

### Selecting the required data source

All the existing connections and the root objects of each are listed, respectively, in two combo boxes in the toolbar of the Database Query window (screenshot below).

In the screenshot above, the database with the name **StyleVision DB** has been selected. Of the
available root objects for this database, the root object `ALTOVA_USER` has been selected. The database and the root object are then displayed in the Browser pane.

Browser Pane: Viewing the DB Objects

The Browser pane provides an overview of objects in the selected database. This overview includes database constraint information, such as whether a column is a primary or foreign key. In IBM DB2 version 9 databases, the Browser additionally shows registered XML schemas in a separate folder.

This section describes the following:

- The **layouts** available in the Browser pane.
- **How to filter** database objects.
- **How to find** database objects.

Browser pane layouts

The default Folders layout displays database objects hierarchically. Depending on the selected object, different context menu options are available when you right-click an item.

To select a layout for the Browser, click the Layout icon in the toolbar of the Browser pane and select the layout from the drop-down list (screenshot below). Note that the icon changes with the selected layout.
The available layouts are:

- **Folders**: Organizes database objects into folders based on object type in a hierarchical tree, this is the default setting.
- **No Schemas**: Similar to the Folders layout, except that there are no database schema folders; tables are therefore not categorized by database schema.
- **No Folders**: Displays database objects in a hierarchy without using folders.
- **Flat**: 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.
- **Table Dependencies**: 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.

To sort tables into User and System tables, switch to Folders, No Schemas or Flat layout, then right-click the Tables folder and select **Sort into User and System Tables**. The tables are sorted alphabetically in the User Tables and System Tables folders.

**Filtering database objects**

In the Browser pane (in all layouts except No Folders and Table Dependencies), schemas, tables, and views can be filtered by name or part of a name. Objects are filtered as you type in the characters, and filtering is case-insensitive by default.

To filter objects in the Browser, do the following:

1. Click the Filter Folder Contents icon in the toolbar of the Browser pane. Filter icons appear next to the Tables and Views folders in the currently selected layout (**screenshot below**).
2. Click the filter icon next to the folder you want to filter, and select the filtering option from the popup menu, for example, Contains.

3. In the entry field that appears, enter the filter string (in the screenshot below, the filter string on the Tables folder is NHE). The filter is applied as you type.

Finding database objects
To find a specific database item by its name, you can use the Browser pane's Object Locator. This works as follows:
1. In the toolbar of the Browser pane, click the Object Locator icon. A drop-down list appears at the bottom of the Browser.

2. Enter the search string in the entry field of this list, for example `name` (screenshot below). Clicking the drop-down arrow displays all objects that contain the search string.

3. Click the object in the list to see it in the Browser.

**Query Pane: Description and Features**

The Query pane is an intelligent SQL editor for entering queries to the selected database. After entering the query, clicking the Execute command of the Database Query window executes the query and displays the result and execution messages in the Results/Messages pane. How to work with queries is described in the next section, *Query Pane: Working with Queries*. In this section, we describe the main features of the Query pane:

- SQL Editor icons in the Database Query toolbar
- SQL Editor options
- Auto-completion of SQL statements
- Definition of regions in an SQL script
- Insertion of comments in an SQL script
- Use of bookmarks

**SQL Editor icons in the Database Query toolbar**

The following icons in the toolbar of the Database Query window are used when working with the SQL Editor:

- **Execute**

  Executes currently selected SQL statement. If script contains multiple statements and none is selected, then all are executed.
**Execute with Data Editing**
Same as for Execute command, except that results (in Results tab) are editable.

**Import SQL File**
Opens an SQL file in the SQL Editor.

**Export SQL File**
Saves SQL queries to an SQL file.

**Undo**
Undoes an unlimited number of edits in SQL Editor.

**Redo**
Redoes an unlimited number of edits in SQL Editor.

**Hide DB Query on XML Open**
Sets whether the DB Query window should be hidden when an XML document is opened for editing.

**Auto-Commit on XML Save**
When an edited XML document is saved in XMLSpy, changes are committed to the DB if this toggle is on. Otherwise, changes have to be explicitly committed in the Results Pane.

**Options**
Open the Options dialog of SQL Editor.

**Open SQL Script in DatabaseSpy**
Opens the SQL script in Altova’s DatabaseSpy product.

---

**Options**
Clicking the **Options** icon in the Database Query toolbar pops up the Options dialog (screenshot below). A page of settings can be selected in the left-hand pane, and the options on that page can be selected. Click the **Reset to Page Defaults** button to reset the options on that page to their original settings.

---

The key settings are as follows:
- **General | Encoding**: Options for setting the encoding of new SQL files, of existing SQL files for which the encoding cannot be detected, and for setting the Byte Order Mark (BOM). (If the encoding of existing SQL files can be detected, the files are opened and saved without changing the encoding.)

- **SQL Editor**: Options for toggling syntax coloring and data source connections on execution on/off. A timeout can be set for query execution, and a dialog to change the timeout can also be shown if the specified time is exceeded. Entry helpers refer to the entry helpers that appear as part of the auto-completion feature. When you type in an SQL statement, the editor displays a list of context-sensitive auto-completion suggestions. These suggestions can be set to appear automatically. If the automatic display is switched off, then you can ask for an auto-completion suggestion in SQL Editor by pressing Ctrl+Spacebar. The buffer for the entry helper information can be filled either on connection to the data source or the first time it is needed. The Text View options window of XMLSpy.

- **SQL Editor | SQL Generation**: The application generates SQL statements when you drag objects from the Browser pane into the Query pane. Options for SQL statement generation can be set in the SQL generation tab. Use the Database pane to select a database kind and set the statement generation options individually for the different database kinds you are working with. Activating the Apply to all databases check box sets the options that are currently selected for all databases. Options include appending semi-colons to statements and surrounding identifiers with escape characters. When the Append semicolons to statement end check box is activated, a semicolon is appended when you generate an SQL statement in the SQL Editor. Note that editing of data in Oracle databases and IBM iSeries and DB2 databases via a JDBC connection is possible only if this check box is unchecked.

- **SQL Editor | Result View**: Options to configure the Result tab.

- **SQL Editor | Fonts**: Options for setting the font style of the text in the Text Editor and in the Result View.

**Definition of regions in an SQL script**

Regions are sections in SQL scripts that are marked and declared to be a unit. Regions can be collapsed and expanded to hide or display parts of the script. It is also possible to nest regions within other regions. Regions are delimited by --region and --endregion comments, respectively, before and after the region. Regions can optionally be given a name, which is entered after the -- region delimiter (see screenshot below).

To insert a region, select the statement/s to be made into a region, right-click, and select Insert Region. The expandable/collapsible region is created. Add a name if you wish. In the screenshot above, also notice the line-numbering. To remove a region, delete the two --region and --endregion delimiters.
Insertion of comments in an SQL script

Text in an SQL script can be commented out. These portions of the script are skipped when the script is executed.

- To comment out a block, mark the block, right-click, and select **Insert/Remove Block Comment**. To remove the block comment, mark the comment, right-click and select **Insert/Remove Block Comment**.
- To comment out a line or part of a line, place the cursor at the point where the line comment should start, right-click, and select **Insert/Remove Line Comment**. To remove the line comment, mark the comment, right-click and select **Insert/Remove Line Comment**.

Use of bookmarks

Bookmarks can be inserted at specific lines, and you can then navigate through the bookmarks in the document. To insert a bookmark, place the cursor in the line to be bookmarked, right-click, and select **Insert/Remove Bookmark**. To go to the next or previous bookmark, right-click, and select **Go to Next Bookmark** or **Go to Previous Bookmark**, respectively. To remove a bookmark, place the cursor in the line for which the bookmark is to be removed, right-click, and select **Insert/Remove Bookmark**. To remove all bookmarks, right-click, and select **Remove All Bookmarks**.

Query Pane: Working with Queries

After connecting to a database, an SQL script can be entered in the SQL Editor and executed. This section describes:

- How an SQL script is entered in the SQL Editor.
- How the script is executed in the Database Query window.

The following icons are referred to in this section:

- **Execute Query** Executes currently selected SQL statement. If script contains multiple statements and none is selected, then all are executed.
- **Execute for Data Editing** Same as for Execute command, except that results (in Results tab) are editable.
- **Import SQL File** Opens an SQL file in the SQL Editor.

Creating SQL statements and scripts in the SQL Editor

The following GUI methods can be used to create SQL statements or scripts:

- **Drag and drop**: Drag an object from the Browser pane into the SQL Editor. An SQL statement is generated to query the database for that object.
- **Context menu**: Right-click an object in the Browser pane and select **Show in SQL Editor | Select**.
- **Manual entry**: Type SQL statements directly in SQL Editor. The Auto-completion feature can help with editing.
Import an SQL script: Click the Import SQL File icon in the toolbar of the Database Query window.

Executing SQL statements
If the SQL script in the SQL Editor has more than one SQL statement, select the statement to execute and click either the Execute icon or Execute with Data Editing icon in the toolbar of the Database Query window. If no statement in the SQL script is selected, then all the statements in the script are executed. The database data is retrieved and displayed as a grid in the Results tab. If Execute with Data Editing was selected, then the retrieved data in the Result Grid can be edited. Messages about the execution are displayed in the Messages tab.

Results and Messages
The Results/Messages pane has two tabs:

- The Results tab shows the data that is retrieved by the query.
- The Messages tab shows messages about the query execution.

Results tab
The data retrieved by the query is displayed in the form of a grid in the Results tab (screenshot below). When the query results have been generated using the Execute Query command, the XML documents in the Results tab are indicated with the XML icon (screenshot below). If the Execute for Data Editing command was used, XML documents are shown with the Editable XML icon.

The following operations can be carried out in the Results tab, via the context menu that pops up when you right-click in the appropriate location in the Results tab:

- Sorting on a column: Right-click anywhere in the column on which the records are to be sorted, then select Sorting | Ascending/Descending/Restore Default.
- Copying to the clipboard: This consists of two steps: (i) selecting the data range; and (ii) copying the selection. Data can be selected in several ways: (i) by clicking a column header or row number to select the column or row, respectively; (ii) selecting individual cells (use the Shift and/or Ctrl keys to select multiple cells); (iii) right-clicking a cell, and selecting Selection | Row/Column/All. After making the selection, right-click, and...
select **Copy Selected Cells**. This copies the selection to the clipboard, from where it can be pasted into another application.

- **Appending a new row**: If the query was executed for editing, right-click anywhere in the Results pane to access the **Append row** command.
- **Deleting a row**: If the query was executed for editing, right-click anywhere in a row to access the **Delete row** command.
- **Editing records**: If the query was executed for editing, individual fields can be edited. To commit changes, click the **Commit** button in the toolbar of the Results tab.
- **Editing XML records**: This feature is supported for IBM DB2, SQL Server, PostgreSQL (8.3 and higher), and Oracle (9 and higher) databases, and only for those DB tables that have a primary key. If the query was executed for editing and an editable field is an XML field, clicking the **Editable XML icon** in the Result Grid opens the Edit XML menu (screenshot below). An XML field can also be opened for data editing by right-clicking the XML field in the Folders pane and selecting the command **Edit Data**.

| Open for editing | Load XML Document from File... | Save XML Document to File... | Assign XML Schema... |

The **Open for Editing** command opens the XML document in an XMLSpy window, and the **Editable XML icon** changes to ![edit](img/edit.png), in which the three dots are red. When this document is saved and if the **Auto-Commit XML Changes icon** in the Query Database toolbar was selected when the document was opened, the changes to the XML document are committed automatically to the database. Otherwise, saved changes will have to be committed using the Commit button of the Results pane. (Note that to toggle between the XML document window and the Database Query window, you must click the **DB | Query Database** command.) The **Load XML Document from File** command loads an external XML document to the selected field in the database. The **Save XML Document to File** saves the XML document in the selected database field to a file location you choose. The **Assign XML Schema** command pops up the **Choose XML Schema dialog**, in which you can select an XML Schema to assign to the XML document. This assignment is saved to the database. XML Schema assignment is explained in more detail in the section, **IBM DB2 | Assign XML Schema**.

- **Set NULL, Set default, Undo changes for this cell**: If the query was executed for editing, right-clicking in a cell provides access to commands that enable you to set a **NULL** value or, if defined, a column default value for that cell. Changes made to a cell can be undone with the **Undo changes for this cell** command; the current edited value is replaced by the value currently in the DB.

The **Results tab** has the following toolbar icons:

- ![statement](img/statement.png) **Go to Statement** Highlights the statement in the SQL Editor that produced the current result.
- ![find](img/find.png) **Find** Finds text in the Results pane. XML document content is also searched.
- ![add](img/add.png) **Add New Line** Adds a new row to the Result Grid.
- ![delete](img/delete.png) **Delete Row** Deletes the current row in the Result Grid.
Menu Commands

Undo Changes to Result Grid

Undoes all changes to the Result Grid.

Commit

Commits changes made in the Result Grid to the database.

Messages tab

The Messages tab provides information on the previously executed SQL statement and reports errors or warning messages.

![Messages tab screenshot]

The toolbar of the Messages tab contains icons that enable you to customize the view, navigate it, and copy messages to the clipboard. The Filter icon enables the display of particular types of messages to be toggled on or off. The Next and Previous icons move the selection down and up the list, respectively. Messages can also be copied with or without their child components to the clipboard, enabling them to be pasted in documents. The Find function enables you to specify a search term and then search up or down the listing for this term. Finally, the Clear icon clears the contents of the Messages pane.

Note: These toolbar icon commands are also available as context menu commands.

23.9.2 IBM DB2

The IBM DB2 menu item rolls out a submenu containing commands (i) to register and unregister schemas with an IBM DB2 database (Manage XML Schemas), and (ii) to assign schemas for XML file validation (Assign XML Schema).

Both these mechanisms require that you connect to the required IBM DB2 database. For a connection example, see Connecting to IBM DB2 (ODBC). In this section the focus is on how to manage schemas in an IBM DB2 database and how to assign XML Schemas to a DB XML file.

Note: The Result Grid of the Database Query window provides important functionality for working with XML files in IBM DB2 databases. This functionality includes the ability to open files for editing, loading XML files into a DB XML files, saving DB XML files externally, and assigning XML Schemas to DB XML files.
Manage XML Schemas

The Manage XML Schemas feature enables schemas to be added to and dropped from individual database schemas in an IBM DB2 database. To manage schemas, you have to do the following:

- Connect to the IBM DB2 database
- Select the database schema for which XML Schemas need to be added or dropped
- Carry out the schema management actions.

These steps are described in detail below.

Connecting to the IBM DB2 database

Clicking the Manage XML Schemas command pops up the XML Schema Management for Databases dialog (screenshot below).

The first thing to do if there is no connection to the required database is to connect to it. If a connection already exists, it appears in the Database combo box. To start the connection process, click the Quick Connect icon in the dialog. This pops up the Quick Connect dialog, through which you can make the connection to the database (for instructions, see Connecting to a Database).

Displaying the list of XML Schemas

After the connection to the IBM DB2 database has been established, the database is listed in the combo box at left (see screenshot below). If more than one connection is currently open, you can select the required database in this combo box. In the screenshot below, the StyleVision DB database is selected.
The combo box at right lists all the database schemas of the currently selected IBM DB2 database. When a database schema is selected in this combo box, all the XML Schemas registered for the selected database schema are displayed in the main pane. In the screenshot above, all the XML Schemas registered with the Altova_User database schema are listed, together with their locations. Checking the Show Details check box causes additional information columns to be displayed in the main pane.

Managing the XML Schemas
The list of schemas in the main pane represents the schemas registered for the selected database schema. After the list of XML Schemas is displayed, you can add schemas to the list or drop (delete) schemas from the list.

To add a schema, click the Add button, browse for the required schema file, and select it. The selected schema file is added to the list in the main pane. Clicking the Commit Changes button registers the newly added schema with the database schema.

To drop a schema, select the schema in the main pane and click the Drop Schema button. A Drop Flag is assigned to the schema, indicating that it is scheduled for dropping when changes are next committed. The Drop Flag can be removed by selecting the flagged schema and clicking the Remove Drop Flag button. When the Commit Changes button is clicked, all schemas that have been flagged for dropping will be unregistered from the database schema.

Clicking the View Schema button opens the schema in XMLSpy. To close the XML Schema Management dialog, click the Close button.

Reports
When the Commit Changes button is clicked, the database is modified according to the changes you have made. A report of the Commit action is displayed in the Report pane (screenshot below), enabling you to evaluate whether the success of the action and to debug possible errors. Each subsequent report is displayed below the previous report.
The report pane has a toolbar containing icons that enable you to customize the display of the report listing, navigate the listing, copy report messages, search for text, and clear the pane (see screenshot below).

The **Filter** icon enables the display of particular types of messages to be toggled on or off. The **Next** and **Previous** icons move the selection down and up the list, respectively. Messages can also be copied with or without their child components to the clipboard, enabling them to be pasted in documents. The **Find** function enables you to specify a search term and then search up or down the listing for this term. Finally, the **Clear** icon clears the contents of the Report pane.
**Assign XML Schema**

The Assign XML Schema assigns a schema to an XML file opened for editing via the Result Grid of the Database Query window. After the assignment is made, the XML file can be validated against the assigned schema. The assignment is written to the DB when the XML file is saved in XMLSpy.

**Opening a DB XML file for editing**

In the Database Query window, when a query is addressed to an XML DB and the query is executed for data editing, the Result Grid at the bottom of the Database Query window provides access to the XML files in the database so these can be edited (see screenshot below).

Clicking the XML icon \ icon\ pops up the following menu.

- Open for editing
- Load XML Document from File...
- Save XML Document to File...
- Assign XML Schema...

Selecting the **Open for Editing** command opens the XML document in XMLSpy, where it can be edited.
Assigning a schema to the DB XML file

It is when the DB XML file is opened for editing in XMLSpy that the IBM DB2 | Assign XML Schema command is enabled. With the XML document active in XMLSpy, clicking the Assign XML Schema command pops up the Choose XML Schema dialog (screenshot below).

A schema can be selected from among those stored in the database (these are listed in the dropdown list of the Schema from Database combo box), or from among external files that can be browsed. Clicking OK assigns the schema to the XML file. Note that the assignment is not written into the XML file. When the XML file is saved in XMLSpy—and if the Auto-Commit XML changes icon in the Query Database toolbar was selected when the document was opened—then the schema assignment is saved to the database. Note that the schema assignment is written to the database—and not to the XML file.

Note: The Edit XML menu in the Result Grid of the Database Query window also has an Assign XML Schema command (see screenshot below), which also assigns a schema to the DB XML file.

The difference between the two Assign XML Schema commands is that the command in the DB | IBM DB2 menu enables you to assign an XML Schema while you are editing the XML file thereby allowing you to change schema assignments while editing the XML document and to validate the XML document immediately.

23.9.3 SQL Server

The SQL Server menu item rolls out a submenu containing the Manage XML Schemas command.
Manage XML Schemas

XML Schema management for databases enables you to add and delete XML Schemas from the schema repository of an XML database. After connecting to the database, XMLSpy provides the XML Schema Management for Databases dialog, in which XML Schemas can be managed.

The dialog box provides a Quick Connect icon which calls the Quick Connect wizard to connect to a data source. If more than one connection currently exists, the required connection can be selected from the combo box on the to left-hand side. The required root object can then be selected from the right-hand side combo box. All the XML Schemas currently in the repository for that root object are displayed in the dialog box. The name, location, and namespace of each schema are listed, as well as the option for decomposition.

Note that the stored schemas can also be viewed in the Database Query window (screenshot below), but they cannot be managed there. To manage schemas, use the XML Schema Management for Databases dialog.
In the XML Schema Management dialog you can do the following:

- Add a schema using the Add Schema button. The selected schema will be appended to the list and marked for addition.
- Mark schemas in the list for deletion with the Drop Schema button. The Drop flag can be removed with the Remove Drop Flag button.
- Open a selected schema in Schema View by clicking the View Schema button.
- Commit the addition and drop (deletion) changes with the Commit Changes button.

After changes have been committed, a report of the commit action can be viewed in the Report tab (screenshot below).
23.9.4 Oracle XML DB

XMLSpy allows you to connect to and query Oracle XML Db databases.

The following database functions are supported:

- Add (and register) an XML schema to the Oracle XML Db. The Oracle XML DB client must be installed for you to be able to register XML schemas through XMLSpy.
- Open and delete schemas
- Query the database using XPath statements (DBUri)
- Browse XML documents (using WebDAV)
- Create an XML document based on a schema saved in the database

General installation process:
- Download and install XMLSpy
- Install Oracle server (if necessary)
- Create an Oracle database

Manage XML Schemas

XML Schema management for databases enables you to add and delete XML Schemas from the schema repository of an XML database. After connecting to the database, XMLSpy provides the XML Schema Management for Databases dialog, in which XML Schemas can be managed.
The dialog box provides a Quick Connect icon which calls the Quick Connect wizard to connect to a data source. If more than one connection currently exists, the required connection can be selected from the combo box on the left-hand side. The required root object can then be selected from the right-hand side combo box. All the XML Schemas currently in the repository for that root object are displayed in the dialog box. The name, location, and namespace of each schema are listed, as well as the option for decomposition.

Note that the stored schemas can also be viewed in the Database Query window (screenshot below), but they cannot be managed there. To manage schemas, use the XML Schema Management for Databases dialog.
In the XML Schema Management dialog you can do the following:

- Add a schema using the **Add Schema** button. The selected schema will be appended to the list and marked for addition.
- Mark schemas in the list for deletion with the **Drop Schema** button. The Drop flag can be removed with the **Remove Drop Flag** button.
- Open a selected schema in Schema View by clicking the **View Schema** button.
- Commit the addition and drop (deletion) changes with the **Commit Changes** button.

After changes have been committed, a report of the commit action can be viewed in the Report tab (screenshot below).
Browse Oracle XML Documents

This command allows you to browse the XML documents available on your server. The server details are automatically filled in if you previously queried the database or listed schemas. If this is not the case, then you have to enter them manually.
Use the tree view to find specific XML files. Double clicking a file in the tree view opens it. You can also click a file and click **Open** to achieve the same thing. The **New Folder** button adds a new folder, the **Delete** button deletes the currently selected XML file.
23.10 Convert Menu

The Convert menu (screenshot below) provides powerful data exchange functionality between data formats:

- Import and export text, word processor, database, and XML files.
- Import database data based on an existing XML Schema.
- Create an XML Schema based on the structure of an existing database.
- Create a database structure based on an existing XML schema.
- Convert between XML instances and JSON instances, and between XML Schemas and JSON Schemas.

23.10.1 Import Text File

This command lets you import any structured text file into XMLSpy and convert it to XML format immediately. This is useful when you want to import legacy data from older systems. The steps for importing data in a text file as an XML document are described below.

1. Select the menu item Convert | Import Text File. The following dialog appears:
2. Select one of the following:
   - Map EDI, CSV, or fixed-width text data into XML (you must have installed Altova MapForce in order to select this option)
   - Convert CSV text file into XML

3. Click **OK**. The Text import dialog appears (screenshot below). How to use this dialog is described below.
Path

Enter the path to the file to import in the Path text box, or select the file using the Browse button to the right of the text box. After the file is selected, a Grid View preview of the XML file is displayed in the Preview pane. Any change in the options selected in this dialog will be reflected in the preview immediately.

Delimiter

To successfully import a text file, you need to specify the field delimiter that is used to separate columns or fields within the file. XMLSpy will auto-detect common row separators (CR, LF, or CR+LF).

String quotes

Text files exported from legacy systems sometimes enclose textual values in quotes to better distinguish them from numeric values. If this is the case, you can specify what kind of quotes are being used in your file, and remove them automatically when the data is imported.
Encoding
The data is converted into Unicode (the basis of all XML documents), so you need to specify which character-set the file is currently encoded in. For US or Western European Windows systems this will most likely be Codepage 1252, also referred to as the ANSI encoding.

Byte order
If you are importing 16-bit or 32-bit Unicode (UCS-2, UTF-16, or UCS-4) files, you can also switch between little-endian and big-endian byte order.

First row contains column names
It is also very common for text files to contain the field names in the first row within the file. If this is the case, check this check box.

Preview
In the Preview pane you can rename column headers by clicking in a name and editing it. The column headers will be the element or attribute names in the XML document. You can also select whether a column should be an element or an attribute in the XML document, or whether it should not be imported into the XML document. Click the column-type icon in each column header to toggle through these options. In the screenshot above, Col1 is an element, Col2 is an attribute, and Col3 will not be imported.

When you are satisfied with the options, click Import. The imported data is converted into an XML document that is displayed in Grid View.

23.10.2 Import Database Data

The Import Database Data command enables you to import data from any of a variety of databases into an XML file. The import mechanism involves two steps:

1. A connection to the database is established. For instructions, see Connecting to a Database.
2. The data to be imported is selected.

To import database data, do the following:

1. When you click the Import Database Data command, the Import Database Data dialog (screenshot below) pops up.
2. Select Convert Database Data into XML and click OK. The Connect to a Data Source dialog appears. (The Map Database Data into XML Based on Existing DTD/Schema option requires the use of Altova MapForce to carry out the mapping.)

3. In the Connect to Data Source dialog, you establish a connection to the database. For instructions, see Connecting to a Database.

4. After the connection to the database is established, the Import Database Data dialog displays tabs and windows that enable you to select the database data to import. These options are described below. After finishing, click the Import button to import the database data.

Data selection and import options
The Import Database Data dialog for setting the selection and import options consists of two parts (shown separately in the screenshots below):

- an upper part with two tabs: (i) Selection, and (ii) Options.
- a lower part, which is a Preview window showing the data according to the data selection and import options.
Selection tab

In the Selection tab (screenshot above), the Source pane (screenshot below) displays either a representation of the tables of the database or an editable SQL statement for selecting the required tables, each view being selected by clicking the respective radio button.
In the Table view, you can select the tables in the database that you want to import by checking the table's check box (screenshot above). The contents of the table can then be displayed in the Preview pane. The table selection can be further filtered in the Preview pane (see below).

The database structure can be displayed differently and can be filtered. The Layout icon in the Source pane enables you to organize database objects into: (i) folders based on object type; (ii) folders based on object type, but without schema folders; (iii) in a hierarchy, but without folders; and (iv) categories of tables, based on their relationships with other tables.

Clicking the Filter Folder Contents icon applies a filter to the selected folder (in the screenshot below, to the Tables folder). Clicking the filter icon on the table pops out a menu with a list of filter possibilities. In the screenshot below, the filter has been set to display objects that contain the text "Sys" in its name. The view is filtered accordingly.

Clicking the Object Locator icon pops up a text field, which behaves like a Search entry field. You can enter a text string and the dropdown list will display all the objects whose names contain that text string. Selecting one of these objects from the dropdown list will highlight that object in the tree.

Options tab

In the Options tab (screenshot below), you can specify how number, date, and time values are to be imported; whether data is imported as elements or attributes; and whether comments and NULL fields are to be included in the import.
When NULL fields are enabled for import, you can enter a substitution XML value for them.

**Preview pane**

The Preview pane (screenshot below) displays the structure of the table currently selected in the Selection tab. When a new table is selected in the Selection tab, click the **Preview** button in the Preview pane to display the table. Click the **Reload** button to refresh the preview.

A field can be specified to be imported as an element or attribute, or not to be imported, by clicking the symbol to the left of the column name. You can click through the element, attribute, and ignore options. In the screenshot above, the **city** field, for example, has been set to be imported as an element while the **street** field has been set to be imported as an attribute.

**Datatype conversions**

Information about the conversion of database datatypes to XML Schema datatypes is listed in the **Appendices**.
23.10.3 Import Microsoft Word Document

This command enables the direct import of any Word document and conversion into XML format if you have been using paragraph styles in Microsoft Word. This option requires Microsoft Word or Microsoft Office (Version 97 or 2000).

When you select this command, the Open dialog box appears. Select the Word document you want to import.

XMLSpy automatically generates an XML document with included CSS stylesheet. Each Word paragraph generates an XML element, whose name is defined as the name of the corresponding paragraph style in Microsoft Word.

23.10.4 Create XML Schema from DB Structure

The Create XML Schema from DB Structure command enables you to create an XML Schema from the structure of any of a variety of databases. The XML Schema-creation mechanism involves two steps:

1. A connection to the database is established.
2. Options for the database data selection and the XML Schema are specified.

To create an XML Schema from a DB structure you would do the following:

1. When you click the Create XML Schema from DB Structure command, the Database Connection Wizard box appears.
2. Using the Database Connection Wizard, you establish a connection to the database (for instructions, see Connecting to a Database).
3. After the connection to the database is established, the Create XML Schema from DB Structure dialog (screenshot below) displays tabs and windows that enable you to select the database structure to import. These options are described below. After finishing, click the Import button to import the database data.
Selection tab

In the Selection tab (screenshot below), the data source is listed in the Source Database pane. The Source pane displays either a representation of the tables of the database or an editable SQL statement for selecting the required tables.
In the Table view, you can select the tables in the database that you want to import by checking the table's check box (screenshot above). The contents of the table can then be displayed in the Preview pane. The table selection can be further filtered in the Preview pane (see below). You can configure the display in the Source pane as described in the section Import Database Data.

**Options tab**

In the Options tab (screenshot below), you can specify the format of the schema, its extension type, whether columns should be imported as elements or attributes, and the database constraints that should be generated in the schema.

---

**Schema format**: You can select between a flat (SQL/XML Standard) and a hierarchical schema.
The flat schema model is based on an ISO-ANSI SQL/XML specification INCITS/ISO/IEC 9075-14-2008. The SQL/XML specification defines how to map databases to XML. Relationships are defined in schemas using identity constraints; there are no references to elements. Hence the schema is flat structure which resembles a tree-like view of the database. The specification can be purchased at the ANSI store. For more information, see www.iso.org.

The hierarchical schema model displays the table dependencies visually, in a type of tree view where dependent tables are shown as indented child elements in the content model. Table dependencies are also displayed in the identity constraints tab. Tables are listed as global elements in the schema, and columns are the elements or attributes of these global elements (The user decides whether to map the columns as elements or as attributes). Relationships are created in a hierarchical way so that a foreign key field in one table is actually a reference to the global element that represents that table.

**Schema extension type:** Schema extension information is additional information read from a database that is then embedded in the schema as either annotation data or attributes. There are four extension type options when generating schemas: (i) no extensions information; (ii) SQL/XML extensions; (iii) MS SQL Server extensions; and (iv) Oracle extensions. These are described below:

- **None:** No additional information is provided by the database.
- **SQL XML:** SQL/XML extensions are only inserted when generating schemas in a flat format. The extension information is stored in annotations and is described in the SQL/XML specification (INCITS/ISO/IEC 9075-14-2008).
- **MS SQL Server:** Selecting Microsoft SQL Server, generates SQL Server extensions. See SQL Server Books Online for resources and MSDN's information about annotating XSD schemas. The following annotation-related elements are generated in the schema: sql:relation, sql:field, sql:datatype, sql:mapped.
- **Oracle:** Oracle extensions are selected by default when working with an Oracle database. Additional database information is stored as attributes. Detailed information can be found in Oracle's online documentation. The following subset of attributes is currently generated: SQLName, SQLType, SQLSchema.

**Note:** Although SQL Server and Oracle extensions can be generated for their respective databases they are not restricted in this way. This proves useful when working with a third database and wanting to generate a schema that later should be working with either SQL Server or Oracle.

**Preview pane**

The Preview pane (screenshot below) displays the structure of the table currently selected in the Selection tab. When a new table is selected in the Selection tab, click the **Reload** button in the Preview pane to refresh the preview.
A field can be specified to be imported as an element or attribute, or not to be imported, by clicking the symbol to the left of the column name. You can click through the element, attribute, and ignore options. In the screenshot above, the city field, for example, has been set to be imported as an element while the street field has been set to be imported as an attribute.

Datatype conversions

Information about the conversion of database datatypes to XML Schema datatypes is listed in the Appendices.

23.10.5 DB Import Based on XML Schema

The DB Import Based on XML Schema command creates an XML document which is valid according to a given XML Schema and contains data imported from a database. For this feature, the following databases are supported:

- Microsoft Access 2000 and 2003
- Microsoft SQL Server
- Oracle
- MySQL
- Sybase
- IBM DB2

The data to be imported is determined by the table that is selected in the database. With the required XML Schema (that on which you wish to base the import) as the active document in Schema View, connect to the database. Then select the table/s you wish to import, and click Import. The data is imported into an XML document, and the document has the structure of the XML Schema that was active when the data was imported.

In the example below, data from an MS Access database is imported with an XML Schema active in Schema View. These would be the steps to carry out for the import:

1. Open the schema file in Schema View (screenshot below).
2. Select the menu command **DB Import based on XML Schema**. This opens the **Connect to Data Source** dialog.
3. Select the Microsoft Access (ADO) option and click **Next**.
4. Click **Browse** and select the database file. Then click **Next**.
5. In the DB Import Based on XML Schema dialog which pops up, go to the Tables tab, select one or more tables you wish to import (for example, **Altova**), then click **Import**. The table is imported into an XML document that is displayed in Grid View.

**Datatype conversions**

Information about the conversion of database datatypes to XML Schema datatypes is listed in the **Appendices**.

### 23.10.6 Create DB Structure from XML Schema

XMLSpy allows you to create an empty database (or skeleton database) based on an existing schema file. The method described below is generally the same for each type of database.

1. Open the schema file in Schema/WSDL View
2. Select the menu command **Convert | Create DB Structure from XML Schema**. This pops up the **Connect to a Data Source** dialog, which enables you to connect to a database (DB).
3. Use the steps described in the section **Connecting to a Data Source** to connect to the required database. For example, to connect to a Microsoft Access database, select the Microsoft Access radio button, and continue the process to select a database. You can use an existing database or create a new database in which the schema structure will be contained.
4. In the Create DB Structure from XML Schema dialog, tables are created from the schema and displayed in a tree format at the location where they will occur in the DB. For
example, in the screenshot below, the Address table is created and selected for export. Tables that should not be exported should be deselected (by unchecking the check box or selecting the appropriate item from the context menu for that table).

Creating DB tables with relationships
If the XML Schema from which the DB structure is generated has relationships defined in the form of identity constraints, then these relationships are automatically created in the generated DB structure and displayed in the Table Structure. Tables with relationships are listed under the sections: Tables with ForeignKeys and Tables used by ForeignKeys. Tables without relationships are listed in the Independent Tables section.

In the Relationships tab, you can create and modify table relationships. The tab lists all possible primary-key/foreign-key relationships (screenshot below).
To create a relationship, do the following:

1. Select one of the possible primary-key/foreign-key relationships.
2. In the lower pane of the dialog, click the Plus button to create a relationship.
3. Select the required columns in each of the two tables from the respective dropdown lists.

You can also remove a relationship by selecting it and then clicking the Minus button.

Notes on database structure and connecting

The schema structure, defined by the identity constraints, is mirrored in the resulting database. The table below shows the type of database created, the restrictions, and the connecting methods, when using the Create DB Structure from XML Schema menu command.
XMLSpy will map both hierarchical and flat formatted schemas. XMLSpy recognizes both formats automatically.

The flat format is mapped to SQL in two different ways.

- SQL Server DB, Oracle DB, or Sybase DB:
  A schema that was generated in flat format, for one of the above databases, will have the schema catalog name extracted and used in the generated SQL script as the DB name. This means that the resulting SQL script will be executed on a target DB whose name must be identical to the schema catalog name.

- Access (2000 or 2003), MySQL, or DB2 DB:
  A schema that was generated in flat format, for one of the above databases, will ignore the schema catalog name when the SQL script is generated. This means that the resulting SQL script will be executed on a target database that was logged into.

Datatype conversions
Information about the conversion of XML Schema datatypes to database datatypes is listed in the Appendices.

23.10.7 Export to Text Files

The command **Convert | Export to Text Files** exports XML data into text formats for exchange with databases or legacy systems. On clicking this command, the Export XML to Text dialog pops. It consists of two parts (*shown in separate screenshots below*):

- an upper part with two tabs: (i) Selection, and (ii) Export Options.
- a lower part, which is a Preview window.

After you have selected the desired options in this dialog (*described below*), click the **Export** button to export to text file/s.
Selection

In the Selection tab (screenshot below), you can select the destination of the file to be exported and text generation options.

![Screen Shot](image)

**Destination:** The exported file can be saved directly to a folder. The file extension can be specified. The filenames will be those of the elements (in the XML file) that will be exported. Alternatively, untitled files can be exported to XMLSpy. These files will be displayed in the GUI, and can be saved later.

**Include comments:** Activate this option to include an XMLSpy-generated comment in the exported XML file. The comment will contain the SQL query used to select the data as well as a list in which there is one listitem for each column header in the database table.

**Create first row with field names:** When activated, the exported tables include the names of columns from the database. Otherwise column names will not be included in the exported text file.

**Remove delimiters:** Removes delimiters that are contained in text values in the exported data. Set the delimiter you want to remove by using the Delimiter combo box in this tab. For example, if this option is activated and the selected delimiter is the apostrophe, when you export the XML value Ba'ker, the string will be Baker in the exported text.

**Remove newlines:** Removes newlines from exported data.

**Delimiter:** Select from the drop-down list the character that you wish to have removed during export. Alternatively, enter the desired character string.
**Encoding**: Select from the drop-down list, the desired encoding for files that are generated during export.

**Byte order**: If you are exporting 16-bit or 32-bit Unicode (UCS-2, UTF-16, or UCS-4) files, you can also switch between little-endian and big-endian byte order.

**Export Options**
Additional export options, which are described below, can be specified in the Options tab (screenshot below):

**Start point of export**: You can choose to export the entire XML document or restrict your export to the data hierarchy starting from the currently selected element. The number of sub-levels below the start point that will be exported is specified in the *Export Depth* option.

**Export depth**: Specifies the number of sub-levels below the start point that will be exported.

**Export fields**: Depending on your XML data, you may want to export only elements, attributes, or the textual content of your elements. Note that you can deselect the export of individual elements in the Preview window.

**Automatic fields**: XMLSpy will produce one output file or table for each element type selected. You can choose to automatically create primary/foreign key pairs to link your data in the relational model, or define a primary key for each element.

**Exclude namespace name**: Together with the *Replace Colon With Underscore* radio button this is an either/or choice. Specifies whether namespace prefixes of elements and attributes should be
excluded or whether the colon in the namespace prefix should be replaced with an underscore.

**Apply Options:** After you have set options, click this button to apply the options. The preview in the preview pane will be updated with the new options.

**Preview window**
The Preview window (*screenshot below*) is displayed below the Selection and Options tabs.

![Preview window screenshot](attachment:image.png)

The *Resulting Elements from XML File* pane shows the node names that will be exported and the name in the generated file. You can select/deselect nodes that will be exported. When an element is selected, a preview of its structure is shown in a second pane below. In this pane, clicking to the left of a column header name toggles the export of that column on and off. In the screenshot above, the last column (*FK_Office*) has been toggled off.

### 23.10.8 Export to a Database

The command **Convert | Export to a Database** exports XML data to a database. On clicking this command, the Connection Wizard starts up and enables you to set up a connection to the database you wish to update. After a connection has been established, the Export Data to Database dialog pops. It consists of two parts (*shown separately in the screenshots below*):

- an upper part with two tabs: (i) Selection, and (ii) Export Options.
- a lower part, which is a Preview window.

After you have selected the desired options in this dialog (*described below*), click the **Export**
button to export to the database.

**Selection**

In the Selection tab, you can select the destination database and table generation options. The destination field selects the connection to the database. You must select whether the data is created as new tables, updates existing tables, or first tries to update an existing table and then creates a new table if an update is not possible. You can also set a stop action based on the number of errors, and, optionally, SQL script logging.

**Export Options**

Export options, which are described below, can be specified in the Options tab (screenshot below):

![Export XML to Text](image)

**Start point of export:** You can choose to export the entire XML document or restrict your export to the data hierarchy starting from the currently selected element. The number of sub-levels below the start point that will be exported is specified in the Export Depth option.

**Export depth:** Specifies the number of sub-levels below the start point that will be exported.

**Export fields:** Depending on your XML data, you may want to export only elements, attributes, or the textual content of your elements. Note that you can deselect the export of individual elements in the Preview window.

**Automatic fields:** XMLSpy will produce one output file or table for each element type selected. You can choose to automatically create primary/foreign key pairs to link your data in the relational model, or define a primary key for each element.
**Exclude namespace name:** Together with the *Replace Colon With Underscore* radio button this is an either/or choice. Specifies whether namespace prefixes of elements and attributes should be excluded or whether the colon in the namespace prefix should be replaced with an underscore.

**Apply Options:** After you have set options, click this button to apply the options. The preview in the preview pane will be updated with the new options.

**Preview window**

The Preview window (screenshot below) is displayed below the Selection and Options tabs.

The Resulting Elements from XML File pane shows the name of the nodes in the XML document that will be exported and its corresponding name in the generated file. You can select/deselect nodes that will be exported. When an element is selected, a preview of its structure in the generated file is shown in a second pane below. This preview can be switched between a preview of: (i) data in the generated structure (*Show Data*); or (ii) definitions of each column in the generated structure (*Show Definition*). The screenshot above shows the column definitions.

In this second pane, clicking to the left of a column name cycles the column through four settings: (i) Include in table structure; (ii) Unique constraint; (iii) Primary Key constraint; (iv) Exclude from table structure. In the screenshot above, the *Location* column has a Unique constraint, while the *Fax* column has been excluded from the table structure. All the other columns are included in the table structure.
When the element's table structure shows field definitions (Show Definition), the definitions can be edited by selecting the definition and selecting an option from the definition's combo box (see screenshot above).

23.10.9 Convert XML Instance to/from JSON

If the active document is an XML document, this command generates a JSON document from it. If the active document is a JSON document, the command generates an XML document from it. The generated document is opened in a new window, and can then be saved to any location. Conversion options are described below. For more information about JSON and JSON editing support in XMLSpy, see the section JSON and JSON Schema.

Sample conversions

Given below is an example of a source XML document, and, below it, the JSON document generated by the Convert XML Instance to/from JSON command.

XML document

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Person first="Jim" last="James">
  <Address>
    <street>4 New Street</street>
    <city>New York</city>
    <state>NY</state>
    <code>10123</code>
  </Address>
  <Tel type="home">
    123 123-1234
  </Tel>
  <Tel type="office">
    123 987-9876
  </Tel>
</Person>
```

JSON document

```json
{
  "XML": {
    "version": 1.0,
    "encoding": "UTF-8"
  },
  "Person": {
    "first": "Jim",
    "last": "James",
    "Address": {
      "street": "4 New Street",
      "city": "New York",
      "state": "NY",
      "code": 10123
    },
    "Tel": [
      { "type": "home",
        "Text": "123 123-1234"
      },
      { "type": "office",
        "Text": "123 987-9876"
      }
    ]
  }
}
```
To convert a JSON document to XML, make the JSON document active and click the **Convert XML Instance to/from JSON** command.

**XML to JSON conversion options**

When you click the **Convert XML Instance to/from JSON** command to convert an XML instance document to a JSON instance document, the Convert XML to JSON dialog (screenshot below) appears. You can select whether you wish to convert to JSON or JSON5. Then set the conversion options you want, and click **OK**. A JSON instance document will be generated from the XML instance, and the generated JSON document will be opened in a new window.

The first two options define whether prefixes should be added to JSON property names so that conflicts with elements at the same level are avoided. The two listings below explain this. The XML attribute `somenode` has been converted to the JSON property `@somenode`. In this way, a conflict with the JSON property `somenode` (created from the XML element `somenode`) is avoided.

**XML instance**

```xml
<root somenode="value">
    <somenode>content</somenode>
</root>
```

**JSON instance**

```json
{
    "root": {
        "@somenode": "value",
        "somenode": "content"
    }
}
```

The next options enable you to specify whether certain types of XML nodes are to be converted or
not. If XML comments are included they are given the name 
"#". Text nodes (that typically occur in elements with mixed content) are given the name 
"$". If an XML nodes has a namespace prefix, then the corresponding JSON name will be created with this namespace prefix. If elements with the same name exist at the same level, they are considered to be equal components. Similarly, nodes such as comments, processing instructions, and text() at the same level are also equal components. If equal components are present at the same level, you are able to choose whether to create an array or not. The options are whether to create the array out of all such equal components, only neighboring equal components, or not to create an array at all.

The Use XML Schema type info option enables conversions to be made on the basis of the XML Schema type of the source node. For example, if a node is defined in the schema as being of type xs:string, then the target JSON object's property will be of JSON type string and will be enclosed in quotes. This is useful if, for example, a number is stored as a string in the source XML node and the conversion to JSON must also be faithful in terms of type.

JSON to XML conversion options

When you click the Convert XML Instance to/from JSON command to convert a JSON instance document to an XML instance document, the Convert JSON to XML dialog (screenshot below) appears. Set the conversion options you want, and click OK. An XML instance document will be generated from the JSON instance, and the generated XML document will be opened in a new window.

Note the following points:

- JSON object properties are converted to XML elements. The first options in the dialog enable you to chose whether some types of properties are created or not.
- Encode colons in property names: If selected, colons in JSON names are encoded and not created as colons. If not selected, colons are left as is.
- Keep JSON value type information: If selected, a property's JSON type information is created as an attribute-value pair of the corresponding element.
- Create container element for every JSON array: The container element in the XML document will be given the name of the JSON array object. The items of the JSON array are created as XML elements within this container. Each is given the name you specify in the Array item element name text box.
23.10.10 Convert XML Schema to/from JSON Schema

If the active document is an XML Schema, this command generates a JSON schema document from the XML schema. If the active document is a JSON schema, the command generates an XML Schema from the JSON schema. The generated document is opened in a new window, and can then be saved to any location. Conversion options are described below. For more information about JSON and JSON editing support in XMLSpy, see the section JSON and JSON Schema.

XML Schema to JSON Schema conversion options

When you click the Convert XML Schema to/from JSON Schema command to convert an XML Schema document to a JSON schema, the Convert XML Schema to JSON Schema dialog (screenshot below) appears. Select the JSON Schema version you want and the conversion options, and click OK. A JSON schema will be generated from the XML Schema, and the generated document will be opened in a new window.

The top part of the dialog provides information about how certain XML Schema components are converted. The bottom part of the dialog provides the following options:

- Create a property named "#" in each subschema: If selected, a property with this name is created in each JSON schema definition.
- Create pattern properties matching properties prefixed with "/", "@xsi", "@xmlns": Specifies, for each of these prefixes, a pattern property to match properties with names that have these prefixes. For more information about pattern properties, see the section JSON Objects and Properties.
- Always create arrays for particles with maxOccurs > 1: In XML Schema, particles are the elements of complex content models. If the number of occurrences is more than one, then the particles are defined as an array in JSON Schema. Otherwise, they are defined as properties of a JSON object.

JSON Schema to XML Schema conversion options

When you click the Convert XML Schema to/from JSON Schema command to convert a JSON Schema document to an XML schema, the Convert JSON Schema to XML Schema dialog (screenshot below) appears. Set the conversion options you want, and click OK. An XML schema
will be generated from the JSON Schema, and the generated document will be opened in a new window.

You can select the following options:

- Whether JSON property names that begin with '@' and '$' are created or not. They would be created, respectively, as attribute nodes and text nodes.
- Whether properties named '#' are created, as XML comment nodes, or not.
- Whether pattern properties that match properties prefixed with '?', '@xsi', and/or '@xmlns' are ignored or not. If not ignored, then the properties prefixed with '?', '@xsi', and '@xmlns' are converted, respectively, to processing instructions, xsi: prefixed attributes, and xmlns: prefixed attributes.
23.11 View Menu

The View menu (screenshot below) controls the display of the active Main window and allows you to change the way the document is displayed.

This section provides a description of commands in the View menu.

23.11.1 Text View

This command switches the current view of the document to Text View, which enables you to edit the document in its text form. It supports a number of advanced text editing features, described in detail in the Text View section of this document.

Note: You can configure aspects of Text View in various tabs of the Options dialog (Tools | Options).

23.11.2 Enhanced Grid View
This command switches the current document into **Grid View**. If the previous view was **Text View**, the document is automatically checked for well-formedness.

### Table View

XMLSpy allows you to display recurring elements in a **table structure** in Grid View. This function is available wherever the Grid View can be activated, and can be used when editing any type of XML file: XML, XSD, XSLT, etc. For more information, see the **Grid View** section of this documentation.

### 23.11.3 Schema Design View

This command switches the current document, if it is an XML Schema document, to Schema Design View. For a detailed description of mechanisms available in this view, see the **Schema View** section of this documentation.

### 23.11.4 Authentic View

This command switches the current document to **Authentic View**.

Authentic View enables you to edit XML documents based on StyleVision Power Stylesheet templates created in Altova's StyleVision application. These templates (StyleVision stylesheets or SPS files) display XML documents in a graphical format that makes editing the XML document easier (than editing it in a text format with markup).

If an XML document is associated with an SPS file (**Authentic | Assign a StyleVision Stylesheet**), the XML document can be viewed in Authentic View. You can also open an SPS file as a new empty template in Authentic View, in one of two ways:
• Select the File | New command and then click the Select a StyleVision stylesheet button.
• Select the Authentic | New Document command and then browse for the SPS file.

See the Authentic View and StyleVision documentation for more information.

23.11.5 Browser View

This command switches the current document to Browser View. An XML-enabled browser renders the XML document using information from available CSS and/or XSL stylesheets.

When switching to Browser View, the document is first checked for validity if the Validate upon saving option in the File section of the Options dialog (Tools | Options) is checked. For more information, see the Browser View section of this documentation.

23.11.6 Expand

This command (shortcut: numeric pad ‘+’) is enabled in Grid View and expands the selected element one level. The element remains selected after expansion, so you can expand the element additional levels by repeatedly clicking the shortcut ‘+’ key.

23.11.7 Collapse

This command (shortcut: numeric pad ‘-’) is enabled in Grid View and collapses the selected element one level. You can expand or collapse any element by clicking the gray bar to the left of the element.

23.11.8 Expand Fully

This command (shortcut: * or x on the numeric keypad) is enabled in Grid View and in Text View if the Text View folding margin is active. It expands all descendant nodes of the selected element.
23.11.9 **Collapse Unselected**

This command (shortcut: Ctrl + numeric pad '-') is enabled in Grid View and keeps the selected item uncollapsed while collapsing all others items. This helps maximize focus on one element and its children while reducing the focus on other nodes.

23.11.10 **Optimal Widths**

This command is enabled in Grid View and adjusts the widths of all columns in Grid View so that each column has a width that exactly accommodates in one line the longest text string in any of its cells. A maximum optimal width can be specified in the View section of the Options dialog (Tools | Options). Note that optimal widths are calculated on the basis of the visible cells of columns. This enables the optimization of the view when individual elements are collapsed or expanded.

23.11.11 **Word Wrap**

This command enables or disables word wrapping in Text View. When word-wrapping is toggled on, text will wrap at the window's edge.

23.11.12 **Go to Line/Character**

This command (shortcut: Ctrl+g) is enabled in Text View and Grid View. It pops up a dialog (screenshot below) in which you can enter the line number and character number to go to. In Text View, the cursor will jump to the position you entered. In Grid View, the node closest to the line and/or character number you entered will be highlighted.
This feature is useful when you need to quickly navigate to a location, for example, when the location of an error is given in an error message.

23.11.13 Go to File

This command is enabled in Text View and Grid View. When the cursor is placed inside text that references a file (in Text View) or in a node (in Grid View) that contains text referencing a file, the referenced document is opened. It opens a document that is being referred to, from within the file you are currently editing.

23.11.14 Text View Settings

The Text View Settings command is enabled in Text View. It opens the Text View Settings dialog (screenshot below), in which you can configure Text View. A shortcut icon to open the dialog is available in the Text toolbar.
Margins
In the Margins pane, the Line Number, Bookmark, and Source Folding margins can be toggled on and off. Each of these is a separate margin in Text View and displays, respectively: (i) line numbers, (ii) bookmarks, and (iii) source folding icons to expand/collapse nodes. The settings of the Margins pane determine whether the margins are displayed in Text View or not. Bookmark commands are in the Edit menu. You can expand and collapse nodes in Text View only if the Folding margin setting is toggled on.

Tabs
The Tab pane enables you to set the tab size in terms of spaces. The radio buttons below the Tab size setting determine whether documents are displayed with tab or space indentation when pretty-printing-with-indentation is enabled in the View section of the Options dialog (Tools | Options).

Visual Aid
The Visual Aid pane contains settings to toggle on indentation guides (tab-distanced vertical lines that show the indentation of the text; see screenshot below), end-of-line markers, and whitespace markers (tabs and space characters). (Tabs are indicated with arrows, while spaces are indicated with dots (both pink in the screenshot below). The colors of whitespace markers can be customized in the Text View options of the Options dialog.)
Enable auto-highlighting

If highlighting is enabled, then all occurrences of a selection in Text View are highlighted. What constitutes a selection can be set via the options in this pane. A selection can be defined to be an entire word or a fixed number of characters, with the text-casing counting or not counting for a match. For a character selection, you can specify the minimum number of characters to match (for example, two or more characters). In Text View, all occurrences of character sequences that match your selection will be highlighted. For word searches, element names, attribute names, attribute values without quotes, and the angular brackets of element tags are considered to be separate words.

Key map

The key map is a list of XMLSpy shortcuts and their associated commands.
23.12 Browser Menu

The commands in the Browser menu are enabled in Browser View only. The Back and Forward commands, however, is enabled in Schema View also, where it takes you to the previously used command.

23.12.1 Back

The Back command (shortcut: Alt + Left arrow) is enabled in Browser View and Schema View.

In Browser View, the Back command displays the previously viewed page. The Backspace key achieves the same effect. The command is useful if you click a link in your XML document and then want to return to your XML document.

In Schema View, the Back command takes you to the previously viewed component or view. It can take you back to up to 500 previously viewed positions.

23.12.2 Forward

The Forward command (shortcut: Alt + Right arrow) is enabled in Browser View. In Schema View it is enabled only after you have used the Back command. The Forward command moves you forward through (i) previously viewed pages in Browser View, and (ii) previous views of schema components in Schema View.

23.12.3 Stop

The Stop command in enabled in Browser View and instructs the browser to stop loading your
document. This is useful if large external files or graphics are being downloaded over a slow Internet connection, and you wish to stop the process.

23.12.4 Refresh

The Refresh (F5) command is enabled in Browser View and updates Browser View by reloading the current document and documents related to the current document (such as CSS and XSL stylesheets, and DTDs).

23.12.5 Fonts

The Fonts command rolls out a sub-menu from which you can select the default font size for rendering the text of your XML document. The selection is available in Browser View only.

23.12.6 Separate Window

The Separate Window command is enabled in Browser View and undocks the Browser View of the document from the other views. As a separate window, Browser View can be displayed side-by-side with an editing view of the document.

To refresh the separated Browser View after making a change in an editing view, press F5 in the editing view. To dock a separate Browser View window back into the window containing the other views, make the Browser View window active and click the Separate Window command.
23.13 Tools Menu

The Tools menu allows you to:

- Check the spelling of your XML documents
- Access the scripting environment of XMLSpy. You can create, manage and store your own forms, macros and event handlers
- View the currently assigned macros
- Compare any two files to check for differences
- Compare any two folders to check for differences
- Access customized commands that use external applications. These commands can be created in the Tools tab of the Customize dialog.
- Define global resources
- Change the active configuration for global resources in XMLSpy
- Add RaptorXML Servers for XML and XBRL validation, and to configure RaptorXML validation options
- Select a Raptor Server configuration as the active configuration
- Customize your version of XMLSpy: define your own toolbars, keyboard shortcuts, menus, and macros
- Define global XMLSpy settings

23.13.1 Spelling

XMLSpy's spellchecker with built-in language dictionaries (see note below) is enabled in Text View, Grid View, and Authentic View. If you wish to spellcheck a document that you have been editing in another view, you can switch to Text View or Grid View and run a spelling check. For
example, if you have been editing an XML Schema document in Schema View, switch to Text View or Grid View for the spelling check.

**Note:** The selection of built-in dictionaries that ship with Altova software does not constitute any language preferences by Altova, but is largely based on the availability of dictionaries that permit redistribution with commercial software, such as the MPL, LGPL, or BSD licenses. Many other open-source dictionaries exist, but are distributed under more restrictive licenses, such as the GPL license. Many of these dictionaries are available as part of a separate installer located at [http://www.altova.com/dictionaries](http://www.altova.com/dictionaries). It is your choice as to whether you can agree to the terms of the license applicable to the dictionary and whether the dictionary is appropriate for your use with the software on your computer.

This section describes how to use the spellchecker. It is organized into the following sub-sections:

- Selecting the spellchecker language
- Defining the scope of the check
- Running the spelling check

**Selecting the spellchecker language**

The spellchecker language can be set as follows:

1. Click the **Tools | Spelling Options** menu command.
2. In the XML Spelling Options dialog that pops up, click the **More Spelling Options** button.
3. In the Spelling Options dialog that now pops up (screenshot below), select one of the installed dictionaries from the dropdown list of the Dictionary Language combo box.

```
Spelling Options

Spelling

- [ ] Always suggest corrections
- [ ] Make corrections only from the main dictionary
- [x] Ignore words in UPPER case
- [ ] Ignore words with numbers
- [x] Split CamelCase words

Dictionary Language

English (US)

Get more dictionaries at [http://www.altova.com/dictionaries](http://www.altova.com/dictionaries)

[OK] [Cancel]
```

4. Click **OK** to finish.

The dictionary language you selected will be used by the spellchecker for spelling checks. If the language you want is not already installed, you can download additional language dictionaries.
How to do this is described in the section, Adding dictionaries for the spellchecker.

Defining the scope of the check

When the spellchecker is run in Text View or Grid View, the scope of the check can be defined immediately before starting the check. Do this by selecting the command Tools | Spelling Options and by defining the required scope in the XML Spelling Options dialog that pops up (see screenshot below).

You can select which elements and attributes are to be checked and whether CDATA sections and comments should be checked. Also see the description of the Spelling Options command for related information.

Running the spellchecker

The Tools | Spelling (Shift+F7) command automatically starts checking the currently active XML document according to the defined scope. If an unknown word is encountered, the Spelling: Not in Dictionary dialog pops up (screenshot below). Otherwise the spelling check runs through to completion.
The various parts of the Spelling: Not in Dictionary dialog and the available options are described below:

Not in Dictionary
This text box contains the word that cannot be found in either the selected language dictionary or user dictionary. The following options are available:

- You can edit the word in the text box manually or select a suggestion from the Suggestions pane. Then click Change to replace the word in the XML document with the edited word. (Double-clicking a suggestion inserts it directly in the XML document.) When a word is shown in the Not in Dictionary text box, it is also highlighted in the XML document, so you can edit the word directly in the document if you like. Clicking Change All will replace all occurrences of the word in the XML document with the edited word.
- You can choose to not make any change and to ignore the spellchecker warning—either just for the current occurrence of the word or for every occurrence of it.
- You can add the word to the user dictionary and so allow the word to be considered correct for all checks from the current check onwards.

Suggestions
This list box displays words resembling the unknown word (supplied from the language and user dictionaries). Double-clicking a word in this list automatically inserts it in the document and continues the spellchecking process.

Ignore once
This command allows you to continue checking the document while ignoring the first occurrence of the unknown word. The same word will be flagged again if it appears in the document.
Ignore all
This command ignores all instances of the unknown word in the whole document.

Add to dictionary
This command adds the unknown word to the user dictionary. You can access the user dictionary (in order to edit it) via the Spelling Options dialog.

Change
This command replaces the currently highlighted word in the XML document with the (edited) word in the Not in Dictionary text box.

Change all
This command replaces all occurrences of the currently highlighted word in the XML document with the (edited) word in the Not in Dictionary text box.

Recheck Document
The Recheck Document button restarts the check from the beginning of the document.

Options
Clicking the Options button opens a dialog box depending on the current view.

- If the current view is Authentic View, the Spelling Options dialog box is opened.
- If the current view is Text View or Grid View, then the XML Spelling Options dialog box is opened.

For more information about these dialog boxes, see the section Spelling Options.

Close
This command closes the Spelling dialog box.

23.13.2 Spelling Options
The Tools | Spelling Options command opens the Spelling Options. Depending on which view is active, the Tools | Spelling Options command opens either the Spelling Options dialog directly (Schema View, Authentic View, Browser View), or the XML Spelling Options dialog (Text View, Grid View, ). The XML Spelling Options dialog has a Spelling Options button to access the Spelling Options dialog.

The various settings available in these two dialogs are described in the sub-sections of this section:

- Spelling check context
- Spelling options
- Adding dictionaries for the spellchecker
- Working with the user dictionary
XML Spelling Options dialog

Clicking the Tools | Spelling Options command in Text View or Grid View opens the XML Spelling Options dialog (screenshot below), in which you can select the scope of the spelling check. You can select which elements and attributes are to be checked and whether CDATA sections and comments should be checked.

![XML Spelling Options dialog](image)

You can compile a list of elements and/or attributes that you wish to have spellchecked or have excluded from the spelling check. Alternatively, you can choose to run the spelling check on all elements and/or attributes or on no element or attribute.

Clicking the More Spelling Options button at the bottom of the dialog opens the Spelling Options dialog.

Spelling options

The Spelling Options dialog is used to define global spellchecker options.
Always suggest corrections:
Activating this option causes suggestions (from both the language dictionary and the user dictionary) to be displayed in the Suggestions list box. Disabling this option causes no suggestions to be shown.

Make corrections only from main dictionary:
Activating this option causes only the language dictionary (main dictionary) to be used. The user dictionary is not scanned for suggestions. It also disables the User Dictionary button, preventing any editing of the user dictionary.

Ignore words in UPPER case:
Activating this option causes all upper case words to be ignored.

Ignore words with numbers:
Activating this option causes all words containing numbers to be ignored.

Split CamelCase words
CamelCase words are words that have capitalization within the word. For example the word "CamelCase" has the "C" of "Case" capitalized, and is therefore said to be CamelCased. Since CamelCased words are rarely found in dictionaries, the spellchecker would flag them as errors. To avoid this, the Split CamelCase words option splits CamelCased words into their capitalized components and checks each component individually. This option is checked by default.

Dictionary Language
Use this combo box to select the dictionary language for the spellchecker. The default selection is US English. Other language dictionaries are available for download free of charge from the Altova website.
Adding dictionaries for the spellchecker

For each dictionary language there are two Hunspell dictionary files that work together: a .aff file and .dic file. All language dictionaries are installed in a Lexicons folder at the following location:

C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons.

Within the Lexicons folder, different language dictionaries are each stored in different folder: <language name>\<dictionary files>. For example, files for the two English-language dictionaries (English (British) and English (US)) will be stored as below:

C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (British)\en_GB.aff
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (British)\en_GB.dic
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (US)\en_US.dic
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (US)\en_US.dic

In the Spelling Options dialog, the dropdown list of the Dictionary Language combo box displays the language dictionaries. These dictionaries are those available in the Lexicons folder and have the same names as the language subfolders in the Lexicons folder. For example, in the case of the English-language dictionaries shown above, the dictionaries would appear in the Dictionary Language combo box as: English (British) and English (US).

All installed dictionaries are shared by the different users of the machine and the different major versions of Altova products (whether 32-bit or 64-bit).

You can add dictionaries for the spellchecker in two ways, neither of which require that the files be registered with the system:

- By adding Hunspell dictionaries into a new subfolder of the Lexicons folder. Hunspell dictionaries can be downloaded, for example, from https://wiki.openoffice.org/wiki/Dictionaries or http://extensions.services.openoffice.org/en/dictionaries. (Note that OpenOffice uses the zipped OXT format. So change the extension to .zip and unzip the .aff and .dic file to the language folders in the Lexicons folder. Also note that Hunspell dictionaries are based on Myspell dictionaries. So Myspell dictionaries can also be used.)
- By using the Altova dictionary installer, which installs a package of multiple language dictionaries by default to the correct location on your machine. The installer can be downloaded via the link in the Dictionary language pane of the Spelling Options dialog (see screenshot below). Installation of the dictionaries must be done with administrator rights, otherwise installation will fail with an error.

Note: It is your choice as to whether you agree to the terms of the license applicable to the dictionary and whether the dictionary is appropriate for your use with the software on your computer.
Working with the user dictionary

Each user has one user dictionary, in which user-allowed words can be stored. During a spellcheck, spellings are checked against a word list comprising the words in the language dictionary and the user dictionary. You can add words to and delete words from the user dictionary via the User Dictionary dialog (screenshot below). This dialog is accessed by clicking the User Dictionary button in the Spelling Options dialog (see second screenshot in this section).

To add a word to the user dictionary, enter the word in the Word text box and click Add. The word will be added to the alphabetical list in the Dictionary pane. To delete a word from the dictionary, select the word in the Dictionary pane and click Delete. The word will be deleted from the Dictionary pane. When you have finished editing the User Dictionary dialog, click OK for the changes to be saved to the user dictionary.

Words may also be added to the User Dictionary during a spelling check. If an unknown word is encountered during a spelling check, then the Spelling dialog pops up prompting you for the action you wish to take. If you click the Add to Dictionary button, then the unknown word is added to the user dictionary.

The user dictionary is located at: C:\Users\<user>\Documents\Altova\SpellChecker\Lexicons\user.dic

23.13.3 Scripting Editor

The Scripting Editor command opens the Scripting Editor window. How to work with the Scripting Editor is described in the Scripting section of this documentation.

Note: The .NET Framework version 2.0 or higher will have to be installed on your machine in order for the Scripting Editor to run.
23.13.4 Macros

Mousing over the Macros command rolls out a submenu containing the macros defined in the Scripting Project that is currently active in XMLSpy (screenshot below).

![Macros Menu](image)

Clicking a macro in the submenu (see screenshot above) runs the macro.

23.13.5 Comparisons

XMLSpy provides a comparison (or differencing) feature, with which you can compare XML and Text files, as well as folders, in order to check for differences.

![Comparison Menu](image)

There are the following menu items in the Tools menu that enable you to perform comparison tasks on files and folders:

- **Compare open file with**
- **Compare directories**
- **Compare options**

These commands are described in detail in the following sub-sections.

**Compare Open File With**

This command allows you to compare the open file with another file. The comparison shows the files to compare tiled vertically in the main window with the differences between them highlighted in each file. If a difference is between two files, then the difference is highlighted in green. If content is different across three files (available in the Enterprise Edition only), then this is referred to as a conflict, and the conflict is highlighted in pink. You can compare files as XML documents (where the structure and semantics of tags is significant) or as text documents.

**Note:** Three-way file comparisons are available in the Enterprise Edition only.

To compare the active file (in the GUI) with another file, do the following:

1. With the file to compare active in the Main Window (only one open file can be active at a given time), click **Tools | Compare open file with**. A Browse dialog appears (screenshot below), in which you can browse for one or two files to compare with the active file.
2. Click **Browse** to select a file via file explorer, a global resource, or a URL. Click **Window** to select a file that is open in one of the windows of XMLSpy.

3. Click **OK**. The Settings dialog appears (*screenshot below*). These settings are described in the topic **Compare Options**. If you wish to not have this dialog appear each time you start a compare session, uncheck **Show settings before starting compare** (located at the bottom of the dialog).
4. Select the required settings, then click **OK**. Two things happen: (i) The files to compare are displayed side-by-side in separate panes; (ii) The Compare Files control window appears. (*Screenshots in next section, File Comparisons.*)

5. You can navigate and merge differences by using the buttons in the Compare Files control window. (*For usage information, see the next section, File Comparisons.*)

6. When you finish reviewing and/or merging differences, click **Done**.

**Note:** Compare settings can be changed during a Compare session (by selecting **Tools** | **Compare options**), but will only take effect from the next Compare session onward; the new settings will not affect the current session.

### File comparisons and merging of differences

There are two types of comparison:

- **Two-way comparison:** Two files are compared. The file that was active (when comparison was requested; called active file for short) is displayed in the left-hand pane and is compared against a second file (*see screenshot below*).

- **Three-way comparison (Enterprise Edition only):** Three files are compared. The active file is displayed in the left-hand pane. The first of the two files that were selected for comparison against the active file is displayed in the middle pane. This middle file is known as the base file. You can merge a difference between the left-hand file and the base file, or a difference between the right-hand file and the base file.

**Note:** You should not move panes in the window; otherwise, the Copy To directions of the buttons in the Compare Files control window might not be correct any longer.

### Differences and conflicts

Differences and conflicts are highlighted in different colors (*see Highlight colors below*) and can be navigated separately.

- **Differences:** When content at corresponding locations in two files is different. Highlighted in green.
- **Conflicts:** When content at corresponding locations in three files is different. Highlighted in pink.

**Note:** Two-way comparisons show only differences, no conflicts.
Navigating differences and conflicts
You can navigate through the document by using the buttons in the Select Difference pane of the Compare Files control window (shown below: two-way comparison, followed by three-way comparison). The pane contains four buttons for two-way comparison, and six buttons for three-way comparison (see screenshots).

The Compare Files control window contains the following buttons:

- **First**: Goes to the first of all the differences and conflicts in the document.
- **Last**: Goes to the last of all the differences and conflicts in the document.
- **Previous**: Goes to the previous difference from the currently active difference or conflict.
- **Next**: Goes to the next difference from the currently active difference or conflict.
- **Previous Conflict**: Goes to the previous conflict from the currently active difference or conflict.
- **Next Conflict**: Goes to the next conflict from the currently active difference or conflict.

Merging differences and conflicts
You can use the buttons in the Merge pane to copy the highlighted content from one pane to another. Use the appropriate Copy-direction button of a pane-pair. In order to enable merging, the following Compare options must be set:

- **Detailed differencing** must be checked, and
- **Ignore node depth** must be unchecked.

When you finish reviewing and/or merging differences, click **Done**.

**Note:** If you wish to undo a merge, stop the Compare session, select the file in which the change is to be undone, and select **Edit | Undo** or press **Ctrl+Z**.

**Note:** While the Compare session is active, no editing or change of views is allowed. Any attempt to edit or change the view of either file will pop up a message warning that the Compare session will be ended.

Highlight colors
If corresponding lines are different in two files, then they are highlighted in green in both files (differences). If lines are different in three files, then they are highlighted in pink in all three files. If a difference is selected, it is indicated in both files in a darker green color. If a conflict is selected, it is indicated in both files in a darker pink color.

The highlight colors used in Compare sessions are given in the table below.

| Difference | Current difference |
Compare Directories

The Compare Directories command allows you to compare two directories, with or without their sub-directories. Directories are compared to indicate missing files, and whether files of the same name are different or not.

To compare two directories:

1. Click Tools | Compare directories. The following dialog appears.

2. Browse for the directories to compare, and check the Include subdirectories check box if you wish to include subdirectories in the Compare.

3. Select the file types you want to compare in the Files of type field. There are three options in the dropdown menu: (i) XML filetypes; (ii) Filetypes defined in XMLSpy; and (iii) all filetypes.

4. Click OK. The Settings dialog (described in Compare Options) appears.

5. Select the required settings for comparing files.

6. Click OK. A dialog will appear indicating the progress of the Compare.

The result will appear in a window, and will look like this:
Directory symbols

All directory names are given in black.

- Directory is collapsed and its contents are not displayed.

- Directory is expanded, indicated by the turned down corner. The contents are displayed.

- Directory contains files, all of which either cannot be compared or are not different from the corresponding file in the compared directory.

- Directory contains one or more files that do not exist in the compared directory.

- Directory contains one or more files that are different from the corresponding file in the compared directory.
Directory contains one or more files that do not exist in the compared directory and one or more files that are different from the corresponding file in the compared directory.

File symbols
The colors in which file names appear depend on their compare status. These colors are noted below.

- **This file cannot be compared (with the corresponding file in the compared directory).** A question mark appears in the middle column. The file name appears in black.

- **This file is not different from the corresponding file in the compared directory.** An equals-to sign appears in the middle column. The file name appears in black.

- **This file does not exist in the compared directory.** The middle column is empty. The file name appears in blue.

- **This file is different from the corresponding file in the compared directory, and the last modification to this file is more recent than the last modification to the corresponding file.** The newer file appears in a brighter red, and the icon shows the brighter red file symbol on the side having the newer file.

Viewing options

- Select what files to show by checking or unchecking the options in the Show pane at the bottom of the Result window.
- Open or close all subdirectories by clicking the appropriate button in the Directories pane.
- Expand or collapse subdirectories by double-clicking the folder icon.
- The Size and Last Modified can be toggled on and off by right-clicking on either file titlebar and clicking Size and Last Modified.

- Change column widths by dragging columns.
- The Result window can be maximized, minimized, and resized.

Comparing and merging files
Double-clicking on a line opens both files on that line in the Main Window, and directly starts a File Compare for the two files. You can then continue as in a regular Compare session (see Compare open file with...).
**Compare Options**

Click **Tools | Compare options** to open the Settings dialog (see screenshot). In it you make the settings for your Compare sessions. The settings that are current when a Compare session is started are the settings that are applied to that Compare session.

![Settings dialog](image)

**View results**

Selects the view in which results are shown. You can select from the following options:

- Grid View (XML comparison)
- Text View with Textual Comparison Only unchecked (XML comparison)
- Text View with Textual Comparison Only checked (Text comparison)

If a view that provides XML comparison is selected, then the documents are treated as XML documents, and XML Compare Options are enabled. If Text comparison is selected, only Compare Options valid for Text comparison (Whitespaces and Case-sensitivity) are enabled; all other Compare Options are disabled.
Note: You can merge differences in both Grid View and Text View, and in both XML and Text comparison modes. If you wish to undo a merge, stop the Compare session, select the file in which the change is to be undone, and select Edit | Undo or press Ctrl + Z.

Detailed differencing
If unselected, differences in immediate sibling elements are represented as a single difference, and the merge option is disabled. If selected, differences in immediate siblings are represented as separate differences, and merging is enabled.

Note: The Detailed differencing check box must be checked to enable merging.

Whitespaces
Whitespace characters are space, tab, carriage return, and line feed. When whitespace is normalized, consecutive whitespace characters are reduced to one whitespace character; however, note that, according to the XML specification, leading and trailing whitespace in attribute values are completely removed when whitespace is normalized. The options here compare files with: (i) whitespace unchanged; (ii) whitespace normalized; and (iii) all whitespace stripped. The Whitespaces option is available for both XML and Text comparisons.

Case sensitivity
If the Ignore case check box is checked, then you have the option of ignoring or not ignoring case in node names (for XML comparisons only). The Case-sensitivity option is available for both XML and Text comparisons.

Namespace/Prefix
These are options for ignoring namespaces and prefixes when searching for differences.

Order
If Ignore order of child nodes is checked, then the position of child nodes relative to each other does not matter. The comparison is made for the entire set of child nodes, and if the only difference between a child node in one document and a child node in the compare document is the relative position in the nodeset, then this difference is ignored. Each child element node is identified by its name, its attributes, and its position. If the names of more than one node in the sibling set are the same, then the position of these nodes is used to identify the nodes even if the "Ignore order of child nodes" option is checked. This option applies to each level separately. If Ignore order of child nodes is unchecked, then differences in order are represented as differences.

The option of ignoring the order of attributes is also available, and applies to the order of attributes of a single element.

Entities
If "Resolve entities" is selected, then all entities in the document are resolved. Otherwise the files are compared with the entities as is.
Text
If "Ignore text" is selected, then differences in corresponding text nodes are not reported.

Ignore node types
Check the node types that will not be compared in the Compare session. Node types that may be ignored are Attributes, CDATA, Comments, Processing Instructions, DOCTYPE statements, and the XML declaration.

Depth
If Ignore node depth is checked, then the additional depth of any element (i.e. more levels of descendants) relative to the depth of the corresponding element in the compared file is ignored. This option must be unchecked to enable merging.

Show settings before starting compare
Checking this option causes the Settings dialog (this dialog) to appear before each file or directory comparison is carried out (via the Compare open file with and Compare Directories commands). Having the Settings dialog appear before each comparison allows you to check and modify the settings for each comparison.

If this command is unchecked, then the Compare session will start directly when a comparison is invoked.

23.13.6 User-defined Tools
Placing the cursor over the User-defined Tools command rolls out a sub-menu containing custom-made commands that use external applications. You can create these commands in the Tools tab of the Customize dialog. Clicking one of these custom commands executes the action associated with this command.

The User-Defined Tools | Customize command opens the Tools tab of the Customize dialog (in which you can create the custom commands that appear in the menu of the User-Defined Tools command.)

23.13.7 Global Resources
The Global Resources command pops up the Global Resources dialog (screenshot below), in which you can:

- Specify the Global Resources XML File to use for global resources.
- Add file, folder, and database global resources (or aliases)
- Specify various configurations for each global resource (alias). Each configuration maps to a specific resource.
How to define global resources is described in detail in the section, Defining Global Resources.

Note: The Altova Global Resources dialog can also be accessed via the Global Resources toolbar (Tools | Customize | Toolbars | Global Resources).

23.13.8 Active Configuration

Mousing over the Active Configuration menu item rolls out a submenu containing all the configurations defined in the currently active Global Resources XML File (screenshot below).

The currently active configuration is indicated with a bullet. In the screenshot above the currently active configuration is Default. To change the active configuration, select the configuration you wish to make active.

Note: The active configuration can also be selected via the Global Resources toolbar (Tools | Customize | Toolbars | Global Resources).

23.13.9 Manage Raptor Servers

The Manage Raptor Servers command enables you to add multiple Raptor Servers to the pool of available Raptor Servers and to then define multiple configurations for each server. For an
overview of how to use RaptorXML Server for validating XML and XBRL documents, see the topic Validating with RaptorXML Server.

Adding a Raptor Server

In the dialog’s Servers pane (screenshot below), click the Add Server icon, then enter the name by which you wish to identify the Raptor server, the network name of the machine on which Raptor is installed (host name), and the port of the Raptor Server. Click OK to save the settings.
Name: Any string you choose. It is used in XMLSpy to identify a particular RaptorXML Server.

Host name: The name or IP address of the network machine on which the Raptor server is installed. Processing will be faster if you use an IP address rather than a host name. The IP address corresponding to localhost (the local machine) is 127.0.0.1.

Port: The port via which the Raptor server is accessed. This port is specified in Raptor's configuration file (called server_config.xml). The port must be fixed and known so that requests can be correctly addressed to the service. For more information about the Raptor configuration file, see the user manuals: RaptorXML Server and RaptorXML+XBRL Server.

After entering the server information, click OK. The server name you entered appears in the server list (in the left of the pane). A green icon appears next to the server's name, indicating that the Raptor server has been started and is running. The details of the server are displayed in the pane (see screenshot above). A red icon indicates that the server is offline. If the server cannot be found, an error message is displayed.

Note: The Raptor server must be running when the server is added. This is necessary so that XMLSpy can obtain information about the server and store it. If, after the server has been added, the server is offline or cannot be found, then these situations are indicated, respectively, by a red icon or an error message.

To edit a server's name, host name, or port, select the server in the left-hand pane, click the Edit button, and, in the dialog that appears, edit the information you want to change. To remove a server from the pool, select the server and click the Remove Selected Server icon.

Server Configurations
A configuration is a set of RaptorXML validation options. When a server is added, it will have a configuration named default. This is a set of RaptorXML options set to their default values. You can edit these values. You can also add new configurations that contain other option values. After you have defined multiple server configurations, you can select one configuration to be the active configuration. This is the configuration that will be used when the Validate on Server command is executed.

The Configurations pane has two parts: (i) a left-hand pane, which shows the configurations and the types of document that can be validated; (ii) a right-hand pane, which displays the options, organized in groups, of the validation type that is selected in the left-hand pane; at the bottom of the right-hand pane is a description of the selected option (see screenshot above).
**Adding a configuration**

In the Configurations pane of the RaptorXML Server Options dialog (screenshot above), click **Add a Configuration**. A new configuration is added with default option values. You can also create a new configuration by clicking **Copy Selected Configuration**. This creates a new configuration with option values that are the same as that of the copied configuration. New configurations are created with default names of the type `config<X>`; you can edit the name of a configuration by double-clicking it and entering the new name. You can then edit any of the configuration's option values.

**Editing a configuration's option values**

First select the validation document in the left-hand pane. This displays the options of that group in the right-hand pane. To edit the value of an option, do one of the following (depending on the type of option value):

- If the value can be one of a set of predefined values, select the value you want from the combo box of that option's value column.
- If the value is not constrained, click in the option's value field and enter the value you want.
- If the value is a file path, in addition to being able to enter the value, you can also browse for the file you want by using the **Browse** button in the option's value column.

If you select an option, its description is displayed in a box at the bottom of the right-hand pane.
For more detailed descriptions of each option, see the command line interface chapters of the RaptorXML Server and RaptorXML(+XBRL) Server user manuals.

Removing a configuration
In the left-hand pane, select the configuration to be removed and click Remove Selected Configuration.

23.13.10 Raptor Servers and Configurations
If you have defined multiple configurations on multiple servers, you can select a server and one of its configurations as the active configuration. The active configuration will be used for subsequent validations. On placing the cursor over the Tools | Raptor Servers and Configurations command (see screenshot below), a submenu appears that contains all the added servers, together with the configuration of each. Select the server configuration you want to make the active configuration. In the screenshot below, the xbrl configuration of the server named Raptor-01 has been selected as the active configuration (indicated by the green arrow).

Note: You can also select the active configuration in the dropdown menu of the Validate on Server icon. This menu also has a command to validate EDGAR on the active server.

23.13.11 Customize
The Customize command lets you customize application menus and toolbars to suit your personal needs. Clicking the command pops up the Customize dialog, which has the following tabs:

- **Commands**: All application and macro commands can be dragged from this tab into menu bars, menus and toolbars.
- **Toolbars**: Toolbars can be activated, deactivated, and reset individually.
- **Tools**: Commands that open external programs from within the interface can be added to the interface.
- **Keyboard**: Keyboard shortcuts can be created for individual application and macro commands.
- **Menu**: Menu bars and context menus to be customized are selected and made active in this tab. Works together with the Commands tab.
- **Macros**: Macros can have new commands associated with them.
- **Plug-ins**: Plug-ins can be activated and integrated in the interface.
- **Options**: Display options for toolbars are set in this tab.
This section also describes the **context menu** that appears when the Customize dialog is open and menu bar, menu, or tool bar items are right-clicked.

### Commands

The **Commands** tab allows you customize your menus and toolbars. You can add application commands to menus and toolbars according to your preference. Note, however, that you cannot create new application commands or menus yourself.

To add a command to a toolbar or menu:

1. Select the menu item **Tools | Customize**. The Customize dialog appears.
2. Select the **All Commands** category in the **Categories** list box. The available commands appear in the **Commands** list box.
3. Click on a command in the **Commands** list box and drag it to an existing menu or toolbar. An **I**-beam appears when you place the cursor over a valid position to drop the command.
4. Release the mouse button at the position you want to insert the command.

Note the following points.
- When you drag a command, a small button appears at the tip of mouse pointer: This indicates that the command is currently being dragged.
- An "x" below the pointer indicates that the command cannot be dropped at the current cursor position.
- If the cursor is moved to a position at which the command can be dropped (a toolbar or menu), the "x" disappears and an **I**-beam indicates the valid position.
- Commands can be placed in menus or toolbars. If you have **created you own toolbar**, you can use this customization mechanism to populate it.
• Moving the cursor over a closed menu, opens that menu, allowing you to insert the command anywhere in that menu.

Adding commands to context menus
You can also add commands to context menus by dragging commands from the Commands list box into the context menu. The procedure is as follows:

1. In the Customize dialog, click the Menu tab.
2. In the Context Menu pane, select a context menu from the combo box. The selected context menu pops up.
3. In the Customize dialog, switch back to the Commands tab.
4. Drag the command you wish to create from the Commands list box and drop it into the desired location in the context menu.

Deleting a command or menu
To delete a command from a menu, context menu (see above for details of accessing context menus), or toolbar, or to delete an entire menu, do the following.

1. Open the Customize dialog (Tools | Customize). The Customize dialog appears.
2. With the Customize dialog open (and any tab selected), right-click a menu or a menu command, and then select Delete from the context menu that pops up. Alternatively, drag the menu or menu command till an "x" icon appears below the mouse pointer, and then drop the menu or menu command. The menu or menu command will be deleted.

To re-instate deleted menu commands, use the mechanisms described in this section. To re-instate a deleted menu, go to Tools | Customize | Menu, and click the Reset button in the Application Frame Menus pane. Alternatively, go to Tools | Customize | Toolbars, select Menu Bar, and click the Reset button.

Toolbars
The Toolbars tab allows you: (i) to activate or deactivate specific toolbars (that is, to decide which ones to display in the interface); (ii) to set what icons are displayed in each toolbar; and (iii) to create your own specialized toolbars.

The toolbars contain icons for the most frequently used menu commands. Information about each icon is displayed in a tooltip and in the Status Bar when the cursor is placed over the icon. You can drag a toolbar to any location on the screen, where it will appear as a floating window.

Note: To add a command to a toolbar, drag the command you want from the Commands list box in the Commands tab to the toolbar. To delete a command from a toolbar, open the Customize dialog, and with any tab selected, drag the command out of the toolbar (see Commands for more details).

Note: Toolbar settings defined in a particular view are, by default, valid for that view only. To make the settings apply to all views, click the check box at the bottom of the dialog.
The following functionality is available:

- **To activate or deactivate a toolbar**: Click its check box in the Toolbars list box.

- **To apply changes to all views**: Click the check box at the bottom of the dialog. Otherwise, changes are applied only to the active view. Note that only changes made after clicking the All Views check box will apply to all views.

- **To add a new toolbar**: Click the New button and give the toolbar a name in the Toolbar Name dialog that pops up. From the Commands tab drag commands into the new toolbar.

- **To change the name of an added toolbar**: Select the added toolbar in the Toolbars pane, click the Rename button, and edit the name in the Toolbar Name dialog that pops up.

- **To reset the Menu bar**: Select the Menu Bar item in the Toolbars pane, and then click Reset. This resets the Menu bar to the state it was in when the application was installed.

- **To reset all toolbar and menu commands**: Click the Reset All button. This resets all toolbars and menus to the states they were in when the application was installed.

- **To delete a toolbar**: Select the toolbar you wish to delete in the Toolbars pane and click Delete.

- **To show text labels of commands in a particular toolbar**: Select that toolbar and click the Show Text Labels check box. Note that text labels have to be activated for each toolbar separately.
**Tools**

The **Tools** tab allows you to set up commands to use external applications from within XMLSpy. These commands will be added to the **Tools | User-defined Tools** menu. For example, the active file in the main window of XMLSpy can be opened in an external application, such as Notepad, by clicking a command in the **Tools | User-defined Tools** menu that you created.

To set up a command to use an external application, do the following:

1. In the **Menu Contents** pane (see screenshot above), click the **New** icon in the title bar of the pane and, in the item line that is created, enter the name of the menu command you want. In the screenshot above, we have entered a single menu command, **Open in Notepad**. We plan to use this command to open the active document in the external Notepad application. More commands can be added to the command list by clicking the **New** icon. A command can be moved up or down the list relative to other commands by using the **Move Item Up** and **Move Item Down** icons. To delete a command, select it and click the **Delete** icon.

2. To associate an external application with a command, select the command in the **Menu Contents** pane. Then, in the **Command** field, enter the path to, or browse for, the executable file of the external application. In the screenshot above, the path to the Notepad application has been entered in the **Command** field.

3. The actions available to be performed with the external application are displayed when you click the flyout button of the **Arguments** field (see screenshot above). These actions are described in the list below. When you select an action, a code string for the action is entered in the **Arguments** field.

4. If you wish to specify a current working directory, enter it in the **Initial Directory** field.

5. Click **Close** to finish.

The command/s you created will appear in the **Tools | User-defined Tools** menu, and in the context menu of Project window files and folders—in the **User-defined Tools** submenu.
When you click the command (in the **Tools | User-defined Tools** menu) that you created, the action you associated with the command will be executed. The command example shown in the screenshot above does the following: It opens, in Notepad, the document that is active in the Main Window of XMLSpy. The external application command is also available in the context menu of files in the Project window (right-click a file in the Project window to display that file’s context menu). Via the Project Window you can also open multiple files (for applications that allow this) by making a multi-selection and then selecting the command from the context menu.

### Arguments

The *Arguments* field specifies the action to be executed by the external application command. The following arguments are available.

- **Active Document File Path**: The command in the **Tools | User-defined Tools** menu opens the document that is active in XMLSpy in the external application. The command in the context menu of a file in the Project window opens the selected file in the external application.
- **Project File Path**: Opens the XMLSpy project file (the *.spp* file) in the external application.

### Initial directory

The *Initial Directory* entry is optional and is a path that will be used as the current directory.

### Keyboard

The *Keyboard* tab allows you to create new keyboard shortcuts, or change existing shortcuts, for any application command.
To assign a new shortcut to a command, or to change an existing shortcut, do the following.

1. Select the All Commands category in the Category combo box. Note that if a macro has been selected as an Associated Command, then macros are also available for selection in the Category combo box and a shortcut for the macro can be set.
2. In the Commands list box, select the command to which you wish to assign a new shortcut or select the command the shortcut of which you wish to change.
3. Click in the Press New Shortcut Key text box, and press the shortcut you wish to assign to that command. The shortcut appears in the Press New Shortcut Key text box. If the shortcut has not yet been assigned to any command, the Assign button is enabled. If the shortcut has already been assigned to a command, then that command is displayed below the text box and the Assign button is disabled. (To clear the Press New Shortcut Key text box, press any of the control keys, Ctrl, Alt or Shift).
4. Click the Assign button to assign the shortcut. The shortcut now appears in the Current Keys list box. You can assign multiple shortcuts to a single command.
5. Click the Close button to confirm.

Deleting a shortcut
A shortcut cannot be assigned to multiple commands. If you wish to delete a shortcut, click it in the Current Keys list box and then click the Remove button.

Set accelerator for
Currently, accelerators can be set only as default. No other mode is available.
Default keyboard shortcuts

The default shortcuts of commonly used commands are listed below. An overview of all the application's menu commands is available in the Keyboard Map (Help | Keyboard Map).

- **Function-key shortcuts (incl. for validation and transformation)**

<table>
<thead>
<tr>
<th>Key</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>Help Menu</td>
</tr>
<tr>
<td>F1 + Alt</td>
<td>Open Last File</td>
</tr>
<tr>
<td>F3</td>
<td>Find Next</td>
</tr>
<tr>
<td>F4 + CTRL</td>
<td>Close Active Window</td>
</tr>
<tr>
<td>F4 + Alt</td>
<td>Close XMLSpy</td>
</tr>
<tr>
<td>F5</td>
<td>Refresh</td>
</tr>
<tr>
<td>F6 + CTRL</td>
<td>Cycle through Open Windows</td>
</tr>
<tr>
<td>F7</td>
<td>Check Well-formedness</td>
</tr>
<tr>
<td>F8</td>
<td>Validate</td>
</tr>
<tr>
<td>F10</td>
<td>XSL Transformation</td>
</tr>
<tr>
<td>F10 + CTRL</td>
<td>XSL-FO Transformation</td>
</tr>
</tbody>
</table>

- **File and Application commands**

<table>
<thead>
<tr>
<th>Key</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt + F1</td>
<td>Open Last File</td>
</tr>
<tr>
<td>CTRL + O</td>
<td>File Open</td>
</tr>
<tr>
<td>CTRL + N</td>
<td>File New</td>
</tr>
<tr>
<td>CTRL + P</td>
<td>File Print</td>
</tr>
<tr>
<td>CTRL + S</td>
<td>File Save</td>
</tr>
<tr>
<td>CTRL + F4</td>
<td>Close Active Window</td>
</tr>
<tr>
<td>CTRL + F6</td>
<td>Cycle through Open Windows</td>
</tr>
<tr>
<td>CTRL + TAB</td>
<td>Switch between Open Documents</td>
</tr>
<tr>
<td>Alt + F4</td>
<td>Close XMLSpy</td>
</tr>
</tbody>
</table>

- **Numeric keypad shortcuts (to expand/collapse nodes)**

<table>
<thead>
<tr>
<th>Key</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num +</td>
<td>Expand</td>
</tr>
<tr>
<td>Num *</td>
<td>Expand Fully</td>
</tr>
<tr>
<td>Num –</td>
<td>Collapse</td>
</tr>
<tr>
<td>CTRL + Num –</td>
<td>Collapse Unselected</td>
</tr>
</tbody>
</table>

- **Miscellaneous keys**

<table>
<thead>
<tr>
<th>Key</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up/Down Arrow Keys</td>
<td>Move Cursor or Selection Bar</td>
</tr>
<tr>
<td>Esc</td>
<td>Abandon Edits or Close Dialog Box</td>
</tr>
<tr>
<td>Return</td>
<td>Confirm Selection</td>
</tr>
</tbody>
</table>
## Editing commands

<table>
<thead>
<tr>
<th>Shortcuts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL + A</td>
<td>Select All</td>
</tr>
<tr>
<td>CTRL + F</td>
<td>Find</td>
</tr>
<tr>
<td>CTRL + G</td>
<td>Go to Line/Char</td>
</tr>
<tr>
<td>CTRL + H</td>
<td>Replace</td>
</tr>
<tr>
<td>CTRL + V</td>
<td>Paste</td>
</tr>
<tr>
<td>CTRL + X</td>
<td>Cut</td>
</tr>
<tr>
<td>CTRL + Y</td>
<td>Redo</td>
</tr>
<tr>
<td>CTRL + Z</td>
<td>Undo</td>
</tr>
</tbody>
</table>

## Text View commands

<table>
<thead>
<tr>
<th>Shortcuts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL + E</td>
<td>Jump between Start/End Tags</td>
</tr>
<tr>
<td>CTRL + Shift + E</td>
<td>Select Element that Contains Cursor</td>
</tr>
<tr>
<td>CTRL + Alt + E</td>
<td>Go to Parent Element</td>
</tr>
<tr>
<td>CTRL + &quot;+&quot;</td>
<td>Zoom In</td>
</tr>
<tr>
<td>CTRL + &quot;-&quot;</td>
<td>Zoom Out</td>
</tr>
<tr>
<td>CTRL + 0</td>
<td>Reset Zoom</td>
</tr>
<tr>
<td>CTRL + mousewheel forwd</td>
<td>Zoom In</td>
</tr>
<tr>
<td>CTRL + mousewheel back</td>
<td>Zoom Out</td>
</tr>
</tbody>
</table>

## Grid View commands

<table>
<thead>
<tr>
<th>Shortcuts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL + D</td>
<td>Append CDATA</td>
</tr>
<tr>
<td>CTRL + E</td>
<td>Append Element</td>
</tr>
<tr>
<td>CTRL + I</td>
<td>Append Attribute</td>
</tr>
<tr>
<td>CTRL + M</td>
<td>Append Comment</td>
</tr>
<tr>
<td>CTRL + T</td>
<td>Append Text</td>
</tr>
<tr>
<td>CTRL + Shift + D</td>
<td>Insert CDATA</td>
</tr>
<tr>
<td>CTRL + Shift + E</td>
<td>Insert Element</td>
</tr>
<tr>
<td>CTRL + Shift + I</td>
<td>Insert Attribute</td>
</tr>
<tr>
<td>CTRL + Shift + M</td>
<td>Insert Comment</td>
</tr>
<tr>
<td>CTRL + Shift + T</td>
<td>Insert Text</td>
</tr>
<tr>
<td>CTRL + Alt + D</td>
<td>Add Child CDATA</td>
</tr>
<tr>
<td>CTRL + Alt + E</td>
<td>Add Child Element</td>
</tr>
<tr>
<td>Key Combination</td>
<td>Command</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>CTRL + Alt + I</td>
<td>Add Child Attribute</td>
</tr>
<tr>
<td>CTRL + Alt + M</td>
<td>Add Child Comment</td>
</tr>
<tr>
<td>CTRL + Alt + T</td>
<td>Add Child Text</td>
</tr>
</tbody>
</table>

**Schema View commands**

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL + Dbl-click Element</td>
<td>Display Element Definition</td>
</tr>
</tbody>
</table>

**Debugger commands**

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>F9</td>
<td>Insert/Remove Breakpoint</td>
</tr>
<tr>
<td>F9 + Shift</td>
<td>Insert/Remove Tracepoint</td>
</tr>
<tr>
<td>F9 + CTRL</td>
<td>Enable/Disable Breakpoint</td>
</tr>
<tr>
<td>F9 + Shift + CTRL</td>
<td>Enable/Disable Tracepoint</td>
</tr>
<tr>
<td>F11</td>
<td>Step Into</td>
</tr>
<tr>
<td>F11 + Shift</td>
<td>Step Out</td>
</tr>
<tr>
<td>F11 + CTRL</td>
<td>Step Over</td>
</tr>
<tr>
<td>F11 + Alt</td>
<td>Start Debugger/Go</td>
</tr>
</tbody>
</table>

**Menu**

The **Menu** tab allows you to customize the two main menu bars (default and application menu bars) as well as the application's context menus.
Customizing the default menu bar and application menu bar

The default menu bar is the menu bar that is displayed when no document is open in the main window. The application menu bar is the menu bar that is displayed when one or more documents are open in the main window. Each menu bar can be customized separately, and customization changes made to one do not affect the other.

To customize a menu bar, select it in the Show Menus For combo box (see screenshot above). Then switch to the Commands tab of the Customize dialog and drag commands from the Commands list box to the menu bar or into any of the menus.

Deleting commands from menus and resetting the menu bars

To delete an entire menu or a command inside a menu, select that menu or menu command, and then either (i) right-click and select Delete, or (ii) drag away from the menu bar or menu, respectively.

You can reset each of these two menu bars (default and application menu bars) to its original installation state by selecting the menu in the Show Menus For combo box and then clicking the Reset button below the combo box.

Customizing the application’s context menus

Context menus are the menus that appear when you right-click certain objects in the application’s interface. Each of these context menus can be customized by doing the following:

1. Select the context menu you want in the Select Context Menu combo box. This pops up
the context menu.
2. Switching to the Commands tab of the Customize dialog.
3. Drag a command from the Commands list box into the context menu.
4. If you wish to delete a command from the context menu, right-click that command in the context menu, and click Delete. Alternatively, you can drag the command you want to delete out of the context menu.

You can reset any context menu to its original installation state by selecting it in the Select Context Menu combo box and then clicking the Reset button below the combo box.

Menu shadows
Click the Menu shadows check box to give all menus shadows.

Macros
The Macros tab allows you to create application commands for macros that were created using XMLSpy's Scripting Editor. These application commands (which run the macros associated with them) can subsequently be made available in menus and toolbars, either from the Macros tab directly or by using the mechanisms available in the Commands tab of the Customize dialog. As application commands, they can also be assigned shortcuts in the Keyboard tab of the Customize dialog.

How macros work in XMLSpy
Macros in XMLSpy work as follows:

- Altova scripting projects (.asprj files) are created in XMLSpy's Scripting Editor. It is these scripting projects that can contain the macros used in XMLSpy.
- Two scripting projects can be active at a time in XMLSpy: (i) An application scripting project, which is specified in the Scripting section of the Options dialog, and (ii) The scripting project of the active XMLSpy project, which is specified in the Script Settings dialog (Project | Script Settings).
- The macros in these two scripting projects are available in the application: in the Project | Macros menu (from where the macros can be run), and in the Macros tab of the Customize dialog (screenshot below), in which they can be set as application commands. After a macro has been set as an application command, the command can be placed in a menu and/or toolbar.

Creating an application command for a macro
In Scripting Editor (Tools | Scripting Editor) create the macro you wish and save it to a scripting project. Specify this file to be either the application scripting project (via the Scripting section of the Options dialog) or the active application project's scripting project (via the application project's Script Settings dialog (Project | Script Settings)). The macros in the scripting project will now appear in the Macros pane of the Macros tab (see screenshot below).

To create an application command for a macro, select the macro in the Macros pane, set the text of the command in the Display Text text box, and click Add Command (see screenshot below). A command associated with the selected macro will be added to the Associated Commands list box.
To edit the icon of an associated command, select the command and click **Edit Icon**. To delete an associated command, click **Remove**.

Placing a macro-associated command in a menu or toolbar
There are two ways to place a macro-associated command in a menu or toolbar:

- Drag the command from the Associated Commands list box to the desired location in the menu or toolbar.
- Use the mechanisms available in the **Commands tab of the Customize dialog**.

In either case, the command will be created at the desired location. Clicking on the command in the menu or toolbar will execute the macro.

**Note:** If a macro has been set as an associated command, you can set a **keyboard shortcut for it**. In the **Keyboard tab of the Customize dialog**, select **Macros** in the **Category** combo box, then select the required macro, and set the shortcut. You must set a macro as an associated command in order for it to be available to be created as a keyboard shortcut.

**Plug-Ins**
The **Plug-Ins** tab allows you to integrate plug-ins and to place commands, where these have been so programmed, in an application menu and/or toolbar. In the Plug-In tab (screenshot below), click **Add Plug-In**, and browse for the plug-in's DLL file (see 'Creating plug-ins' below). Click **OK** to add the plug-in. Multiple plug-ins can be added.
After a plug-in has been added successfully, a description of the plug-in appears in the dialog and the **Remove Plug-In** button becomes enabled. If the plug-in code creates toolbars and menus, these will be immediately visible in the application interface. To remove a plug-in select it and click **Remove Plug-In**.

**Creating plug-ins**

Source code for sample plug-ins has been provided in the application's *(My) Documents folder: Examples\IDEPlugin* folder. To build a plug-in from such source code, do the following:

1. Open the solution you want to build as a plug-in in Visual Studio.
2. Build the plug-in with the command in the Build menu.
3. The plug-in's DLL file that will be created in the Bin or Debug folder. This DLL file is the file that must be added as a plug-in *(see above)*.

For more information about plug-ins, see the section **IDE Plugins**.
Options

The **Options** tab allows you to define general environment settings.

![Options Tab](image)

Click the check boxes to toggle on the following options:

- *Show ScreenTips on toolbars*: Displays a popup when the mouse pointer is placed over an icon in any toolbar. The popup contains a short description of the icon function, as well as the associated keyboard shortcut, if one has been assigned and if the *Show shortcut keys* option has been checked.
- *Show shortcut keys in ScreenTips*: Defines whether shortcut information will be shown in screen tips.
- *Large icons*: Toggles the size of toolbar icons between standard and large.

Customize Context Menu

The **Customize context menu** (screenshot below) is the menu that appears when you have the Customize dialog open and then right-click an application menu, menu command, or toolbar icon.
The following functionality is available:

- **Reset to Default**: Currently no function.
- **Copy Button Image**: Copies the icon you right-click to the clipboard.
- **Delete**: Deletes the selected menu, menu command, or toolbar icon. For information about how to restore deleted items, see below.
- **Button Appearance**: Pops up the Button Appearance dialog (see screenshot below), in which you can set properties that define the appearance of the selected toolbar icon. See the description below for details.
- **Image, Text, Image and Text**: Mutually exclusive options that determine whether the selected toolbar icon will be an icon only, text only, or both icon and text. You can select one of these options to make the change. Alternatively, you can make this change in the Button Appearance dialog.
- **Start Group**: Inserts a vertical group-divider to the left of the selected toolbar icon. This makes the selected toolbar icon the first of a group of icons.

**The Button Appearance dialog**

Right-click a toolbar icon and click **Button Appearance** to get the Button Appearance dialog (screenshot below). Via this dialog you can edit the toolbar icon image, as well as its text. Currently only toolbar icons for macros and from plug-ins can be edited using this dialog.
The following editing functionality is available for the selected toolbar icon (the one that was right-clicked to get the Customize context menu):

- **Image only, Text only, Image and text**: Select the desired radio button to specify what form the toolbar icon will take.
- **Image editing**: When Image only or Image and text is selected, then the image editing options are enabled. Click **New** to create a new image that will be added to the user-defined images in the images pane. Select an image and click **Edit** to edit.
Image selection: Select an image from the Images pane and click OK to use the selected image as the toolbar icon.

Text editing and selection: When Text only or Image and text is selected, then the Button Text text box is enabled. Enter or edit the text and click OK to make this the text of the toolbar icon.

Note: The Button Appearance dialog can also be used to edit the text of menu commands. Right-click the menu command (with the Customize dialog open), click Button Appearance, and then edit the menu command text in the Button Text text box.

Restoring deleted menus, menu commands, and toolbar icons

If a menu, menu command, or toolbar icon has been deleted by using the Delete command in the Customize context menu, these can be restored as follows:

- **Menus**: Go to Tools | Customize | Menu, and click the Reset button in the Application Frame Menus pane. Alternatively, go to Tools | Customize | Toolbars, select Menu Bar, and click the Reset button.
- **Menu commands**: Go to Tools | Customize | Commands, and drag the command from the Commands list box into the menu.
- **Toolbar icons**: Go to Tools | Customize | Commands, and drag the command from the Commands list box into the toolbar.
23.13.12 Restore Toolbars and Windows

The Restore Toolbars and Windows command closes down XMLSpy and re-starts it with the default settings. Before it closes down a dialog pops up asking for confirmation about whether XMLSpy should be closed (screenshot below).

![Screenshot of the Restore Toolbars and Windows dialog]

This command is useful if you have been resizing, moving, or hiding toolbars or windows, and would now like to have all the toolbars and windows as they originally were.

23.13.13 Options

The Tools | Options command enables you to define global application settings. These settings are organized in sections (see left pane in screenshot below). For example, the File section (shown in the screenshot below) contains options that specify how you want XMLSpy to open and save files. To specify options of a particular section, select that section in the left pane and specify the property values you want. The OK button saves changes to the registry and closes the dialog. The Apply button causes changes to be displayed in currently open documents.
Each section of the Options dialog is described in detail in its sub-section of this section.

**File**

The File section defines the way XMLSpy opens and saves documents. Related settings are in the Encoding section.
Open/New file in Grid view
You can choose to open an existing file or create a new file either in Grid View or in Text View. If you select Grid View, you can also choose to automatically expand all lines.

Automatic reload of changed files
If you are working in a multi-user environment, or if you are working on files that are dynamically generated on a server, you can watch for changes to files that are currently open in the interface. Each time XMLSpy detects a change in an open document, it will prompt you about whether you want to reload the changed file.

Validation
If you are using DTDs or schemas to define the structure of your XML documents, you can automatically check the document for validity whenever it is opened or saved. During Open and Save operations, you have the option of validating files only if the file-size is less than a size you specify in MB. If the document is not valid, an error message will be displayed. If it is valid, no message will be displayed and the operation will proceed without any notification. XMLSpy can also cache these files in memory to save any unnecessary reloading (e.g. when the schema being referred to is accessed through a URL). If your schema location declaration uses an URL, disable the "cache DTD/Schema files in memory" option to have changes made to the schema appear immediately, and not use the cached version of the schema.

XML Schema Version
The XSD mode that is enabled in Schema View depends on both (i) the presence/absence—and, if present, the value—of the /xs:schema/@vc:minVersion attribute of the XSD document, and (ii)
the XML Schema Version option selected in the File section of the Options dialog (Tools | Options, screenshot below).

The following situations are possible. XML Schema Version in the table below refers to the selection in the XML Schema Version pane shown above. The vc:minVersion values in the table refer to the value of the xs:schema/@vc:minVersion attribute in the XML Schema document. For more details, see the section Editing Views | Schema View | XSDMode.

<table>
<thead>
<tr>
<th>XML Schema Version</th>
<th>vc:minVersion attribute</th>
<th>XSD mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always v1.0</td>
<td>Is absent, or is present with any value</td>
<td>1.0</td>
</tr>
<tr>
<td>Always v1.1</td>
<td>Is absent, or is present with any value</td>
<td>1.1</td>
</tr>
<tr>
<td>Value of @vc:minVersion</td>
<td>Attribute has value of 1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>Value of @vc:minVersion</td>
<td>Attribute is absent, or attribute is present with a value other than 1.1</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Project
When you start XMLSpy, you can open the last-used project automatically.

Save File
When saving an XML document, XMLSpy includes a short comment <!-- Edited with XMLSpy http://www.altova.com --> near the top of the file. This option can only be deactivated by licensed users, and takes effect when editing or saving files in the Enhanced Grid or Schema Design View.

When saving a content model diagram (using the menu option Schema design | Generate Documentation), XMLSpy includes the XMLSpy logo. This option can only be deactivated by licensed users.

If a StyleVision Power Stylesheet is associated with an XML file, the 'Authentic: save link to design file' option will cause the link to the StyleVision Power Stylesheet to be saved with the XML file.

Line breaks
When you open a file, the character coding for line breaks in it are preserved if Preserve old is selected. Alternatively, you can choose to code line breaks in any of three codings: CR&LF (for PC), CR (for MacOS), or LF (for Unix).
No output formatting for

In Text View, the indentation of an element can be made to reflect its position in the element hierarchy (see **Save File**). You can, however, override this indentation for individual elements. To do this, enter the element name in the **No output formatting for** field. All elements entered in this field will be formatted such that their descendant elements have no whitespace between them (see **screenshots**).

Hierarchical indentation for all elements:

```
11  <xs:simpleType>
12  | <xs:restriction base="xs:string">
13  |   | <xs:maxLength value="255"/>
14  | </xs:restriction>
15  </xs:simpleType>
```

**No output formatting** has been specified for element **xs:restriction**:

```
11  <xs:simpleType>
12  | <xs:restriction base="xs:string">xs:maxLength value="255"</xs:restriction>
13  </xs:simpleType>
```

**Save and exit**

After making the settings, click **OK** to finish.

**File Types**

The **File Types** section (**screenshot below**) allows you to customize the behavior of XMLSpy on a per-file-type basis. (**Note that, in the screenshot below, some options, such as those related to XBRL and Avro support, are available in the Enterprise Edition only.**)
Choose a file type from the File Types list box, and then customize the functions for that particular file type as described below.

**Windows Explorer settings**
You can define the file type description and MIME-compliant content type used by Windows Explorer and whether XMLSpy is to be the default editor for documents of this file type.

**Conformance**
XMLSpy provides specific intelligent editing features, as well as other features, for different file types. XMLSpy sets the features for a particular file type on the basis of the conformance you set in this option. For example, in the screenshot above, files with the `.xqu` file extension are set to be conformant to XQuery Update. XMLSpy will therefore open `.xqu` files with XQuery Update editing support. XMLSpy lets you set the following conformance options: XML, XQuery, ZIP, JSON, other formats. XML conformance is further differentiated between XML, DTD, and XML Entity file types. A large number of file types are defined with a default conformance that is appropriate for the file type. We recommend that you do not modify these settings unless you are adding a new file type or deliberately wish to set a file type to another kind of conformance.

**Default view**
This group lets you define the default view to be used for each file type. The screenshot above shows the File types section of the Enterprise edition. If your edition is not the Enterprise edition, it will have fewer views than shown in the screenshot. If a particular conformance can be viewed in one view only, then that view is selected by default and view selection is disabled. For example, XQuery Update documents can only be viewed in Text View, so this view is selected by default and view selection is disabled.
**Grid View**
This check box lets you define whether the Grid View should automatically build tables.

**Text View**
This check box lets you set syntax-coloring for particular file types.

**Disable automatic validation**
This option enables you to disable automatic validation per file type. Automatic validation typically takes place when a file is opened or saved, or when a view is changed.

**Save empty elements in short <E/> format**
Some applications that use XML documents or output generated from XML documents may have problems understanding the short `<Element/>` form for empty elements defined in the XML 1.0 Specification. You can instruct XMLSpy to save elements in the longer (but also valid) `<Element></Element>` form.

**Use RaptorXML Server to validate on Open/Save**
Specifies whether RaptorXML Server should be used to validate files of the selected file type when the file is opened and saved. For this to work, a RaptorXML Server must be set up and configured.

**Add new file extension**
Adds a new file type to the File types list. You must then define the settings for this new file type using the other options in this tab.

**Delete selected file extension**
Deletes the currently selected file type and all its associated settings.

**Save and exit**
After making the settings, click OK to finish.
Encoding

The **Encoding** section specifies options for file encodings.

**Default encoding for new XML files**

The default encoding for new XML files can be set by selecting an option from the dropdown list. A new document is created with an XML declaration containing the encoding value you specify here. If a two- or four-byte encoding is selected as the default encoding (i.e. UTF-16, UCS-2, or UCS-4) you can also choose between little-endian and big-endian byte-ordering.

The encoding of existing XML files will be retained and can only be changed with the **File | Encoding** command.

**Open XML files with unknown encoding as**

If the encoding of an XML file cannot be determined or if the XML document has no encoding specification, the file will be opened with the encoding you select in this combo box.

**Open non-XML files in**

Existing and new non-XML files are opened with the encoding you select in this combo box. You can change the encoding of the document by using the **File | Encoding** command.

**BOM (Byte Order Mark)**

When a document with two-byte or four-byte character encoding is saved, the document can be saved either with (i) little-endian byte-ordering and a little-endian BOM (**Always create BOM if not UTF-8**); or (ii) the detected byte-ordering and the detected BOM (**Preserve detected BOM on saving**).

**Save and exit**

After making the settings, click **OK** to finish.
Editing

The Editing section enables you to specify editing behaviour in XMLSpy.

Intelligent editing

While editing documents, XMLSpy provides intelligent editing based on these settings. You can also customize various aspects of the Entry Helper behavior in this pane. The customization settings made here will be applied when relevant to the file type being edited. For example, the option to load entry helpers on opening the file and sorting attributes will not be applicable to DTD or XQuery documents.

Mandatory child elements of choice groups are auto-appended on the basis of the setting made in this pane. You can select whether (i) the first branch (element) of the choice group, (ii) all branches, or (iii) the branch with the smallest number of descendant elements is generated. Note that the All branches selection could generate an invalid document since only one branch from a choice group is allowed.

Text View

The Auto-complete option automatically adds unambiguous structural components. For example, when the closing angular bracket of the start tag of an element is entered, then the end tag of that element is automatically added if this option is enabled.

In Text View, Auto-completion and entry helpers can be disabled if a file is bigger than the size...
specified in the Disable Auto-completion... combo box. This is useful if you wish to speed up the editing of large files and can do without the auto-completion feature and entry helpers. If the file size is bigger than that specified for this option, then the Text View context menu contains a toggle command for switching on and off Auto-completion and entry helper use. So you can always switch these editing aids on and off at any time during editing (in the event of files having a size greater than the size specified for this option). If the value specified for this option is smaller than the size of the opened file, locations indicated in error messages will not correctly correspond to the location in Text View.

Default copy to clipboard in grid view as
You can choose the format in which data will be exported to foreign applications using the clipboard. If you select XML-Text, the contents of the clipboard will be formatted and tagged just like the resulting XML file itself. The structured text mode attempts to format the clipboard contents as a table, for use in a spreadsheet or database application. This option does not affect the internal clipboard format that XMLSpy uses for copying and pasting.

Table view
You can also control, how XMLSpy decides when to display repeating elements in the Table View.

Save and exit
After making the settings, click OK to finish.

View
The View section enables you to customize the XML documents presentation in XMLSpy.

Grid View
XML elements in Grid View can be collapsed into a single line displaying the element name. When collapsed, the element's attributes can also be displayed in that line. If the Show Attribute Previews option is checked, attributes are displayed in gray with collapsed elements. Otherwise, attributes are not displayed with the collapsed element. Columns in the grid can be set to adjust automatically to optimal widths. Additionally, the maximum optimal width and cell height can be limited. If the content of a cell is more than can fit in a cell, this is indicated by an ellipsis.

Pretty-print
When you select Edit | Pretty-Print XML Text in Text View or switch from another view to Text View, the XML document will be "pretty-printed". The pretty-printing will be with or without indentation according to whether the Use Indentation option in this dialog is checked or not. The amount of indentation can be specified in the Tabs pane of the Text View Settings dialog.

Program logo
You can turn off the splash screen on program startup to speed up the application. Also, if you have a purchased license (as opposed to, say, a trial license), you will have the option of turning off the program logo, copyright notice, and registration details when printing a document from XMLSpy.
Window title
The window title for each document window can contain either the file name only or the full path name.

Authentic View
XML files based on a StyleVision Power Stylesheet are automatically opened in the Authentic View when this option is active.

Browser View
You can choose to see the browser view in a separate window, enabling side-by-side placement of the edit and browser views.

Schema view
An XML Schema datatype can be derived from another datatype. For example, a datatype for E-mail elements can be derived from a base datatype of xs:string (for example, by restricting the xs:string datatype to a specific set of characters). If the base datatype is subsequently changed, you can set the following options:

- **Preserve content**: If the definitions used to define the derived type can be used with the new base type, checking this option will automatically preserve the definitions.
- **Confirm on every modification**: After changing the base type, a dialog (see screenshot below) will pop up asking whether the old definitions should be preserved and used with the new base type.

```
Base Type Modification

The component "Phone" contains facets, which still can be used in combination with the new base type "xs:normalizedString". Do you want to preserve them?

☑ Request confirmation before every base type modification

[ ] Yes [ ] No
```

Save and exit
After making the settings, click OK to finish.

Fonts and Colors
The Fonts and Colors section provides customization options for the appearance of text items in the various views of XMLSpy.
Options for the following views are available:

- **Text View**
- **Grid View**
- **Schema Design View**

In the left-hand pane of the dialog, select the view what you want to customize. The text item types that can be formatted are displayed in the right-hand pane (screenshot above). Select the text item type that you want to format, and then assign to it the desired formatting property values.

**Text View**

The **Text View** section enables you to customize the appearance of individual text-item types in various types of documents (see screenshot below and list of documents in Document Types section below). For example, you can specify different formatting for the display of element names and attribute names in XML documents, or for the display of keywords and variables in XQuery documents.
Document types
Text View items of the following types of documents can be formatted:

- XML generic
- XQuery
- CSS
- JSON
- C-family
- Python
- Miscellaneous
- Output

How to customize
To customize individual text item types of a particular document type, do the following:

1. In the combo box at top left, select the type of document for which you wish to customize text. On doing this, the text item types of that document type appear in the box below the combo box. (In the screenshot above, XML generic has been selected as the document type.)

2. Select the text item type you wish to customize by clicking it. (In the screenshot above, Element names has been selected.)

3. Set the font properties of the selected text item type by using the options in the panes on the right-hand side. The Text View Background option enables the selection of a background color for the entire Text View.

Note the following points:

- The same font face, size, and style are used for all text item types of a particular
document type (such as the XML generic document type). Within a document type, only
the text color and background color of individual text types can be changed. This enables
the syntax coloring feature.

- In the **Generic XML** category, the **Element names** text type consists of three subtypes:
  (i) **Element names** applies to element names that are not selected; (ii) **Element names - match** applies to those element names that are selected (names in which the cursor is placed) and where the start tag name matches the end tag name; (i) **Element names - match error** applies to those element names that are selected, but where the start tag name matches the end tag name. Element names that are being edited will therefore be highlighted with different background colors according to whether the start tag names match the end tag names or not. These highlight colors can be changed by changing the respective background colors. Highlighting is turned on by default, and can be turned off by deselecting the **Highlight elements** option in the **Text View Settings** dialog.

- In the **Miscellaneous** category: (i) **Selection** refers to the currently selected text content; **inactive selection** refers to other occurrences in the document of the same text content; (ii) **Find active marker** refers to the currently selected occurrence of a search result, whereas **Find marker** refers to other (inactive) occurrences of the search result; (iii) **Debug/Call marker** refers to the currently selected step in a debug session; (iv) **Visible Whitespace** refers to the whitespace markers in a document. The whitespace in a document can be made visible by **switching on whitespace markers**.

**Set defaults**
The **Set Defaults** button resets fonts to the original installation settings.

**Save and exit**
After making the settings, click **OK** to finish.

**Grid View**
The **Grid View** section (screenshot below) allows you to customize the appearance of text in the **Grid View of XML documents (XML View and XML Table View)**. In the combo box (see screenshot), select the document display for which you want to configure Grid View. Then select the text item type you want to format and assign to it the desired formatting properties (listed below).
XML View
You can set the following properties for the selected text item type:

- **Font face and size**: The selected font will also be used in printouts of Grid View. If you want to use the same font face or size for all text item types, check the respective *Use the same for all* check box.
- **Text style, text color, text background**: Sets the style, color, and background of individual text item types. The current settings are immediately reflected in the list in the left-hand pane, so you can preview how the text item will look.

XML Table View
The display of [XML Table View](#) can additionally be configured to differentiate (i) selected headers from unselected headers, and (ii) selected empty cells from unselected empty cells. (Highlighting empty cells serves as a visual aid for quickly identifying missing content.) The differentiation is achieved by assigning different background colors to each item (**see screenshot below**).
The **Header Unselected** and **Header Selected** options refer to column and row headers. The screenshot below shows headers unselected; its background color is as set in the dialog above.

The **Header Selected** color is activated when all headers are selected—not when an individual header is selected. The screenshot below uses the background colors defined in the dialog shown above. All headers can be selected by clicking the cell that intersects the column and row headers, or by selecting the element created as the table—or any of its ancestors.

The **Missing Content** settings refer to non-existent elements or attributes in the XML.
Different background colors can be set for the corresponding table cells according to whether these cells are selected or unselected. In the screenshot below, the first row is selected.

<table>
<thead>
<tr>
<th>Person</th>
<th>First</th>
<th>Last</th>
<th>Title</th>
<th>PhoneExt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Vernon</td>
<td>Callaby</td>
<td>[Selected]</td>
<td>562</td>
</tr>
<tr>
<td>2</td>
<td>Frank</td>
<td>Further</td>
<td>Accounts Receivable</td>
<td>471</td>
</tr>
<tr>
<td>3</td>
<td>Loby</td>
<td>Matise</td>
<td>[Selected]</td>
<td>963</td>
</tr>
</tbody>
</table>

**Note:** In addition to the colors you define here, XMLSpy uses the regular selection and menu color preferences set in the Display Settings in the Control Panel of your Windows installation.

**Set Defaults**

The **Set Defaults** button resets fonts to the original installation settings.

**Save and exit**

After making the settings, click **OK** to finish.

**Schema Design View**

The **Schema Design View** section enables you to customize the appearance of the **Schema View** display of **XML Schema** and **JSON Schema** documents.
How to customize

To customize individual text item types of the selected document type, do the following:

1. In the combo box at top left, select XML Schema or JSON Schema. The text item types of the selected document type appear in the box below the combo box.
2. Select the text item type you wish to format by clicking it.
3. Set the font properties of the selected text item type by using the options in the panes on the right-hand side.

**Note:**
The Doc.Schema Header(2) and Doc.Element Header(2) text item types refer, respectively, to the schema header and to the element headers in the generated documentation of the schema. Compare the colors of these properties in the dialog above with the colors of the schema header and element header of the generated documentation in the screenshot below.
Set defaults
The Set Defaults button resets fonts to the original installation settings.

Save and exit
After making the settings, click OK to finish.
**XSL**

The **XSL** section (screenshot below) enables you to define options for **XSLT transformations** and **XSL-FO transformations** carried out from within the application.

**XSLT transformations**

XMLSpy contains the Altova RaptorXML XSLT 1.0, XSLT 2.0, and XSLT 3.0 engines, which you can use for XSLT transformations as well as for validating the XML files used in transformations. The appropriate XSLT engine (1.0, 2.0, or 3.0) is used (according to the value of the **version** attribute of the **xsl:stylesheet** or **xsl:transform** element). This applies both for XSLT transformations as well as for XSLT debugging using XMLSpy’s XSLT/XQuery Debugger.

For transforming XML documents using XSLT, you could use one of the following:

- The built-in Altova XSLT Engines (XSLT 1.0, XSLT 2.0, and XSLT 3.0).
- The MSXML 3.0, 4.0, or 6.0 parser (which is pre-installed). If you know which version of the MSXML parser is running on your machine, you could select it; otherwise, you should let the application select the version automatically. (The **Choose version automatically** option is active by default.) In this case, the application tries to select the most recent available version.
- An external XSLT processor of your choice. You must specify the command line string that the external XSLT processor uses to run a transformation. The following variables are available for building the command line string:

  - `%1` = XML document to process
  - `%2` = Output file to generate
  - `%3` = XSLT stylesheet to use (if the XML document does not contain a reference to a
stylesheet)

For example, say you have a processor that uses the following command to run an XSLT transformation:

```
myxsltengine.exe -o output.xml input.xml stylesheet.xslt parameter-name=parameter-value
```

Then, in XMLSpy, you can use the variables listed above to generate this command. Select the External XSL Transformation Program radio button, and enter the following line in the text box:

```
c:\myxsltengine\myxsltengine.exe -o %2 %1 %3 parameter-name=parameter-value
```

Check the respective check boxes to show the output and error messages of the external program in the Messages Window in XMLSpy.

**Note:** The parameters set in XMLSpy’s XSLT Input Parameters dialog are passed to the internal Altova XSLT Engines only. They are not passed to any other XSLT Engine that is set up as the default XSLT processor.

The Reuse output window option causes subsequent transformations to display the result document in the same output window. If the XML file belongs to a project and Reuse output window option is disabled, the setting only takes effect if the Save in folder output file path (screenshot below) in the relevant project properties is also disabled.

XSL-FO transformations

FO documents are processed using an FO processor, and the path to the executable of the FO processor must be specified in the text box for the XSL-FO transformation engine. The transformation is carried out using the XSL/XQuery | XSL-FO Transformation menu command. If the source file (the active document when the command is executed in the IDE) is an XSL-FO document, the FO processor is invoked for the transformation. If the source document is an XML document, an XSLT transformation is required to first convert the XML document to an XSL-FO document. This XSLT transformation can be carried out either by the XSLT engine you have specified as the default engine for the application (see above), or by the XSLT engine that might be built into the FO processor you have specified as the default FO processor for the application. To select between these two options, click the appropriate radio button.

After making the settings, click OK to finish.

**Note:** Unless you deselected the option to install the FOP processor of the Apache XML Project, it will have been installed in the folder C:\ProgramData\Altova\SharedBetweenVersions. If installed, the path to it will automatically have been entered in the XSL-FO Engine input box. You can set the path to any FO processor you wish to
use. Note, however, that the same path will be used by other Altova products that use FO processors and have settings to select the FO processor (StyleVision and Authentic Desktop).

**Save and exit**
After making the settings, click **OK** to finish.

**XQuery**
The XQuery section *(screenshot below)* defines options related to the editing and execution of XQuery and XQuery Update documents.

**General XQuery options**
The following options are available:

- **Serialization**: Serialization refers to the way in which text is written to the output document. You can choose the serialization method (adaptive*, HTML, JSON, text, XHTML, or XML) and serialization encoding of the output for different types of input. Output serialization can be selected separately for XML input and JSON input, separately. *(Note: The adaptive method enables an instance document to be processed*
without error; it automatically determines the serialization method on the basis of the input document.)

- **Omit XML declaration**: Omits the XML declaration in the serialized (output) document.
- **Indent output**: Indents the output document to show the document hierarchy.
- **Always skip XML source**: When an XQuery document is executed, XMLSpy can prompt for an XML source on which to execute the XQuery document. The prompt is a dialog that enables you to browse for the XML file. Select this option to skip this dialog and directly execute the XQuery document. If this option is selected, then the XQuery document should be able to execute correctly without being passed an XML document. This could be either because no XML document is required, or because XML data is accessed via functions within the XQuery document.
- **Validate XML files**: Validates XML files that are used in the execution of XQuery documents. Invalid XML files are flagged, and the XQuery document is not processed.
- **DB2 row retrieval**: In displays that show DB data, you can specify the maximum number of rows to be retrieved. XMLSpy recognizes .xqr file extensions as XQuery-for-DB files.
- **XQuery default version**: Specifies the XQuery engine version to use for execution of XQuery documents that do not have a version keyword. This applies to both XQuery and XQuery Update documents, and selects the default XQuery Engine to use.

**XQuery Update options**

The following XQuery Update options are available:

- **Updating**: When an XQuery Update file is executed, target XML files can either be updated directly on disk, or be opened in XMLSpy and updated in memory. The Open Files on Updating option enables you to review the updates and save the file to disk or reject the updates (by closing the file without saving).
- **Preserve original formatting**: Preserves the original formatting of the updated document as much as possible.

**Save and exit**

After making the settings, click OK to finish.

**JSON**

The JSON section (screenshot below) provides options for the following settings:

- Enable validation against the format attribute. The format attribute specifies the formatting of property values; for example, date or email formatting. If this option is checked, then the formats of property values are also validated; otherwise the formatting is ignored for validation.
Java

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, XMLSpy 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 XMLSpy.

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 XMLSpy 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 XMLSpy for the new settings to take effect.
take effect.

Changing the Java VM path affects the following areas:

- JDBC connectivity
- Java extension functions for XSLT/XPath

**Scripting**

The **Scripting** section *(screenshot below)* allows you to enable the Scripting Environment on application startup. Check the *Activate Scripting* check box to do this. You can then specify the Global Scripting Project file *(see screenshot below)*.

![Scripting screenshot]

To set a global scripting project for XMLSpy, check the *Activate Scripting* check box and then browse for the Altova Scripting Project (*.asprj) file you want. You can also specify: (i) whether Auto-Macros in the scripting project should be automatically executed when XMLSpy starts, and (ii) whether application event handler scripts in the project should be automatically executed or not; check or uncheck the respective check boxes accordingly.

**Save and exit**

After making the settings, click OK to finish. Macros in the Global Scripting Project will then be displayed in the submenu of the Macros command.

**Source Control**

The **Source Control** section *(screenshot below)* enables you to specify the source control provider, and the settings and default logon ID for each source control provider.
Source Control Plugin
The current source control plugin can be selected from among the currently installed source control systems. These systems are listed in the dropdown list of the combo box. After selecting the required source control, specify the login ID for it in the next text box. The Advanced button pops up a dialog specific to the selected source control plugin, in which you can define settings for that source control plugin. These settings are different for different source control plugins.

User preferences
A range of user preferences is available, including the following:

- Status updates can be performed in the background after a user-defined interval of time, or they can be switched off entirely. Very large source control databases could consume considerable CPU and network resources. The system can be speeded up, however, by disabling background status updates or increasing the interval between them.
- When opening and closing projects, files can be automatically checked out and checked in, respectively.
- The display of the Check Out and Check In dialogs can be suppressed.
- The Reset button is enabled if you have checked/activated the Don't show this again option in one of the dialog boxes. On clicking the Reset button, the Don't show this again prompt is re-enabled.

Save and exit
After making the settings, click OK to finish.
Network Proxy

The **Network Proxy** section (screenshot below) enables you to configure custom proxy settings. The default is to use the system's proxy settings, so the settings will work without user intervention. If you wish to specify an alternative network proxy, use one of the options to define the setting you want.

**Note:** The network proxy settings are shared among all Altova MissionKit applications. So, if you change the settings in one application, all MissionKit applications will be affected.

**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 (\*).

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).
23.14 Window Menu

To organize the individual document windows in an XMLSpy session, the **Window** menu contains standard commands common to most Windows applications.

You can cascade the open document windows, tile them, or arrange document icons once you have minimized them. You can also switch the various Entry Helper windows on or off, or switch to an open document window directly from the menu.

**23.14.1 Cascade**

This command rearranges all open document windows so that they are all cascaded (i.e. staggered) on top of each other.

**23.14.2 Tile Horizontally**

This command rearranges all open document windows as **horizontal tiles**, making them all visible at the same time.

**23.14.3 Tile Vertically**

This command rearranges all open document windows as **vertical tiles**, making them all visible at the same time.

**23.14.4 Project Window**

This command lets you switch the **Project Window** on or off.

This is a dockable window. Dragging on its title bar detaches it from its current position and makes it a floating window. Click right on the title bar, to allow docking or hide the window.
23.14.5 **Info Window**

This command lets you switch the *Info Window* on or off.

This is a dockable window. Dragging on its title bar detaches it from its current position and makes it a floating window. Click right on the title bar, to allow docking or hide the window.

23.14.6 **Entry Helpers**

This command lets you switch all three Entry-Helper Windows on or off.

All three Entry helpers are dockable windows. Dragging on a title bar detaches it from its current position and makes it a floating window. Click right on the title bar to allow docking or hide the window.

23.14.7 **Output Windows**

The Output Windows are a set of tabbed output windows, such as the Messages window (which displays messages like validation results), the Find in Files window, and the XPath window (which shows XPath evaluation results). The initial setting is for them to open at below the Main Window. The Output Windows command lets you switch the Output Windows on or off.

The Output Windows window is dockable. Dragging on its title bar detaches it from its current position and makes it a floating window. Click right on the title bar to allow docking or to hide the window.

For a complete description of Output Windows see [Output Windows](#) in the section, Text View.

23.14.8 **Project and Entry Helpers**

This command toggles on and off the display of the Project Window and the Entry Helpers together.

23.14.9 **All On/Off**

This command lets you switch all dockable windows on, or off:

- the *Project Window*
- the *Info Window*
- the three Entry-Helper Windows
- the *Output Windows*

This is useful if you want to hide all non-document windows quickly, to get the maximum viewing area for the document you are working on.
23.14.10 Currently Open Window List

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.
23.15 Help Menu

The Help menu contains commands required to get help or more information about XMLSpy, as well as links to information and support pages on the Altova web server.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents...</td>
<td>F1</td>
</tr>
<tr>
<td>Index...</td>
<td></td>
</tr>
<tr>
<td>Search...</td>
<td></td>
</tr>
<tr>
<td>Keyboard Map...</td>
<td></td>
</tr>
<tr>
<td>Software Activation...</td>
<td></td>
</tr>
<tr>
<td>Order Form...</td>
<td></td>
</tr>
<tr>
<td>Registration...</td>
<td></td>
</tr>
<tr>
<td>Check for Updates...</td>
<td></td>
</tr>
<tr>
<td>Support Center...</td>
<td></td>
</tr>
<tr>
<td>FAQ on the Web...</td>
<td></td>
</tr>
<tr>
<td>Download Components and Free Tools...</td>
<td></td>
</tr>
<tr>
<td>XMLSpy on the Internet...</td>
<td></td>
</tr>
<tr>
<td>XMLSpy Training...</td>
<td></td>
</tr>
<tr>
<td>About XMLSpy...</td>
<td></td>
</tr>
</tbody>
</table>

The Help menu also contains the Registration dialog, which lets you enter your license key-code once you have purchased the product.

23.15.1 Table of Contents, Index, Search

- Table of Contents
  - Description
    Opens the onscreen help manual of XMLSpy 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 XMLSpy 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.
23.15.2 Keyboard Map

The **Help | Keyboard Map** command causes an information box to be displayed that contains a menu-by-menu listing of all commands in XMLSpy. Menu commands are listed with a description and shortcut keystrokes for the command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Keys</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileClose</td>
<td></td>
<td>Close the active document</td>
</tr>
<tr>
<td>FileCloseAI</td>
<td></td>
<td>Close all open documents</td>
</tr>
<tr>
<td>FileEncoding...</td>
<td>Ctrl+N</td>
<td>Set or change the character set encoding for the current document</td>
</tr>
<tr>
<td>FileExit</td>
<td></td>
<td>Quit the application; prompts to save documents</td>
</tr>
<tr>
<td>FileNew...</td>
<td>Ctrl+N</td>
<td>Create a new document</td>
</tr>
<tr>
<td>FileOpen...</td>
<td>Ctrl+O</td>
<td>Open an existing document</td>
</tr>
<tr>
<td>FileOpenURL...</td>
<td>Ctrl+P</td>
<td>Open an existing document directly from a URL</td>
</tr>
<tr>
<td>FilePrint...</td>
<td>Ctrl+P</td>
<td>Print the active document</td>
</tr>
<tr>
<td>FilePrintPreview</td>
<td></td>
<td>Print Preview</td>
</tr>
<tr>
<td>FilePrintSetup...</td>
<td></td>
<td>Change the printer and printing options</td>
</tr>
<tr>
<td>FileReload</td>
<td></td>
<td>Reload open file</td>
</tr>
<tr>
<td>FileSave</td>
<td>Ctrl+S</td>
<td>Save the active document</td>
</tr>
<tr>
<td>FileSaveAI</td>
<td></td>
<td>Save all open documents</td>
</tr>
<tr>
<td>FileSaveAs...</td>
<td></td>
<td>Save the active document with a new name</td>
</tr>
<tr>
<td>FileSaveToURL...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FileSendByEmail...</td>
<td></td>
<td>Send document by e-mail</td>
</tr>
</tbody>
</table>

To view commands in a particular menu, select the menu name in the Category combo box. You can print the command by clicking the printer icon.

You should note the following points about shortcuts:

- Certain commands (and their shortcuts) are applicable only within a certain view. For example, most of the commands in the XML menu are applicable only in Grid View. Other commands (such as **File | Save** or **XML | Check Well-Formedness**) are available in multiple views.
- Other cool shortcuts: For example, **Shift+F10** brings up the context menu in Text View.
and Schema View; **Ctrl+E** when the cursor is inside an element start or end tag in Text View moves the cursor to the end or start tag, respectively.

- In the **Keyboard tab** of the Customize dialog, you can also set your own shortcuts for various menu commands.

### 23.15.3 Activation, Order Form, Registration, Updates

#### 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.
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](https://www.altova.com/). For more information about Altova LicenseServer and licensing via Altova LicenseServer, see the [Altova LicenseServer documentation](https://docs.altova.com/).
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

**Description**

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

**Description**

Checks with the Altova server whether a newer version than yours is currently available and displays a message accordingly.

### 23.15.4 Other Commands

#### Support Center

**Description**

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.

#### XMLSpy on the Internet

**Description**

A link to the Altova website on the Internet. You can learn more about XMLSpy and
related technologies and products at the [Altova website](https://www.altova.com).

**XMLSpy Training**

*Description*

A link to the Online Training page at the [Altova website](https://www.altova.com). Here you can select from online courses conducted by Altova’s expert trainers.

**About XMLSpy**

*Description*

Displays the splash window and version number of your product. If you are using the 64-bit version of XMLSpy, this is indicated with the suffix (x64) after the application name. There is no suffix for the 32-bit version.
23.16 Command Line

Certain XMLSpy actions can be carried out from the command line. These commands are listed below:

Open a file

Command: `xmlspy.exe file.xml`
Action: Opens the file, `file.xml`, in XMLSpy

Note: If an XML file has an SPS file already assigned to it, then the XML file is opened in Authentic View. Otherwise, the XML file is opened in Text View. If an SPS file is not assigned, one can be assigned with the `/sps` flag (see below).

Open multiple files

Command: `xmlspy.exe file1.xml file2.xml`
Action: Opens the files, `file1.xml` and `file2.xml`, in XMLSpy

Assign an SPS file to an XML file for Authentic View editing

Command: `xmlspy.exe myxml.xml /sps mysps.sps`
Action: Opens the file, `myxml.xml` in Authentic View with `mysps.sps` as its SPS file. The `/sps` flag specifies that the SPS file that follows is to be used with the XML file that precedes the `/sps` flag (for Authentic View editing).

Open a new XML template file via an SPS file

Command: `xmlspy.exe mysps.sps`
Action: Opens a new XML file in Authentic View. The display will be based on the SPS and the new XML file will have a skeletal structure based on the SPS schema. The name of the newly created XML file must be assigned when saving the XML file.

Open an SPS file as an XML document in Text View

Command: `xmlspy.exe /raw mysps.sps`
Action: Opens the file `mysps.sps` as an XML document in Text View. The `/raw` flag specifies that the SPS file that follows is to be edited as an XML file.
XMLSpy is an Automation Server: It exposes programmable objects to other applications called Automation Clients. An Automation Client can directly access the objects and functionality that the Automation Server makes available. So, an Automation Client of XMLSpy can use, for example, the XML validation functionality of XMLSpy. As a consequence, developers can enhance their applications with the ready-made functionality of XMLSpy.

The programmable objects of XMLSpy are made available to Automation Clients via the Application API of XMLSpy, which is a COM API. The Application API of XMLSpy will also be called Application API for short from now onwards. The object model of the Application API and a complete description of all the available objects are provided in this documentation (see the section Application API).

Execution environments

The Application API can be accessed from within the following environments:

- **Scripting Editor**
- **IDE Plug-ins**
- **External programs**
- **ActiveX Integration**

Each of these environments is described briefly below.

**Scripting Editor: Customizing and modifying XMLSpy functionality**

You can customize your installation of XMLSpy by modifying and adding functionality to it. You can also create Forms for user input and modify the user interface so that it contains new menu commands and toolbar shortcuts. All these features are achieved by writing scripts that interact with objects of the Application API. To aid you in carrying out these tasks efficiently, XMLSpy offers you an in-built Scripting Editor. A complete description of the functionality available in the Scripting Editor and how it is to be used is given in the Scripting Editor section of this documentation. The supported programming languages are **JScript** and **VBScript**.

**IDE Plug-ins: Creating plug-ins for XMLSpy**

XMLSpy enables you to create your own plug-ins and integrate them into XMLSpy. You can do this using XMLSpy’s special interface for plug-ins. A description of how to create plug-ins is given in the section **XMLSpy IDE Plug-ins**.

An application object gets passed to most methods that must be implemented by an IDE plug-in and gets called by the application. Typical languages used to implement an IDE plug-in are **C#** and **C++**. For more information, see the section **XMLSpy IDE Plugins**.

**External programs**

Additionally, you can manipulate XMLSpy with external scripts. For example, you could write a
script to open XMLSpy at a given time, then open an XML file in XMLSpy, validate the file, and print it out. External scripts would again make use of the Application API to carry out these tasks. For a description of the Application API, see the section Application API.

Using the Application API from outside XMLSpy requires an instance of XMLSpy to be started first. How this is done depends on the programming language used. See the section Programming Languages, for information about individual languages.

Essentially, XMLSpy will be started via its COM registration. Then the Application object associated with the XMLSpy instance is returned. Depending on the COM settings, an object associated with an already running XMLSpy can be returned. Any programming language that supports creation and invocation of COM objects can be used. The most common of these are listed below.

- **JScript** and **VBScript** script files have a simple syntax and are designed to access COM objects. They can be run directly from a DOS command line or with a double click on Windows Explorer. They are best used for simple automation tasks.
- **C#** is a a full-fledged programming language that has a wide range of existing functionality. Access to COM objects can be automatically wrapped using C#.
- **C++** provides direct control over COM access but requires relatively larger amounts of code than the other languages.
- **Java**: Altova products come with native Java classes that wrap the Application API and provide a full Java look-and-feel.
- Other programming languages that make useful alternatives are: Visual Basic for Applications, Perl, and Python.

**ActiveX Integration**

A special case of accessing the Application API is via the XMLSpy ActiveX control. This feature is only available if the XMLSpy integration package is installed. Every ActiveX Control has a property that returns a corresponding COM object for its underlying functionality. The manager control provides an Application object, the document control a Document object, and the placeholder object, in cases where it contains the project tree, returns the Project object. The methods supported by these objects are exactly as described in the Interfaces section of the Application API. Care must be taken not to use methods that do not make sense in the context of ActiveX control integration. For details see ActiveX Integration.

**About Programmers' Reference**

The documentation contained in the Programmers' Reference for XMLSpy consists of the following sections:

- **Scripting Editor**: a user reference for the Scripting Environment available in XMLSpy
- **IDE Plug-ins**: a description of how to create plug-ins for XMLSpy
- **Application API**: a reference for the Application API
- **ActiveX Integration**: a guide and reference for how to integrate the XMLSpy GUI and XMLSpy functionality using an ActiveX control
1 Scripting Editor

The Scripting Editor of XMLSpy uses the Form Editor components of the Microsoft .NET Framework, and thus provides access to the Microsoft .NET Framework. This means that JScripts and VBScripts not only work with the XMLSpy API—which is a COM API and the API of XMLSpy—but can also access and use classes of the Microsoft .NET framework.

You can therefore create and use your own macros and forms within XMLSpy, and thus add to and modify the functionality of your installation of XMLSpy.

**Note:** Visual Basic is **not supported** as a language in the scripting environment. Only VBScript and JavaScript are. Ensure that you use VBScript syntax and not Visual Basic syntax in the scripting environment.

**Note:** Microsoft's **.NET Framework 2.0 or higher** is a system prerequisite for Scripting Editor, and it must be installed before XMLSpy is installed.

The Scripting Editor

The Scripting Editor (**screenshot below**) opens in a separate window and is accessed via the **Tools | Scripting Editor** menu command in the XMLSpy GUI. The programming languages that can be used in the Scripting Environment are **JScript** and **VBScript**. The scripting language can be changed by right-clicking the Project item in the Project window, selecting **Scripting Language**, and selecting the language you want.
**What you can do with the Scripting Editor**

In the Scripting Editor, you can create Forms, Event Handlers, and Macros to build up a Scripting Project. A Scripting Project can then be set as the Global Scripting Project for XMLSpy, thus enabling scripts in the Scripting Project to be used in the application. Additionally, different Scripting Projects can be assigned to different XMLSpy projects, thus allowing different scripts to be used for different XMLSpy projects.

Every script project can define the .NET runtime version it wants to use. An application can handle multiple scripting projects with different .NET runtime versions simultaneously, but the appropriate .NET version must be installed. For example, script projects with .NET 4.0 will only run on computers having .NET 4.0 installed.

**Documentation about the Scripting Editor**

The documentation describing the Scripting Environment (this section) is organized into the following parts:

- **An overview**, which provides a high level description of the Scripting Editor and Scripting Projects.
- **A list of steps required to create a Scripting Project**.
- **An explanation of Global Declarations**, together with an example.
- **A description of how to create Forms**.
- **A discussion of XMLSpy-specific event handlers**.
- **An explanation of how to use macros** in the Scripting Editor and in XMLSpy.
1.1 Overview

The Scripting Editor provides an interface in which you can: (i) graphically design Forms while assigning scripts for components in the Form; (ii) create Event Handlers, and (iii) create Macros.

These Forms, Event Handlers, and Macros are organized into scripting projects, which are then assigned to XMLSpy application projects and can be used in the application.

Variables and functions can be defined in a Global Declarations script, which is always executed before Macro or Event Handler scripts.

This section gives an overview of the Scripting Editor and Scripting Projects. It is organized into the following sections:

- **Scripting Projects in XMLSpy**, which describes how the scripting projects you create with the Scripting Editor will be used in XMLSpy.
- **The Scripting Editor GUI**, which provides a detailed look at the different parts of the Scripting Editor GUI and how they are to be used.
- **Components of a Scripting Project**, which explains the different components that go to make up a scripting project.

The details about the creation of the various components (Global Declarations, Forms, Event Handlers, and Macros) are described in their respective sections.

.NET assemblies

Every scripting project can have references to .NET assemblies—in addition to the default references. .NET assemblies can be added for the whole scripting project or for individual macros (by using the new CLR.LoadAssembly command in the source code; see Built-in Commands). Assemblies can be added, for example, from the Global Assembly Cache.

To add an assembly, right-click the project or macro, and, from the context menu that pops up, select **Add .NET Assembly | Assembly from Global Cache (GAC)**.

This works in the same way as with Visual Studio and allows access not only to the complete Microsoft .NET Framework but also to any user-defined assembly.

1.1.1 Scripting Projects in XMLSpy

All scripts and scripting information created in the Scripting Editor are stored in **Altova Scripting Projects (.asprj files)**.

You can create any number of Altova Scripting Projects. After a scripting project has been created, it can be used in the following ways:

- It can be set as the global scripting project for XMLSpy. Scripts in the global scripting project can then be called from within the application, and macros of the Global Scripting Project can be used for all XMLSpy projects.
- It can be assigned to an XMLSpy project (as an application project). When an XMLSpy project is open in XMLSpy, scripts in the associated scripting project can be called.

Your XMLSpy package contains a sample scripting project called **SampleScripts.asprj**. This file contains global declarations for a few standard tasks and is located in the folder: C:\Users
Setting the global scripting project of an application
The global scripting project of an application is set in the Scripting tab of the Options dialog of XMLSpy (screenshot below, Tools | Options).

To set a global scripting project for XMLSpy, check the Activate Scripting check box and then browse for the Altova Scripting Project (.asprj) file you want. You can also specify: (i) whether Auto-Macros in the scripting project should be automatically executed when XMLSpy starts, and (ii) whether application event handler scripts in the project should be automatically executed or not; check or uncheck the respective check boxes accordingly.

Note: Nested script execution is possible, i.e. Macros can call other macros, and events are received during macro, or event, execution.

Assigning a scripting project to an XMLSpy project
A scripting project is assigned to an XMLSpy project as follows:

1. In the XMLSpy GUI, open the required application project.
2. Select the menu command Project | Script Settings.
3. Check the Activate Project Scripts check box and select the required scripting project (.asprj file). If you wish to run Auto-Macros when the XMLSpy project is loaded, check the Run Auto-Macros check box.
4. Click OK to finish.

Note: To deactivate (that is, unassign) the scripting project of an XMLSpy project, uncheck the Activate Project Scripts check box.

1.1.2 The Scripting Editor GUI
The Scripting Editor GUI is shown below. It has the following parts:

- A toolbar
- A Scripting Project Tree pane (top left-hand side)
- A Properties and Events pane (bottom left)
- A Main Window with Design and Source tabs
Scripting Editor toolbar
The Scripting Editor toolbar contains icons for:

- Standard file commands such as **New**, **Open**, **Save**, and **Print**. These commands are used to create new scripting projects, open existing scripting projects, and save and print scripting projects.
- Standard editing commands such as **Copy**, **Paste**, **Undo**, **Redo**, **Find**, and **Replace**. Note that the **Find** and **Replace** commands are applied to code in the Source tab of the Scripting Editor.

Scripting Project Tree
The Scripting Project Tree (screenshot below) shows the various components of the scripting project, structured along four main branches: (i) Global Declarations, (ii) Forms, (iii) Events, and (iv) Macros.

The Scripting Project Tree provides access to each component of the scripting project. For example, in order to display and edit a particular Form, expand the Forms folder in the tree (see
right-click the Form you wish to display or edit, and click **Open** from the context menu that pops up.

A quicker way to open a Form, Event, macro, or the Global Declarations script, is to double-click the respective icon, or text. To delete a Form or Macro from the scripting project, right-click the component and select the **Delete** command from the context menu.

The Scripting Project Tree pane contains a toolbar with icons (*screenshot below*).

The icons, from left to right, are for: (i) **creating a new macro**, (ii) **creating a new form**, (iii) **running a macro**, and (iv) **debugging a macro**. These commands are also available in the context menu that appears when you right-click any component in the Scripting Project Tree.

### Properties and Events

The Properties and Events pane (*screenshot below*) displays the following:

- Form properties, when the Form is selected
- Object properties, when an object in a Form is selected. (The screenshot below shows, at left, the properties of the object selected in the Form at right.)
- Form events, when a Form is selected
- Object events, when an object in a Form is selected

To switch between the properties and events of the selected component, click, respectively, the **Properties** icon (third from left in the Properties and Events toolbar, see *screenshot above*) and the **Events** icon (fourth from left).

The first and second icons from left in the toolbar are, respectively, the **Categorized** and **Alphabetical** icons. These display the properties or events either organized by category or organized in ascending alphabetical order.

When a property or event is selected, a short description of it is displayed at the bottom of the Properties and Events pane.
Main Window
The Main Window displays one component at a time and has one or two tabs depending on what is being displayed. If a Global Declarations script, an Event, or a Macro is being displayed, then a single tab, the Source tab, displays the source code of the selected component.

The Source tab supports:
- syntax coloring
- source code folding
- setting/deleting bookmarks using CTRL+F2
- autocompletion entry helper with parameter info
- Goto Brace, Goto Brace Extend
- Zoom In / Zoom Out
- full method/property signature shown next to the autocompletion entry helper
- brace highlighting during code entry
  if ( x == y.GetName(a, b, c) )
- mouse over popups; placing the mouse over a known method or property, displays its signature (and documentation if available)

If a Form is being displayed, then the Main Window has two tabs: a Design tab showing and enabling the layout of the Form, and a Source tab containing the source code for the Form. Content in both the Design tab and Source tab can be edited.

Note: Since JScript and VB Script are untyped languages, entry helpers and auto-completion is supported only in cases of "fully qualified constructs" and "predefined" names.

If names start with objDocument, objProject, objXMLData, or objAuthenticRange, members of the corresponding interface will be shown. Auto-completion entry helper and parameter info are shown during editing, but can also be obtained on demand by pressing Ctrl+Space.

Form Object Palette
The Form Object Palette contains all the objects that are available for designing Forms and looks something like the screenshot below. Registered ActiveX controls can be added to the Form Object Palette by right-clicking the pane and selecting the Add ActiveX Control command.
To insert an object from the Form Object Palette click the object you want in the palette, then click at the location in the Form where you wish to insert the object. The object will be placed at this location. In many cases you will need to supply some properties of the object via the Properties and Events pane. You can drag the object to other locations as well as resize it. Further, a number of editing commands, such as centering and stacking objects, can be accessed via the context menu of the selected Form object.

Some Form objects, such as Timer, are not added to the Form but are created as Tray Components in a tray at the bottom of the Main Window. You can select the object in the tray and set properties and event handlers for the object via the Properties and Events pane. For an example of how Tray Components are handled, see Form usage and commands.

### 1.1.3 Components of a Scripting Project

An Altova Scripting Project consists of the following four major components:

- **Global Declarations**, a component which contains definitions of variables and functions that are available to, and can be used by, all Forms, Macros, and Event Handler scripts in the scripting project.

- **Forms**, a component which contains all the Forms defined in the scripting project.

- **Events**, a component which contains Event Handler scripts for all application-based—as opposed to Form-based—events.

- **Macros**, a component which contains all the Macros defined in the scripting project.

These components are displayed in and accessed via the Scripting Project Tree of the Scripting Editor (screenshot below).
Given below is a brief description of each of these components.

**Global Declarations**
The Global Declarations component is a script that contains variables and functions that can be used by Forms, Event Handlers, and Macros. The functions make use of the XMLSpy API to access XMLSpy functionality. Creating a variable or function in the Global Declarations module enables it to be accessed from all the Forms, Event Handlers and Macros in the scripting project.

To add a variable or function, open the Global Declarations component (by right-clicking it in the Scripting Project Tree and selecting **Open**) and edit the Global Declarations script in the Main Window. In this script, add the required variable or function.

**Forms**
In the Scripting Editor, you can build a Form graphically using a palette of Form objects such as text input fields and buttons. For example, you can create a Form to accept the input of an element name and to then remove all occurrences of that element from the active XML document.

For such a Form, a function script can be associated with a text box so as to take an input variable, and an Event Handler can be associated with a button to start execution of the delete functionality, which is available in the XMLSpy API. A Form is invoked by a call to it either within a function (in the Global Declarations script) or directly in a Macro. For details of how to create and edit Forms, see the **Forms** section.

**Event handling**
Event Handler scripts can be associated with a variety of available events. You can control events that occur both within Forms (**Form events**) and within the general application interface (**application events**). The script associated with an event is executed immediately upon the triggering of that event.

Most events have parameters which provide detailed information about the event. The return value from the script typically instructs the application about how to continue its processing (for example, the application may not allow editing).

An Event Handler runs when the relevant event occurs in the Form or in XMLSpy. For details about how to create event handlers, see **Event Handlers**.

**Macros**
Macros are used to implement complex or repetitive tasks. Macros do not use either parameters or return values.

In a Macro, it is possible to access all variables and functions declared in the Global Declarations and to display Forms for user input.
For a simple example of creating a Macro, see Writing a Macro. Also see Running Macros for a description of the ways in which a Macro can be called. A Macro is run from within the XMLSpy interface by clicking Tools | Macros | [MacroName]
1.2 Creating a Scripting Project

The broad steps for creating a Scripting Project are as follows:

1. Open the Scripting Editor by clicking the command **Tools | Scripting Editor**.

2. In the Scripting Editor, open a new scripting project by clicking the **New** icon in the Scripting Editor toolbar. The Project Settings dialog (*screenshot below*) pops up. You can also access this dialog by right-clicking a project in the Scripting Project Tree pane (in the top left part of the Scripting Editor window) and clicking the command **Project Settings**.

   ![Project Settings Dialog](image)

   Select either JScript or VBScript in the first combo box and the .NET Framework in the second combo box. To enable higher .NET Frameworks (such as .NET Framework 4.5 on Windows 8), check the **Automatically use higher .NET Framework** check box. Then click **OK**. The new Scripting Project is created.

3. Click the **Save** icon in the Scripting Editor toolbar to save the Scripting Project as a .asprj file.

4. A Scripting Project can be considered to be made up of several components that work together. These components will typically be a combination of: Global Declarations, Forms, Events, and Macros. They can be created in any order, but you should clearly understand how they work together. The way each type of component is called and executed is **described below**. How to create each type of component is described in the respective sections about the component type.

5. After you have finished creating all the required components, save the Scripting Project (by clicking the **Save** icon in the Scripting Editor toolbar).

6. Close the Scripting Editor.

**Note:** Right-clicking the Project folder and selecting **Project Settings** lets you change the scripting language at any time.

**How Forms, Event Handlers, and Macros are called and executed**

Forms, Event Handlers, and Macros are all created in the Scripting Editor. However, the way they are called and executed is different for each and has a bearing on how you create your scripting
projects.

- A Form is invoked by a call to it either within a function in the Global Declarations script or directly in a Macro.

- An Event Handler runs when the relevant event occurs in XMLSpy. If an Event Handler for a single event is defined in both the Global Scripting Project and the XMLSpy-project-specific Scripting Project, then the event handler for the project-specific Scripting Project is executed first and that for the Global Scripting Project immediately afterwards.

- A Macro is executed from within the XMLSpy interface by clicking Tools | Macros | [MacroName]. In a Macro, it is possible to access all variables and functions declared in the Global Declarations and to display Forms for user input.
1.3 Global Declarations

The Global Declarations component is present by default in every Scripting Project (see screenshot below), and therefore does not have to be created. In order to add variables and functions to the Global Declarations script of a Scripting Project, you need to open the Global Declarations script and add the code fragment to the Global Declarations script. See Components of a Scripting Project and Creating a Scripting Project for more information.

To open the Global Declarations script of a Scripting Project, right-click the Global Declarations item in the Scripting Project Tree (screenshot above), and select Open. The Global Declarations script opens in the Main Window.

Note: Every time a macro is executed or an event handler is called, global declarations are re-initialzied.

Given below is an example function. Remember that creating a variable or function in the Global Declarations script makes this variable or function accessible to all Forms, Event Handlers, and Macros.

**Example function**
A function called RemoveAllNamespaces would have code like this:

```js
function RemoveAllNamespaces(objXMLData)
{
    if(objXMLData == null)
        return;

    if(objXMLData.HasChildren) {
        var objChild;

        // spyXMLDataElement := 4
        objChild = objXMLData.GetFirstChild(4);

        while(objChild) {
            RemoveAllNamespaces(objChild);
            try {
                var nPos,txtName;
                txtName = objChild.Name;

                if((nPos = txtName.indexOf(":")) >= 0) {
                    objChild.Name = txtName.substring(nPos+1);
                }
            }
        }
    }
}
```
objChild = objXMLData.GetNextChild();
}
catch(Err) {
    objChild = null;
}
}
}

Note:

- It is possible to define local variables and helper functions within macros and event handlers. Example:

```javascript
//return value: true allows editing
//return value: false disallows editing
var txtLocal;
function Helper()
{
    txtMessage = txtLocal;
    Application.ShowForm("MsgBox");
}
function On_BeforeStartEditing(objXMLData)
{
    txtLocal = "On_BeforeStartEditing()";
    Helper();
}
```

- Recursive functions are supported.
1.4 **Forms**

Creating and editing Forms in the Scripting Editor consists of the following steps:

1. **Creating a New Form.** The new Form is created and named, and has properties defined for it.
2. **Designing the Form.** A Form is designed by adding Form Objects to it and assigning values for the different Form Objects.
3. **Scripting Form Events.** Scripts are assigned to Form-related events.

### 1.4.1 Creating a New Form

Creating a new Form in the Scripting Editor involves the following steps:

1. **Creating a new Form and naming it**
2. **Specify the properties of the Form**

**Creating a new Form and naming it**

To add a new Form to a scripting project, click the **Add Form** icon (highlighted in screenshot below) in the toolbar of the Project Overview pane. Enter the name of the new Form.

A new Form is added to the project. It appears in the Main Window and an entry for it is created in the Scripting Project Tree pane, under the Forms heading. Press the F2 function key to rename the form, or right click the form name and select Rename from the context menu. In the screenshot below, we have named the new Form *Registration*.

**Form properties**

The properties of the Form, such as its size, background color, and font properties, can be set in the Properties pane. The screenshot below shows the size and background-color property values in bold, in the *Layout* and *Appearance* categories, respectively.
Testing a Form
You can test a form in the Scripting Editor by right-clicking it in the Project Overview pane and selecting the Test Form Command.

1.4.2 Form Design and Form Objects
Designing a Form consists of the following steps:
- Placing an object from the Form Object Palette in the Form design.
- Assigning values for the properties of individual Form Objects.
- Assigning scripts for Form-based events.

The Form Object Palette
The Form Object Palette contains all the objects that are available for designing Forms and looks something like the screenshot below. Registered ActiveX controls can be added to the Form Object Palette by right-clicking the pane and selecting the Add ActiveX Control command.
To insert an object from the Form Object Palette click the object you want in the palette, then click at the location in the Form where you wish to insert the object. The object will be placed at this location. In many cases you will need to supply some properties of the object via the Properties and Events pane. You can drag the object to other locations as well as resize it. Further, a number of editing commands, such as centering and stacking objects, can be accessed via the context menu of the selected Form object.

Some Form objects, such as Timer, are not added to the Form but are created as Tray Components in a tray at the bottom of the Main Window. You can select the object in the tray and set properties and event handlers for the object via the Properties and Events pane. For an example of how Tray Components are handled, see Form usage and commands.

Some of the most commonly used objects are described below:

- **Label**: Adds text fields such as captions or field descriptions.
- **Button**: Adds a button. It is possible to assign bitmaps as background images for these buttons.
- **CheckBox**: Adds a check box, which enables Yes/No type selections.
- **ComboBox**: Adds a combo box, which allows the user to select an option from a drop-down menu.
- **ListBox**: Adds a list box, which displays a list of items for selection.
- **TextBox**: Enables the user to enter a single line of text.
- **RichTextBox**: Enables the user to enter multiple lines of text.
Creating objects and setting their properties

To create an object in the Form, first select the required object in the Form Object Palette and then click the location in the Form where you want to insert it. After the object has been inserted, you can resize it as well as drag it to another location in the Form.

When an object is selected in the design, you can specify its properties in the Properties and Events pane. In the toolbar of the Properties and Events pane, click the Properties icon to display a list of the object's properties.

For example, in the screenshot below, the Label object with the text Start Date has been selected in the design. In the Properties and Events pane, the name of the object (which is the name that is to be used to identify the object in code, Label1 in the screenshot below) is given in the Design category of properties; in this case, the name of the object is Label1.

The text of the label (which is what appears in the Form) must be entered as the value of the Text property in the Appearance category of properties.

To assign other object properties, enter values for them in the Properties and Events pane.

Testing a Form

You can test a form in the Scripting Editor by right-clicking it in the Project Overview pane and selecting the Test Form Command.

1.4.3 Form Events

When an object is selected in the design, clicking on the Events icon in the toolbar of the Properties and Events pane (fourth icon from left), displays all the events available for that object (see screenshot below). These can be displayed either by category (screenshot below) or alphabetically.
For each event, you can enter the name of an existing event handler or function. Alternatively:

- you can double click on an event to create: (i) an empty function script in the Source tab of the Main Window, and (ii) an association of the newly created function with the selected event.
- double click a button in the design tab, to directly generate the handler stub in the code window.

The screenshot below was taken after the Click event was double-clicked. Notice that an empty event handler function called `FormExample_Label1_Click` has been created in the Main Window and that, in the Properties and Events pane, this function has been associated with the Click event.

```
function FormExample_Label1_Click( objSender, eEventArgs )
{
    // Occurs when the component is clicked.
}
```

Enter the required scripting code and save the project.

**Writing the required scripts**

After the visual design of the form is complete, form objects will typically be associated with suitable scripts. The example below is a script that adds colors when a button is clicked. The script is inserted as an event handler for the Click event of the button `Button1` (the event is available in the Properties and Events pane when the button is selected in the design):

```
function FormExample_Button1_Click( objSender, eEventArgs )
{
    // Sets the ForeColor (red) of the button.
}
```
// Sets the BackColor (blue) of the button.
objSender.BackColor = CLR.Static("System.Drawing.Color").Blue;
// Sets the form BackColor (green).
objSender.FindForm().BackColor = CLR.Static("System.Drawing.Color").Green;
}
1.5 Events

The Events folder of the scripting project (see screenshot below) contains folders for the following type of events:

- Application Events
- Document Events
- Authentic View Events
- Grid View Events
- Text View Events

Note that these events are XMLSpy-specific, as opposed to Form-based events. Each of the folders listed above contains a set of events for which Event Handler scripts can be written.

Application Events, for example, are shown in the screenshot below.

To access the event handler script of any of these events, right-click the event and select Open from the context menu. The script will be displayed in the Main Window (see screenshot below) and can be edited there. After you have finished editing the script, save changes by clicking the Save command in the toolbar of the Scripting Editor.

Note the following points:

- Event Handlers need function headers with the correct spelling of the event name. Otherwise the Event Handler will not be called.
- It is possible to define local variables and helper functions within Macros and Event Handlers. Example:

```javascript
function Helper() {
    var txtLocal;
    txtMessage = txtLocal;
}
```
Application Events

OnInitialize
The `OnInitialize` event is raised after the main window becomes visible but before any project is loaded. This event is not raised if the application can't be loaded at all.

OnRunning
If the application is completely loaded and after the `OnInitialize` event occurs, the `OnRunning` event is raised.

OnShutdown
The event is raised after any open project and all documents have been closed on shutdown of the application. The main window is no longer visible.

Example
The following script is an Event Handler for the `On_BeforeOpenProject` event. It allows you to add a script that will be executed each time before XMLSpy opens a project. The example script below sequentially opens all XML files located in the XML folder of the project and validates them. If the validation fails, the script shows the validation error and stops. If a file passes the validity test, it will be closed and the next file will be opened.

Enter the following script for the `On_BeforeOpenProject()` event, and then save the scripting project.

```javascript
function On_BeforeOpenProject() {
    var bOK;
    var nIndex, nCount;
    var objItems, objXMLFolder = null;

    objItems = Application.CurrentProject.RootItems;
    nCount = objItems.Count;

    // search for XML folder
    for (nIndex = 1; nIndex <= nCount; nIndex++) {
        var txtExtensions;
        txtExtensions = objItems.Item(nIndex).FileExtensions;
        if (txtExtensions.indexOf("xml") >= 0) {
            objXMLFolder = objItems.Item(nIndex);
            break;
        }
    }

    Application.ShowForm("MsgBox");
}
function On_BeforeStartEditing(objXMLData) {
    txtLocal = "On_BeforeStartEditing()";
    Helper();
}
```

- In order for events to be processed, the Process Events options must be toggled on in the Scriptings options of XMLSpy. See [Scripting Projects in XMLSpy](https://www.altova.com/xmlspy/scripting/programming-points) for details.
- Also see [Programming Points](https://www.altova.com/xmlspy/scripting/programming-points).
// does XML folder exist?
if (objXMLFolder) {
    var objChild, objDoc;
    nCount = objXMLFolder.ChildItems.Count;

    // step through associated xml files
    for (nIndex = 1; nIndex <= nCount; nIndex++) {
        objChild = objXMLFolder.ChildItems.Item(nIndex);
        try {
            objDoc = objChild.Open();

            // use JScript method to access out-parameters
            var strError = new Array(1);
            var nErrorPos = new Array(1);
            var objBadData = new Array(1);

            bOK = objDoc.IsValid(strError, nErrorPos, objBadData);

            if (!bOK) {
                // if the validation fails, we should display the
                // message from XMLSpy
                // of course we have to create the form "MsgBox" and
                // define the global txtMessage variable
                //
                // txtMessage = Position:" + nErrorPos[0] + ":" + // strError[0];
                // txtMessage +="\n\nXML:
" + objBadData[0].Name + ", " +
                // objBadData[0].TextValue;

                // Application.ShowForm("MsgBox");

                break;
            }
        }
        objDoc.Close(true);
        objDoc = null;
    }
    catch (Err) {
        // displaying the error description here is a good idea

        // txtMessage = Err.Description;
        // Application.ShowForm("MsgBox");

        break;
    }
}
}
Testing the Event Handler
Switch to XMLSpy, and open a project to see how the BeforeOpenProject event is handled.
1.6 Macros

Macros automate repetitive or complex tasks. In the Scripting Environment, you can create a script that calls application functions as well as custom functions that you have defined. This flexibility provides you with a powerful method of automating tasks within XMLSpy. This section about macros is organized as follows:

- Creating and Editing a Macro describes how to create a new macro and edit an existing one.
- Running a Macro explains how a macro can be run from the Scripting Editor and from the broader XMLSpy environment as well.
- Debugging describes how macros can be debugged.

Key points about macros
Given below is a summary of important points about macros.

- Any number of macros can be added to the active scripting project. These macros are saved in the Altova Scripting Project file (.asprj file).
- Functions that are used in a macro can be saved as a Global Declaration. All Global Declarations are also saved in the Altova scripting project file (.asprj file).
- The macro can be tested by running it from within the Scripting Editor, and it can be debugged from within the Scripting Editor.
- XMLSpy can have one global Scripting Project, and a second scripting project, assigned to the currently loaded project, active at any one time; the macros are available to both of them. See Running a Macro for details.

1.6.1 Creating and Editing a Macro

The following operations enable you to create a new macro and edit an existing macro.

Creating a new macro
Right-click the Macro folder in the Scripting Projects tree and select Add Macro from the context menu. (The Add Macro command can also be selected from the context menu of any item in the Scripting Projects tree.) Alternatively, click the New Macro icon in the toolbar of the Scripting Projects tree.

The newly created (and empty) macro document is displayed in the Main Window, and the name of the macro is displayed in the title bar of the Scripting Editor (screenshot below).

Naming or renaming a macro
To name or rename a macro, click the macro name in the Scripting Project tree and press the F2 function key, or right click the name and select Rename from the context menu.

**Opening a macro**
To open a macro, right-click the macro in the Macros folder of the Scripting Project tree (see screenshot above), and select the Open command. The macro is displayed in the Main Window and its name is displayed in the title bar of the Scripting Editor (screenshot below). Alternatively, double-clicking a macro in the Scripting Project tree opens it in the Main Window.

**Editing the macro**
To edit a macro, enter or edit its code in the Main Window. For example, the following code creates the Form named Form1 in memory and then shows it. Form1 must already have been created (using the Scripting Editor’s Form creation) before this macro is run.

```vbscript
objForm = CreateForm( 'Form1' );
objForm.ShowDialog();
```

The following macro uses the RemoveAllNamespaces function to remove all namespaces in the active XML document.

```vbscript
if(Application.ActiveDocument != null) {
    RemoveAllNamespaces(Application.ActiveDocument.RootElement);
    Application.ActiveDocument.UpdateViews();
}
```

The RemoveAllNamespaces function itself will have to be defined in the Global Declarations script. After the RemoveAllNamespaces function has been defined, the macro is complete and can be run.

**Note:** Macros do not support parameters or return values.

**Setting a macro as an Auto-Macro**
When a macro is set as an Auto-Macro it can be run automatically when: (i) XMLSpy is started, or (ii) an Altova XMLSpy project is loaded in XMLSpy. To specify whether Auto-Macros should be run in each of these two events, check the Run Auto-Macros option in the Automatic Script Processing pane of the relevant dialogs:

- **When XMLSpy is started:** the Scripting tab of the XMLSpy Options dialog (Tools | Options menu command).
- **When an XMLSpy project is loaded into XMLSpy:** the Scripting dialog (screenshot below, Project | Scripting Settings menu command).
To set a macro as an Auto-Macro, right-click the macro in the Scripting Project tree and select the command **Set as Auto-Macro**. This is a toggle command; so to remove the Auto-Macro setting of a macro, select the command again.

### 1.6.2 Running a Macro

To run a macro in the Scripting Editor, right-click the macro in the Scripting Project tree and select the command **Run Macro**.

There are different ways to run a macro from XMLSpy:

- **Via the Tools | Macros menu** of XMLSpy.
- **By creating and using a toolbar button** for a macro.
- **By creating and using a menu item** for a macro.

Note that only one macro can be run at a time. After a macro (or event) is executed, the script is closed and global variables lose their values.

### The XMLSpy command to run Macros

The **Tools | Macros** menu command (**screenshot below**) opens a submenu containing the macros defined in the Scripting Project that is currently active in XMLSpy. The active Scripting Projects are specified in the Scripting tab of the Options dialog, or in the Scripting tab of the project settings.

From the submenu of available macros, select the macro to run. The macro will be executed.
**Toolbar icon**
You can create an icon in the toolbar or a menu item that runs a selected macro. To do this, click **Tools | Customize | Macros**. This causes the Customize dialog to be displayed (screenshot below).

![Customize dialog](image)

Now do the following:

1. In the Macros tab of the Customize dialog, select the required macro from the Macros pane. The macros in the Macros pane are those in the active Scripting Project (which is specified in the Scripting tab of the Options dialog).
2. In the **Display Text** input field enter the name of the icon. This name will appear when the cursor is placed over the icon when it is in the toolbar.
3. Click **Add Command** to add it to the list of commands.
4. Select the command and click **Edit Icon** to create a new icon.
5. Drag the finished icon from the **Associated Commands** pane and drop it on to the toolbar or menu when the cursor changes from an arrow to an I-beam or line.
6. Macros can even be assigned their own shortcuts in the Keyboard tab of the Customize dialog (see screenshot above).

To remove the toolbar icon, open the Macros tab of the Customize dialog and drag the icon out of the toolbar and into the **Associated Commands** pane. Select the command in the **Associated Commands** pane and click **Remove** to remove the command from the pane.

**Item in the Tools menu**
The XMLSpy API includes a function, `AddMacroMenuItem()`, to add macros as menu items to the **Tools** menu. This function can be used to add one or more macros to the **Tools | Macros** list of macros. Typically, you should do this as follows:

1. Add the macro menu item by calling the XMLSpy API function, `AddMacroMenuItem()`.
Application.AddMacroMenuItem("DeleteElements","Delete Elements Dialog");

- The function's first parameter (DeleteElements in the example listing above) is the name of the macro. If you run the macro and there is an open project having scripts associated with it, XMLSpy searches for the macro in the project scripts first. If there are no project scripts, or if XMLSpy cannot find the macro, then it looks for the macro in the global scripts.

- The second parameter (Delete Elements Dialog) is the display text for the menu item.

2. Reset the **Tools** menu by calling `ClearMacroMenu()`. This removes all previously added menu items.

The best way to call these two functions is with the Autorun macro of the global scripting project or the **On_OpenProject** event.

### 1.6.3 Debugging a Macro

You can debug a macro using an installed debugger. To do this, right-click the macro in the Scripting Project tree and select the command **Debug Macro**.

This pops up the Just-In-Time Debugging dialog (screenshot below), which lists the debuggers available on the machine. Select the debugger you wish to use and click **Yes**.

![Just-In-Time Debugging](image)

An exception 'Runtime Error' has occurred in Script.

**Possible Debuggers:**

New instance of Microsoft Script Editor

Set the currently selected debugger as the default.

Do you want to debug using the selected debugger?

- Yes
- No

The selected debugger starts.
1.7 Programming Points

The following programming points should be noted:

- All namespaces and types of the following .NET assemblies can be accessed in the Microsoft .NET Framework per default:
  
  System  
  System.Data  
  System.Design  
  System.Drawing  
  System.Windows.Forms  
  System.XML  

  Additional assemblies can be added to the scripting project via the project's context menu, or dynamically (at runtime) in the source code by using CLR.LoadAssembly.

- Out-parameters from methods of the XMLSpy API require special variables in JScript. Given below are some examples.

  // use JScript method to access out-parameters  
  var strError = new Array(1);  
  var nErrorPos = new Array(1);  
  var objBadData = new Array(1);  
  bOK = objDoc.IsValid(strError,nErrorPos,objBadData);END

- Out-parameters from methods of the.NET Framework require special variables in JScript. For example:

  var dictionary =  
  CLR.Create( "System.Collections.Generic.Dictionary< System.String,  
  System.String >" );  
  dictionary.Add("1", "A");  
  dictionary.Add("2", "B");  

  // use JScript method to access out-parameters  
  var strOut = new Array(1);  
  if ( dictionary.TryGetValue("1", strOut) ) // TryGetValue will set  
  the out parameter  
  alert( strOut[0] ); // use out parameter

- .NET Methods that require integer arguments should not be called directly with JScript Number Objects which are Floating Point Values.

  For example, instead of:

  var objCustomColor = CLR.Static( "System.Drawing.Color" ).FromArgb( 128, 128, 128 );

  use:

  var objCustomColor = CLR.Static( "System.Drawing.Color" ).FromArgb( Math.floor( 128 ), Math.floor( 128 ), Math.floor( 128 ) );
To iterate .NET collections the JScript Enumerator as well as the .NET iterator technologies can be used:

For example:

```javascript
// iterate using the JScript iterator
var itr = new Enumerator( coll );
for ( ; !itr.atEnd(); itr.moveNext() )
   alert( itr.item() );

// iterate using the .NET iterator
var itrNET = coll.GetEnumerator();
while( itrNET.MoveNext() )
   alert( itrNET.Current );
```

.NET templates can be instantiated as shown below:

```javascript
var coll = CLR.Create( "System.Collections.Generic.List<System.String>" );
```

or

```javascript
CLR.Import( "System" );
CLR.Import( "System.Collections.Generic" );
var dictionary = CLR.Create( "Dictionary< String, Dictionary< String, String > >" );
```

.NET Enum values are accessed as shown below:

```javascript
```

Enumeration literals, as defined in the Altova type libraries, can now be used instead of numerical values.

```javascript
objExportXMIFileDlg.XMIType = eXMI21ForUML23;
```

## 1.7.1 Built-in Commands

This section lists:

- **Built-in commands**
  - `alert`
  - `conform`
  - `doevents`
  - `CreateForm`
  - `lastform`
  - `prompt`
  - `ShowForm`
  - `watchdog`

- **.NET interoperability commands**
  - `CLR.Create`
  - `CLR.Import`
  - `CLR.LoadAssembly`
Built-in commands
The following built-in commands are available.

ShowForm(strFormName : String)
Instantiates a New Form object from the given form name and immediately shows it as Dialog.
Return Value: A Number that represents the generated DialogResult (System.Windows.Forms.DialogResult).

Example:
var dialogResult = ShowForm( "FormName" );

Shows Form "FormName" as Dialog:

The DialogResult can be evaluated e.g. by:

if ( dialogResult ==
alert("ok");
else
alert("cancel");

CreateForm(strFormName : String)
Instantiates a New Form object from the given Form name.
Return Value: The Form object (System.Windows.Forms.Form) of the given name, or null if no Form with such name exists.

Example:
var myForm = CreateForm( "FormName" );
if ( myForm != null )
{
    myForm.textboxFirstName.Text = "Daniel";
    myForm.textboxLastName.Text = "Smith";
    var dialogResult = myForm.ShowDialog();
}

Shows Form "FormName" as Dialog - TextBoxes are initialized:
The DialogResult can be evaluated e.g. by:

```javascript
if ( dialogResult ==
    alert( "ok" );
else
    alert( "cancel" );
```

**lastform**

This global field can be used to conveniently access the last form object that was created. **Return Value:** Returns a reference to the last form object (System.Windows.Forms.Form) that was successfully instantiated via CreateForm() or ShowForm().

Example:

```javascript
CreateForm( "FormName" );
if ( lastform != null )
{
    lastform.textboxFirstName.Text = "Daniel";
    lastform.textboxLastName.Text = "Smith";
    var dialogResult = lastform.ShowDialog();
}
```

Shows Form "FormName" as Dialog - TextBoxes are initialized (similar to the CreateForm example above):

**doevents()**

Processes all Windows messages currently in the message queue. **Return Value:** None

Example:

```javascript
for ( i=0; i < nLongLastingProcess; ++i )
{
```
// do long lasting process

doevents(); // process windows messages; give UI a chance to update
}

**watchdog(bEnable : boolean)**

Long running CPU-intensive scripts cause the watchdog to ask the user if the script should be terminated. The `watchdog()` method is used to disable or enable this behavior.

Per default the watchdog is enabled.

*Return Value*: None

Example:

```javascript
watchdog( false ); // disable watchdog - we know the next statement is CPU intensive but it will terminate for sure
doCPUIntensiveScript();
watchdog( true ); // re-enable watchdog
```

**Usage tip:**

Calling `watchdog(true)` can also be used to reset the watchdog. This can be useful before executing long running (CPU intensive) tasks to ensure they have the maximum allowed script processing quota.

**alert(strMessage : String)** or **MsgBox(strMessage : String)**

An alert box is used to show a given message. The user will have to click "OK" to proceed.

*Return Value*: None

Example:

```javascript
alert( "Hello World" );
```

**confirm(strMessage : String)**

Opens a dialog that shows the given confirm message.

A confirm box is often used to verify or accept something. The user will have to click either "OK" or "Cancel" to proceed.

*Return Value*: A Boolean that represents the users answer. If the user clicks "OK", the dialog returns true, if the user clicks "Cancel", the dialog returns false.

Example:

```javascript
if ( confirm( "Continue processing?" ) == false )
    return;
```
prompt(strMessage : String, strDefault : String)
Opens a dialog that shows the given prompt message and a TextBox control with a default answer.
A prompt box is often used to input a simple string value.
_Return Value_: A String that contains the TextBox value or null if the user selected "Cancel".

Example:
```javascript
var name = prompt( "Please enter your name", "Daniel Smith" );
if ( name != null )
    alert( "Hello " + name + "!" );
```

**.NET interoperability commands**
To allow further interoperability with the .NET Framework additional functions are provided under CLR.

CLR.Import(strNamespaceCLR : String)
This is the scripting equivalent to the C# using / VB.Net imports keyword. This allows to leave out the given namespaces in successive calls like CLR.Create() and CLR.Static().
_Return Value_: None

Example:
```javascript
Instead of always having to use full qualified names:

    if ( ShowForm( "FormName" ) ==
    {
        var sName = lastform.textboxFirstName.Text + " " +
                    lastform.textboxLastName.Text;
```
one can import namespaces and use the short form:

```csharp
CLR.Import( "System.Windows.Forms" );
if ( ShowForm( "FormName" ) == CLR.Static( "DialogResult" ).OK )
{
    var sName = lastform.textboxFirstName.Text + " " + lastform.textboxLastName.Text;
    CLR.Static( "MessageBox" ).Show( "Hello " + sName );
}
```

**Please note:**
Importing a namespace does not add or load the corresponding assembly to the scripting project! Assemblies can be added to the scripting project dynamically (at runtime) in the source code by using `CLR.LoadAssembly`.

`CLR.ShowImports()`
Opens a MessageBox dialog that shows the currently imported namespaces. The user will have to click "OK" to proceed.

*Return Value:* None

**Example:**
```csharp
CLR.ShowImports();
```

`CLR.LoadAssembly(strAssemblyNameCLR : String)`
Loads the .NET assembly with the given long assembly name or file path.

*Return Value:* A Boolean value. True if the assembly could be loaded, false otherwise.

**Example:**
```csharp
// set clipboard text (if possible)
// System.Windows.Clipboard is part of the PresentationCore assembly, so load this assembly first:
if ( CLR.LoadAssembly( "PresentationCore, Version=3.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35", true ) )
{
    var clipboard = CLR.Static( "System.Windows.Clipboard" );
    if ( clipboard != null )
        clipboard.SetText( "HelloClipboard" );
```
CLR.ShowLoadedAssemblies()
Opens a MessageBox dialog that shows the currently loaded assemblies. The user will have to click "OK" to proceed.
Return Value: None

Example:
CLR.ShowLoadedAssemblies();

CLR.Create(strTypeNameCLR : String, constructor arguments ...)
Creates a new .NET object instance for the given typename. If more than one argument is passed the successive arguments are interpreted as the arguments for the constructor of the .NET object.
Return Value: A reference to the created .NET object

Examples:
var objArray = CLR.Create("System.Collections.ArrayList");

var newItem = CLR.Create( "System.Windows.Forms.ListViewItem", "NewItemText" );

var coll = CLR.Create( "System.Collections.Generic.List<System.String>" );

CLR.Import( "System" );
CLR.Import( "System.Collections.Generic" );
var dictionary = CLR.Create( "Dictionary< String, Dictionary< String, String > >" );
CLR.Static(strTypeNameCLR : String)
Gives access to .NET types that have no instances and contain only static members.
Return Value: A reference to the static .NET object

Examples:
var enumValStretch =
CLR.Static("System.Windows.Forms.ImageLayout").Stretch

var clipboard = CLR.Static("System.Windows.Clipboard");
clipboard.SetText("HelloClipboard");

if (ShowForm("FormName") ==
CLR.Static("System.Windows.Forms.DialogResult").OK)
  alert("ok");
else
  alert("cancel");

Form usage and commands
Form usage is as follows:

With Form objects, the Form Component Tree can be accessed naturally via field access:

For example, suppose there is a Form designed as follows:

MyForm
  ButtonPanel
    OkButton
    CancelButton
  TextEditor
  AxMediaPlayer1

TrayComponents:
  MyTimer

The Form can then be instantiated from script as:

var objForm = CreateForm("MyForm");

To access one its components the field access can be used:


or

objForm.TextEditor.Text = "Hello World";

To access Tray Components use the following method on the Form object:

var objTrayComponent = <A form object>.GetTrayComponent(strComponentName : String);

In our example to get a reference to the Timer Component to enable it use the following:
var objTimer = objForm.GetTrayComponent("MyTimer");
objTimer.Enabled = true;

For ActiveX Controls the underlying COM object can be accessed via the OCX property:

var ocx = lastform.AxMediaPlayer1.OCX; // get underlying COM object
ocx.enableContextMenu = true;
ocx.URL = "mms://apasf.apa.at/fm4_live_worldwide";
1.8 **Migrating to Scripting Editor 2010 and Later**

The Scripting Editor in XMLSpy from version 2010 onwards uses a different underlying technology than earlier versions used. Consequently, scripting projects that were created with versions of XMLSpy prior to version 2010 might need to be modified. The following points need to be noted.

- If a previous Scripting Projects (.prj file) is opened with the new Scripting Editor (version 2010 and later), the visual layout of Forms will be migrated as faithfully as possible and scripts will be copied as they are in the .prj file. You will then need to modify the scripts to be in accordance with the new technology used by the Scripting Editor, and which is described in this documentation.

- **TheView object**: The old Scripting Environment provided an artificial property named `TheView` that was only accessible from inside event handlers. It was used to access the Form that trigged the event (either directly or from one of its child controls). The new Scripting IDE does **not** provide this artificial property but instead provides the same functionality, and much more, with orthogonal built-in scripting helper functions combined with the power of the .NET framework.

  Since all event handlers in the new Scripting Environment get a sender object as a first parameter, the source that triggered the event is always available. By calling the .NET function `FindForm()` on the sender object one can access the Form object easily. Alternatively (if only one Form is involved) the built-in property `lastform` can be used.

  Note that the use of `lastform` is not constrained to event handlers (as was the case with `TheView`). It can be used everywhere in script code.

Given below is a list of methods and properties of the `TheView` object, each accompanied by an alternative mechanism offered by the new Scripting Environment.

### Methods

The following methods were provided by the `TheView` object and must be migrated as explained:

- `Cancel()`
  In the new scripting environment the same can be achieved with: `lastform.Close();` // Use .NET `Form.Close()`

- `IsFormOpen(Name as String) as Boolean`
  Since for .NET Forms there is a distinction between showing a Form and instantiating a Form, the previous concept does not directly translate. Instead the user can ask if a certain Form is currently shown. For example:

  ```java
  var objFormPencilSelector = CreateForm("PencilSelector");
  var objFormColorSelector = CreateForm("ColorSelector");
  ...

  if (objFormColorSelector.Visible)
  {
    ...
  }
  ```

- `FormFind(Name as String) as Object`
  The new Scripting Environment allows you to instantiate more Forms of the same kind. In the old
Scripting Environment each Form could only exist once (as a Singleton). Thus there is no equivalent of FormFind(). In the new Scripting Environment.

OpenDoc(File as String)
The same can be achieved with: Application.OpenDocument( File as String )

PumpData()
This corresponds to the built-in function doevents() which processes all Windows messages currently in the message queue.

RunClick(), RunInitialize(), RunTerminate()
There is no direct replacement for these methods. Call the corresponding handlers directly instead.

Properties
The following properties were provided by the TheView object and must be migrated as explained:

ToolTipText as String
To use tooltips in the new scripting environment, the .NET infrastructure can be used. This allows fine-grained control of tooltip behaviour (adjusting delays, when to show, etc). For example, to provide tooltips for a Form with two controls, the following code could be added to the Form's Load event handler:

```javascript
//Occurs whenever the user loads the form.
function MyForm_Load( objSender, e_EventArgs )
{
    // Create the ToolTip and associate with the Form container.
    var toolTip = CLR.Create("System.Windows.Forms.ToolTip");

    // Set up the delays for the ToolTip.
    toolTip.AutoPopDelay = 3000;
    toolTip.InitialDelay = 1000;
    toolTip.ReshowDelay = 500;

    // Force the ToolTip text to be displayed whether or not
    // the form is active.
    toolTip.ShowAlways = true;

    // Set up the ToolTip text for several Controls.
    toolTip.SetToolTip(objSender.ProgressBar1,                  
        "Shows the progress of the operation");
    toolTip.SetToolTip(objSender.Button1,                       
        "Click Button to start the processing");
}
```

Color as Long
Since all Form/controls in the new Scripting Environment are .NET controls from the System.Windows.Forms namespace, the possibilities to modify colors, background image, fonts, and all other visual aspects are numerous. For example, every Visual Component has the properties BackColor and ForeColor to modify the visual appearance. The following handler could be used to change the color of a button at runtime:
function TestForm_Button1_Click( objSender, e_EventArgs )
{
}

Please refer to the .NET documentation to find out more about this topic:
2 IDE Plugins

XMLSpy allows you to create your own IDE plug-ins and integrate them into XMLSpy.

Use plug-ins to:

- Configure your version of XMLSpy, add commands through menus, icons, buttons etc.
- React to events from XMLSpy.
- Run your specific code within XMLSpy with access to the complete XMLSpy API

XMLSpy expects your plug-in to implement the IXMLSpyPlugin interface. C# and C++ examples are included with your installation package and are located in the XMLSpy2019\Examples\IDEPlugin folder of your XMLSpy installation.

| Windows 7, 8, 10 | C:/Users/<username>/Documents |

See ATL sample files for an example using C++.
2.1 Registration of IDE PlugIns

XMLSpy maintains a specific key in the Registry where it stores all registered IDE plug-ins:

```plaintext
HKEY_CURRENT_USER\Software\Altova\XML Spy\PlugIns
```

All values of this key are treated as references to registered plug-ins and must conform to the following format:

- **Value name:** ProgID of the plug-in
- **Value type:** must be REG_SZ
- **Value data:** CLSID of the component

Each time the application starts the values of the "PlugIns" key is scanned, and the registered plug-ins are loaded.

**Register plug-in manually**

To register a plug-in manually, use the "Customize" dialog box of the XMLSpy "Tools" menu. Use the "Add Plug-In..." button to specify the DLL that implements your plug-in. XMLSpy registers the DLL as a COM server and adds the corresponding entry in its "PlugIns" key.

If you experience problems with manual registration you can check if the CLSID of your plug-in is correctly registered in the "PlugIns" key. If this is not the case, the name of your plug-in DLL was probably not sufficiently unique. Use a different name or perform direct registration.

**Register plug-in directly**

A plug-in can be directly registered as an IDE plug-in by first registering the DLL and then adding the appropriate value to the "PlugIns" key of XMLSpy during plug-in setup for example. The new plug-in will be activated the next time XMLSpy is launched.

**Creating plug-ins**

Source code for sample plug-ins has been provided in the application's (My) Documents folder: Examples\IDEPlugin folder. To build a plug-in from such source code, do the following:

1. Open the solution you want to build as a plug-in in Visual Studio.
2. Build the plug-in with the command in the Build menu.
3. The plug-in's DLL file that will be created in the Bin or Debug folder. This DLL file is the file that must be added as a plug-in (see above).
2.2 **ActiveX Controls**

ActiveX controls are supported. Any IDE Plugin which is also an ActiveX control will be displayed in a Dialog Control Bar. A sample Plugin that is also an ActiveX control is included in the XMLSpyPlugInActiveX folder in the Examples folder of your application folder.
2.3 **Configuration XML**

The IDE plug-in allows you to change the user interface (UI) of XMLSpy. This is done by describing each separate modification using an XML data stream. The XML configuration is passed to XMLSpy using the `GetUIModifications` method of the `IXMLSpyPlugIn` interface.

The XML file containing the UI modifications for the IDE PlugIn, must have the following structure:

```xml
<ConfigurationData>
  <ImageFile>path To image file</ImageFile>
  <Modifications>
    <Modification>
      ...
    </Modification>
  </Modifications>
</ConfigurationData>
```

You can define icons or toolbar buttons for the new menu items which are added to the UI of XMLSpy by the plug-in. The path to the file containing the images is set using the `ImageFile` element. Each image must be 16 x 16 pixels using max. 256 colors. The image references must be arranged from left to right in a single (`<ImageFile>...` line. The rightmost image index value, is zero.

The `Modifications` element can have any number of `Modification` child elements. Each `Modification` element defines a specific change to the standard UI of XMLSpy. Starting with version 4.3, it is also possible to remove UI elements from XMLSpy.

**Structure of Modification elements**

All Modification elements consist of the following two child elements:

```xml
<Modification>
  <Action>Type of action</Action>
  <UIElement Type="type of UI element">...
  </UIElement>
</Modification>
```

Valid values for the `Action` element are:

- Add - to add the following UI element to XMLSpy
- Hide - to hide the following UI element in XMLSpy
- Remove - to remove the UI element from the "Commands" list box, in the customize dialog

You can combine values of the `Action` element e.g. "Hide Remove"

The `UIElement` element describes any new, or existing UI element for XMLSpy. Possible elements are currently: new toolbars, buttons, menus or menu items. The `type` attribute, defines which UI element is described by the XML element.

**Common UIElement children**

The ID and Name elements are valid for all different types of XML UIElement fragments. It is however possible, to ignore one of the values for a specific type of UIElement e.g. Name is ignored for a separator.
<ID></ID>  
{Name}</Name>

If UIElement describes an existing element of the UI, the value of the ID element is predefined by XMLSpy. Normally these ID values are not known to the public. If the XML fragment describes a new part of the UI, then the ID is arbitrary and the value should be less than 1000. The Name element sets the textual value. Existing UI elements can be identified just by name, for e.g. menus and menu items with associated sub menus. For new UI elements, the Name element sets the caption e.g. the title of a toolbar, or text for a menu item.

**Toolbars and Menus**

To define a toolbar its necessary to specify the ID and/or the name of the toolbar. An existing toolbar can be specified using only the name, or by the ID if it is known. To create a new toolbar both values must be set. The **type** attribute must be equal to “ToolBar”.

```xml
<UIElement Type="ToolBar">
  <ID>1</ID>
  <Name>TestPlugIn</Name>
</UIElement>
```

To specify an XMLSpy menu you need two parameters:

- The ID of the menu bar which contains the menu. If no XML documents are open in the main window, the menu bar ID is 128. If one or more XML documents are open, the menu bar ID is 129.
- The menu name. Menus do not have an associated ID value. The following example defines the "Edit" menu of the menu bar which is active, when at least one XML document is open:

```xml
<UIElement Type="Menu">
  <ID>129</ID>
  <Name>Edit</Name>
</UIElement>
```

An additional element is used if you want to create a new menu. The **Place** element defines the position of the new menu in the menu bar:

```xml
<UIElement Type="Menu">
  <ID>129</ID>
  <Name>PlugIn Menu</Name>
  <Place>12</Place>
</UIElement>
```

A value of -1 for the **Place** element sets the new button or menu item at the end of the menu or toolbar.

**Commands**

If you add a new command, through a toolbar button or a menu item, the UIElement fragment can contain any of these sub elements:

```xml
<MacroName></MacroName>
<Info></Info>
<ImageID></ImageID>
```
If MacroName is specified, XMLSpy searches for a macro with the same name in the scripting environment and executes it each time this command is processed. The Info element contains a short description string which is displayed in the status bar, when the mouse pointer is over the associated command (button or menu item). ImageID defines the index of the icon the external image file. Please note that all icons are stored in one image file.

To define a toolbar button create an UIElement with this structure:

```xml
<UIElement Type="ToolBarItem">
  <!--don't reuse local IDs even the commands do the same-->  
  <ID>5</ID>
  <Name>Open file from repository...</Name>
  <!--Set Place To -1 If this is the first button To be inserted-->  
  <Place>-1</Place>
  <ImageID>0</ImageID>
  <ToolBarID>1</ToolBarID>
   <!--instead of the toolbar ID the toolbar name could be used-->  
  <ToolBarName>TestPlugIn</ToolBarName>
</UIElement>
```

Additional elements to declare a toolbar button are Place, ToolBarID and ToolBarName. ToolBarID and ToolBarName are used to identify the toolbar which contains the new or existing button. The textual value of ToolBarName is case sensitive. The (UIElement) type attribute must equal "ToolBarItem".

To define a menu item, the elements MenuID, Place and Parent are available in addition to the standard elements used to declare a command. MenuID can be either 128 or 129. Please see "Toolbars and Menus" for more information on these values.

The Parent element is used to identify the menu where the new menu entry should be inserted. As sub menu items have no unique Windows ID, we need some other way to identify the parent of the menu item.

The value of the Parent element is a path to the menu item. The text value of the Parent element, must equal the parent menu name of the submenu, where the submenu name is separated by a colon. If the menu has no parent, because its not a submenu, add a colon to the beginning of the name. The type attribute must be set to "Menuitem". Example for an UIElement defining a menu item:

```xml
<UIElement Type="Menuitem">
  <!--the following element is a Local command ID-->  
  <ID>3</ID>
  <Name>Open file from repository...</Name>
  <Place>-1</Place>
  <MenuID>129</MenuID>
  <Parent>:PlugIn Menu</Parent>
  <ImageID>0</ImageID>
</UIElement>
```

XMLSpy makes it possible to add toolbar separators and menus if the value of the ID element is set to 0.
2.4 ATL sample files

The following pages show how to create a simple XMLSpy IDE plug-in DLL using ATL. To build the DLL it is necessary to know about ATL, the wizards that generate new ATL objects, as well as MS VisualStudio.

To access the API the implementation imports the Type Library of XMLSpy. The code reads various properties and calls methods using the smart pointers provided by the #import statement.

In addition, the sample code uses the MFC class CString and the ATL conversion macros such as W2T.

At a glance the steps to create an ATL DLL are as follows:

1. Open VisualStudio and select "New..." from the "File" menu.
2. Select the "Projects" tab.
3. Select "ATL COM AppWizard" and type in a project name.
4. Select "Support for MFC" if you want to use MFC classes, or if you want to create a project for the sample code.

Having created the project files you can add an ATL object to implement the IXMLSpyPlugIn interface:

1. Select "New ATL Object..." from the "Insert" menu.
2. Select "Simple Object" from the wizard and click "Next".
3. Type in a name for the object.
4. On the "Attributes" tab, select "Custom" for the type of interface, and disable Aggregation.

These steps produce the skeleton code for the implementation of the IDE plug-in interface. Please see the following pages on how to modify the code and achieve some basic functionality.

2.4.1 Interface description (IDL)

The IDL of the newly created ATL object contains a declaration for one COM interface.

- This interface declaration must be replaced by the declaration of IXMLSpyPlugIn as shown below.
- The IDL must also contain the definition of the SPYUpdateAction enumeration.
- Replace the generated default interface name, (created by the wizard) with "IXMLSpyPlugIn" in the coclass declaration. The IDL should then look something like the example code below:

Having created the ATL object, you then need to implement the IDE plug-in interface of XMLSpy.

```plaintext
import "oaidl.idl";
import "ocidl.idl";

// ----- please insert the following block into your IDL file ----- 
typedef enum {
  spyEnable = 1,
  spyDisable = 2,
  spyCheck = 4,
  spyUncheck = 8
};
```
} SPYUpdateAction;

// ----- end insert block -----

// ----- E.g. Interface entry automatically generated by the ATL wizard ----- [ object, uuid(AB7CD86A-8145-429A-A1F3-270692E08AFC), helpstring("IXMLSpyPlugIn Interface") pointer_default(unique) ] interface IXMLSpyPlugIn : IUnknown { 
};

// ----- end automatically generated Interface Entry -----

// ----- replace the Interface Entry (shown above) generated for you by the ATL wizard, with the following block ----- 

[ odl, uuid(88F2A622-4B7E-42CD-8D04-3C0E5389DD85), helpstring("IXMLSpyPlugIn Interface") ]

interface IXMLSpyPlugIn : IUnknown {

HRESULT _stdcall OnCommand([in] long nID, [in] IDispatch* pXMLSpy); 

HRESULT _stdcall OnUpdateCommand([in] long nID, [in] IDispatch* pXMLSpy, 
[out, retval] SPYUpdateAction* pAction); 


HRESULT _stdcall GetUIModifications([out, retval] BSTR* pModificationsXML); 

HRESULT _stdcall GetDescription([out, retval] BSTR* pDescription); 
};

// ----- end replace block -----

// ----- The code below is automatically generated by the ATL wizard and will look slightly different in your case ----- 

[ uuid(24FE0D1B-3FC0-494E-B36E-1D4CE412B014), version(1.0), helpstring("XMLSpyIDEPlugInDLL 1.0 Type Library") ]
IDE Plugins

ATL sample files 1046
library XMLSPYIDEPLUGINDLLLib
{
importlib("stdole32.tlb");
importlib("stdole2.tlb");

[
uuid(3800E791-7F6B-4ACD-9E32-2AC184444501),
helpstring("XMLSpyIDEPlugIn Class")
]
coclass XMLSpyIDEPlugIn
{
[default] interface IXMLSpyPlugIn;
// ----- define IXMLSpyPlugIn as the
default interface ----};
};

2.4.2

Class definition
In the class definition of the ATL object, several changes must be made. The class has to derive
from IXMLSpyPlugIn, the "Interface Map" needs an entry for IXMLSpyPlugIn, and the methods of
the IDE plug-in interface must be declared:
#ifndef __XMLSPYIDEPLUGIN_H_
#define __XMLSPYIDEPLUGIN_H_
#include "resource.h"

// main symbols

/////////////////////////////////////////////////////////////////////////////
// CXMLSpyIDEPlugIn
class ATL_NO_VTABLE CXMLSpyIDEPlugIn :
public CComObjectRootEx<CComSingleThreadModel>,
public CComCoClass<CXMLSpyIDEPlugIn, &CLSID_XMLSpyIDEPlugIn>,
public IXMLSpyPlugIn
{
public:
CXMLSpyIDEPlugIn()
{
}
DECLARE_REGISTRY_RESOURCEID(IDR_XMLSPYIDEPLUGIN)
DECLARE_NOT_AGGREGATABLE(CXMLSpyIDEPlugIn)
DECLARE_PROTECT_FINAL_CONSTRUCT()
BEGIN_COM_MAP(CXMLSpyIDEPlugIn)
COM_INTERFACE_ENTRY(IXMLSpyPlugIn)
END_COM_MAP()
// IXMLSpyIDEPlugIn
public:
virtual HRESULT _stdcall OnCommand(long nID, IDispatch* pXMLSpy);
virtual HRESULT _stdcall OnUpdateCommand(long nID, IDispatch* pXMLSpy,
SPYUpdateAction* pAction);

© 2019 Altova Gmb H

Programmers' Reference


virtual HRESULT _stdcall OnEvent(long nEventID, SAFEARRAY **arrayParameters, IDispatch* pXMLSpy, VARIANT* pReturnValue);

virtual HRESULT _stdcall GetUIModifications(BSTR* pModificationsXML);

virtual HRESULT _stdcall GetDescription(BSTR* pDescription);
};
#endif

2.4.3 Implementation

The code below shows a simple implementation of an XMLSpy IDE plug-in. It adds a menu item and a separator (available with XMLSpy) to the Tools menu. Inside the OnUpdateCommand() method, the new command is only enabled when the active document is displayed using the Grid View. The command searches for the XML element which has the current focus, and opens any URL starting with "http://", from the textual value of the element.

Ọụzụ

USES_CONVERSION;

if(nID == 1) {
    IApplicationPtr ipSpyApp;

    if(pXMLSpy) {
        if(SUCCEEDED(pXMLSpy->QueryInterface(__uuidof(IApplication), (void**) &ipSpyApp))) {
            IDocumentPtr ipDocPtr = ipSpyApp->ActiveDocument;

            // we assume that grid view is active
            if(ipDocPtr) {
                IGridViewPtr ipGridPtr = ipDocPtr->GridView;

                if(ipGridPtr) {
                    IXMLDataPtr ipXMLData = ipGridPtr->CurrentFocus;

                    CString strValue = W2T(ipXMLData->TextValue);

                    if(!strValue.IsEmpty() && (strValue.Left(7) == _T("http://")))
                        ::ShellExecute(NULL, _T("open"), W2T(ipXMLData->TextValue), NULL, NULL, SW_SHOWNORMAL);
                }
            }
        }
    }
}
return S_OK;
}

HRESULT CXMLSpyIDEPlugIn::OnUpdateCommand(long nID, IDispatch* pXMLSpy, SPYUpdateAction* pAction)
{
  *pAction = spyDisable;

  if(nID == 1) {
    IApplicationPtr ipSpyApp;

    if(pXMLSpy) {
      if(SUCCEEDED(pXMLSpy->QueryInterface(__uuidof(IApplication),(void**) &ipSpyApp))) {
        IDocumentPtr ipDocPtr = ipSpyApp->ActiveDocument;

        // only enable if grid view is active
        if((ipDocPtr != NULL) && (ipDocPtr->CurrentViewMode == spyViewGrid))
          *pAction = spyEnable;
      }
    }
  }

  return S_OK;
}

HRESULT CXMLSpyIDEPlugIn::OnEvent(long nEventID, SAFEARRAY **arrayParameters, IDispatch* pXMLSpy, VARIANT* pReturnValue)
{
  return S_OK;
}

HRESULT CXMLSpyIDEPlugIn::GetUIModifications(BSTR* pModificationsXML)
{
  CComBSTR bstrMods = _T(" <ConfigurationData>
  <Modifications>");
  // add "Open URL..." to Tools menu
  bstrMods.Append (_T(" <Action>Add</Action>
  <UIElement type="MenuItem">" _T(ID) _T(Name) _T(Place) _T(MenuID) _T(Parent)
  </UIElement>
  </Modification>"));
// add separator to Tools menu
bstrMods.Append(_T("\n  <Modification>\n    <Action>Add</Action>\n    <UIElement type="MenuItem">\n      <ID>0</ID>\n      <Place>1</Place>\n      <MenuID>129</MenuID>\n      <Parent>:Tools</Parent>\n    </UIElement>\n  </Modification>");

// finish modification description
bstrMods.Append(_T("\n  </Modifications>\n</ConfigurationData>");

return bstrMods.CopyTo(pModificationsXML);
}

HRESULT CXMLSpyIDEPlugIn::GetDescription(BSTR* pDescription)
{
  CComBSTR bstrDescr = _T("ATL C++ XMLSpy IDE PlugIn; This PlugIn demonstrates the implementation of a simple ATL DLL as a IDE PlugIn for XMLSpy.");
  return bstrDescr.CopyTo(pDescription);
}
2.5 IXMLSpyPlugIn

See also

Methods
OnCommand
OnUpdateCommand
OnEvent
GetUIModifications
GetDescription

Description
If a DLL is added to XMLSpy as an IDE plug-in, it is necessary that it registers a COM component that answers to an IXMLSpyPlugIn interface with the reserved uuid(88F2A622-4B7E-42CD-8D04-3C0E5389DD85), for it to be recognized as a plug-in.

2.5.1 OnCommand

See also

Declaration: OnCommand (nID as long, pXMLSpy as IDispatch)

Description
The OnCommand() method of the interface implementation, is called each time a command added by the the IDE plug-in (menu item or toolbar button) is processed. nID stores the command ID defined by the ID element of the respective UIElement.

pXMLSpy holds a reference to the dispatch interface of the Application object of XMLSpy.

Example

Public Sub IXMLSpyPlugIn_OnCommand(ByVal nID As Long, ByVal pXMLSpy As Object)
    If (Not (pXMLSpy Is Nothing)) Then
        Dim objDlg
        Dim objDoc As XMLSpyLib.Document
        Dim objSpy As XMLSpyLib.Application
        Set objSpy = pXMLSpy

        If nID = 3 Or nID = 5 Then
            Set objDlg = CreateObject("MSComDlg.CommonDialog")
            objDlg.Filter = "XML Files (*.xml)|*.xml|All Files (*.*)|*.*||"
            objDlg.FilterIndex = 1
            objDlg.ShowOpen

            If Len(objDlg.FileName) > 0 Then
                Set objDoc = objSpy.Documents.OpenFile(objDlg.FileName, False)
            Set objDoc = Nothing
            End If
        End If

        If nID = 4 Or nID = 6 Then
            Set objDlg = CreateObject("MSComDlg.CommonDialog")
            objDlg.Filter = "All Files (*.*)|*.*||"
            objDlg.Flags = cdlOFNPathMustExist
        End If
    End If
objDlg.ShowSave

If Len(objDlg.FileName) > 0 Then
    Set objDoc = objSpy.ActiveDocument
    If Not (objDoc Is Nothing) Then
        objDoc.SetPathName objDlg.FileName
        objDoc.Save
        Set objDoc = Nothing
    End If
End If
End If

Set objSpy = Nothing
End Sub

### 2.5.2 OnUpdateCommand

See also

**Declaration:** `OnUpdateCommand(nID as long, pXMLSpy as IDispatch) as SPYUpdateAction`

**Description**
The `OnUpdateCommand()` method is called each time the visible state of a button or menu item needs to be set. `nID` stores the command ID defined by the `ID` element of the respective `UIElement`.

`pXMLSpy` holds a reference to the dispatch interface of the `Application` object.

Possible return values to set the update state are:

- `spyEnable` = 1
- `spyDisable` = 2
- `spyCheck` = 4
- `spyUncheck` = 8

**Example**

```vba
Public Function IXMLSpyPlugIn_OnUpdateCommand(ByVal nID As Long, ByVal pXMLSpy As Object) As SPYUpdateAction
    IXMLSpyPlugIn_OnUpdateCommand = spyDisable
    If (Not (pXMLSpy Is Nothing)) Then
        Dim objSpy As XMLSpyLib.Application
        Set objSpy = pXMLSpy
        If nID = 3 Or nID = 5 Then
            IXMLSpyPlugIn_OnUpdateCommand = spyEnable
        End If
        If nID = 4 Or nID = 6 Then
            If objSpy.Documents.Count > 0 Then
                IXMLSpyPlugIn_OnUpdateCommand = spyEnable
            Else
                IXMLSpyPlugIn_OnUpdateCommand = spyDisable
            End If
        End If
    End If
End Function
```
IXMLSpyPlugIn_OnUpdateCommand = spyDisable
End If
End If
End If
End Function

2.5.3 OnEvent

See also

*Declaration:* OnEvent(*nEventID* as long, *arrayParameters* as SAFEARRAY(VARIANT),
*pXMLSpy* as IDispatch) as VARIANT

**Description**

OnEvent() is called each time an event is raised from XMLSpy.

Possible values for *nEventID* are:

- **On_BeforeStartEditing** = 1
- **On_EditingFinished** = 2
- **On_FocusChanged** = 3
- **On_Beforedrag** = 4
- **On_BeforeDrop** = 5
- **On_OpenProject** = 6
- **On_OpenDocument** = 7
- **On_CloseDocument** = 8
- **On_SaveDocument** = 9

Events available since XMLSpy 4r4:

- **On_DocEditDragOver** = 10
- **On_DocEditDrop** = 11
- **On_DocEditKeyDown** = 12
- **On_DocEditKeyUp** = 13
- **On_DocEditKeyPressed** = 14
- **On_DocEditMouseMove** = 15
- **On_DocEditButtonUp** = 16
- **On_DocEditButtonDown** = 17
- **On_DocEditContextMenu** = 18
- **On_DocEditPaste** = 19
- **On_DocEditCut** = 20
- **On_DocEditCopy** = 21
- **On_DocEditClear** = 22
- **On_DocEditSelectionChanged** = 23

Events available since XMLSpy 2004:

- **On_DocEditDragOver** = 10
Events available since XMLSpy 2004r4 (type library version 1.4):

- On_BeforeOpenProject = 25
- On_BeforeOpenDocument = 26
- On_BeforeSaveDocument = 27
- On_BeforeCloseDocument = 28
- On_ViewActivation = 29
- On_DocEditKeyboardEvent = 30
- On_DocEditMouseEvent = 31

Events available since XMLSpy 2006 SP1 (type library version 1.5):

- On_BeforeValidate = 32

Events available since XMLSpy 2007 (type library version 1.6):

- On_BeforeShowSuggestions = 33
- On_ProjectOpened = 34
- On_Char = 35

Events available since XMLSpy 2009 (type library version 2.2):

- On_Init = 36
- On_Running = 37
- On_Shutdown = 38

Events available since XMLSpy 2012 (type library version 2.8):

- On_AuthenticBeforeSave = 39
- On_AuthenticContextMenuActivated = 40
- On_AuthenticLoad = 41
- On_AuthenticToolbarButtonClicked = 42
- On_AuthenticToolbarButtonExecuted = 43
- On_AuthenticUserAddedXMLNode = 44

The names of the events are the same as they appear in the Scripting Environment of XMLSpy. For IDE plug-ins the names used are immaterial. The events are identified using the ID value.

arrayParameters is an array which is filled with the parameters of the currently raised event. Order, type and meaning of the single parameters are available through the scripting environment of XMLSpy. The events module of a scripting project, contains predefined functions for all events prior to version 4.4. The parameters passed to the predefined functions are identical to the array elements of the arrayParameters parameter.

Events raised from the Authentic View of XMLSpy do not pass any parameters directly. An "event" object is used instead. The event object can be accessed through the Document object of the active document.

pXMLSpy holds a reference to the dispatch interface of the Application object of XMLSpy.

If the return value of OnEvent() is set, then neither the IDE plug-in, nor an event handler inside of the scripting environment will get this event afterwards. Please note that all IDE plug-ins get/
process the event before the Scripting Environment does.

2.5.4 GetUIModifications

See also

**Declaration:** GetUIModifications() as String

**Description**
The GetUIModifications() method is called during initialization of the plug-in, to get the configuration XML data that defines the changes to the UI of XMLSpy. The method is called when the plug-in is loaded for the first time, and at every start of XMLSpy.

See also Configuration XML for a detailed description how to change the UI.

**Example**

```vba
Public Function IXMLSpyPlugIn_GetUIModifications() As String
    ' GetUIModifications() gets the XML file with the specified modifications of
    ' the UI from the config.xml file in the plug-in folder
    Dim strPath As String
    strPath = App.Path
    If Len(strPath) > 0 Then
        Dim fso As New FileSystemObject
        Dim file As file
        Set file = fso.GetFile(strPath & "\config.xml")
        If (Not (file Is Nothing)) Then
            Dim stream As TextStream
            Set stream = file.OpenAsTextStream(ForReading)
            ' this replaces the token "**path**" from the XML file with
            ' the actual installation path of the plug-in to get the image file
            Dim strMods As String
            strMods = stream.ReadAll
            strMods = Replace(strMods, "**path**", strPath)
            IXMLSpyPlugIn_GetUIModifications = strMods
        Else
            IXMLSpyPlugIn_GetUIModifications = ""
        End If
    End If
End Function
```

2.5.5 GetDescription

See also

**Declaration:** GetDescription() as String

**Description**
GetDescription() is used to define the description string for the plug-in entries visible in the Customize dialog box.
Example

Public Function IXMLSpyPlugIn_GetDescription() As String
    IXMLSpyPlugIn_GetDescription = "Sample Plug-in for XMLSpy; This Plug-in demonstrates the implementation of a simple VisualBasic DLL as a Plug-in for XMLSpy."
End Function
3 Application API

The COM-based API of XMLSpy (also called the Application API from now on) enables other applications to use the functionality of XMLSpy. As a result, it is possible to automate a wide range of tasks, from validating an XML file to modifying complex XML content (with the XMLData interface).

XMLSpy and its Application API follow the common specifications for automation servers set out by Microsoft. It is possible to access the methods and properties of the Application API from common development environments, such as those using C, C++, VisualBasic, and Delphi, and with scripting languages like JScript and VBScript.

Execution environments for the Application API

The Application API can be accessed from the following execution environments:

- External programs (described below and in the Overview part of this section)
- From within the built-in Scripting Editor of XMLSpy. For a description of the scripting environment, see the section, Scripting Editor.
- XMLSpy allows you to create and integrate your own plug-ins into the application using a special interface for plug-ins. A description of how to create plug-ins is given in the section IDE Plug-ins.
- Via an ActiveX Control, which is available if the integration package is installed. For more information, see the section ActiveX Integration.

External programs

In the Overview part of this section, we describe how the functionality of XMLSpy can be accessed and automated from external programs.

Using the Application API from outside XMLSpy requires an instance of XMLSpy to be started first. How this is done depends on the programming language used. See the section, Programming Languages, for information about individual languages.

Essentially, XMLSpy will be started via its COM registration. Then the Application object associated with the XMLSpy instance is returned. Depending on the COM settings, an object associated with an already running XMLSpy can be returned. Any programming language that supports creation and invocation of COM objects can be used. The most common of these are listed below.

- JScript and VBScript script files have a simple syntax and are designed to access COM objects. They can be run directly from a DOS command line or with a double click on Windows Explorer. They are best used for simple automation tasks.
- C# is a full-fledged programming language that has a wide range of existing functionality. Access to COM objects can be automatically wrapped using C#.
- C++ provides direct control over COM access but requires relatively larger amounts of code than the other languages.
- Java: Altova products come with native Java classes that wrap the Application API and provide a full Java look-and-feel.
- Other programming languages that make useful alternatives are: Visual Basic for Applications, Perl, and Python.
Programming points
The following limitations must be considered in your client code:

- Be aware that if your client code crashes, instances of XMLSpy may still remain in the system.
- Don't hold references to objects in memory longer than you need them, especially those from the XMLData interface. If the user interacts between two calls of your client, then there is no guarantee that these references are still valid.
- Don't forget to disable dialogs if the user interface is not visible.
- See Error handling in JScript (and in C# and Java) for details of how to avoid annoying error messages.
- Free references explicitly if you are using C or C++.

This documentation
This documentation section about the Application API is broadly divided into two parts.

- The first part consists of an Overview, which describes the object model for the API and explains how the API is accessed via various programming languages.
- The second part is a reference section (Interfaces and Enumerations) that contains descriptions of the interface objects of the Application API.
3.1 Overview

This overview of the Application API is organized as follows:

- The Object Model describes the relationships between the objects of the Application API.
- Programming Languages explains how the most commonly used programming languages (JScript, VBScript, C#, and Java) can be used to access the functionality of the Application API. Code listings from the example files supplied with your application package are used to describe basic mechanisms.
- The DOM and XMLData explains the relationship between the Application API's XMLData interface and the DOM.
- Obsolete: Authentic View Row Operations supplies information about obsolete objects for Authentic View table row operations.
- Obsolete: Authentic View Editing Operations supplies information about obsolete objects for Authentic View editing operations.

3.1.1 Object Model

The starting point for every application which uses the Application API is the Application object. This object contains general methods like import/export support and references to the open documents and any open project.

The Application object is created differently in various programming languages. In scripting languages such as JScript or VBScript, this involves calling a function which initializes the application's COM object. For examples, see the Programming Languages section.

The picture below shows the links between the main objects of the Application API:

The application object consists of the following parts:
2. Reference to current project and methods for creating and opening projects.
3. Methods to support the export to and import from databases, text files, and Word documents.
4. URL management.
5. Methods for macro menu items.

Once you have created an Application object you can start using the functionality of XMLSpy. In most cases, you either open a project and access the documents from there or you directly open a document via the Documents interface.

### 3.1.2 Programming Languages

Programming languages differ in the way they support COM access. A few examples for the most frequently used languages ([links below](#)) will help you get started. The code listings in this section show how basic functionality can be accessed. This basic functionality is included in the files in the API Examples folder and can be tested straight away. The path to the API Examples folder is given below:

```
Windows 7, Windows 8, Windows 10
C:\Users\<username>\Documents\Altova\XMLSpy2019\n```

**JScript**

The JScript listings demonstrate the following basic functionality:

- Start application or attach to a running instance
- Simple document access
- Iteration
- Error handling
- Events
- Import and export of data

**VBScript**

VBScript is different than JScript only syntactically; otherwise it works in the same way. The listings below describe is an example of how VBScript can be used. For more information, refer to the [JScript examples](#).

- **Events**: Shows how events are handled using VBScript.

**C#**

C# can be used to access the Application API functionality. The code listings show how to access the API for certain basic functionality.

- **Start XMLSpy**: Starts XMLSpy, which is registered as an automation server, or activates the program if XMLSpy is already running.
- **Open OrgChart.pxf**: Locates one of the example documents installed with XMLSpy and opens it. If this document is already open it becomes the active document.
- **OnDocumentOpened Event On/Off**: Shows how to listen to XMLSpy events. When turned on, a message box will pop up after a document has been opened.
• **Open ExpReport.xml**: Opens another example document.
• **Toggle View Mode**: Changes the view of all open documents between Text View and Authentic View. The code shows how to iterate through open documents.
• **Validate**: Validates the active document and shows the result in a message box. The code shows how to handle errors and COM output parameters.
• **Shutdown XMLSpy**: Stops XMLSpy.

**Java**
The XMLSpy API can be accessed from Java code. The Java sub-section of this section explains how some basic XMLSpy functionality can be accessed from Java code. It is organized into the following sub-sections:

- Mapping Rules for the Java Wrapper
- Example Java Project
- Application Startup and Shutdown
- Simple Document Access
- Iterations
- Use of Out-Parameters
- Event Handlers

**JScript**
This section contains listings of JScript code that demonstrate the following basic functionality:

- Start application or attach to a running instance
- Simple document access
- Iteration
- Error handling
- Events
- Import and export of data

**Example files**
The code listings in this section are available in example files that you can test as is or modify to suit your needs. The JScript example files are located in the JScript folder of the API Examples folder:

| Windows 7, Windows 8, Windows 10 | C:\users\<username>\documents\Altova\XMLSpy2019\ |

The example files can be run in one of two ways:

- *From the command line*: Open a command prompt window, change the directory to the path above, and type the name of one of the example scripts (for example, `Start.js`).
- *From Windows Explorer*: In Windows Explorer, browse for the JScript file and double-click it.

The script is executed by Windows Script Host that is packaged with Windows operating system. For more information about Windows Script Host, refer to MSDN documentation ([https://msdn.microsoft.com](https://msdn.microsoft.com)).
1061

Application API

Overview

Start Application
The JScript below starts the application and shuts it down. If the COM object of the 32-bit
XMLSpy cannot be found, the code attempts to get the COM object of the 64-bit application;
otherwise, an error is thrown. If an instance of the application is already running, the running
instance will be returned.
Note:

For 32-bit XMLSpy, the registered name, or programmatic identifier (ProgId) of the COM
object is XMLSpy.Application. For 64-bit XMLSpy, the name is
XMLSpy_x64.Application.

// Initialize application's COM object. This will start a new instance of the
application and
// return its main COM object. Depending on COM settings, the main COM object
of an already
// running application might be returned.
try {
objSpy = WScript.GetObject("", "XMLSpy.Application");
catch(err) {}

}

if( typeof( objSpy ) == "undefined" )
{
try
{
objSpy = WScript.GetObject("", "XMLSpy_x64.Application")
catch(err)
{
WScript.Echo( "Can't access or create XMLSpy.Application" );
WScript.Quit();
}
}

}

// if newly started, the application will start without its UI visible. Set it
to visible.
objSpy.Visible = true;
WScript.Echo(objSpy.Edition + " has successfully started. ");
objSpy.Visible = false;
COM connections
//objSpy.Visible = true;

// will shutdown application if it has no more
// will keep application running with UI visible

The JScript code listed above is available in the sample file Start.js (see Example Files).
Simple Document Access
After you have started the application as shown in Start Application, you will most likely want to
programmatically open a document in order to work with it. The JScript code listing below
illustrates how to open two documents from the XMLSpy Examples folder and set one of them as
the active document.
// Locate examples via USERPROFILE shell variable. The path needs to be
adapted to major release versions.
objWshShell = WScript.CreateObject("WScript.Shell");
majorVersionYear = objSpy.MajorVersion + 1998
strExampleFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\
\My Documents\\Altova\\XMLSpy" + majorVersionYear + "\\Examples\\";

Altova XMLSpy 2019 Professional Edition

© 2019 Altova Gmb H


// Tell XMLSpy to open two documents. No dialogs
objDoc1 = objSpy.Documents.OpenFile(strExampleFolder + "OrgChart.pxf", false);
objSpy.Documents.OpenFile(strExampleFolder + "ExpReport.xml", false);

// The document currently active can be easily located.
objDoc2 = objSpy.ActiveDocument;

// Let us make sure that the document is shown in grid view.
objDoc2.SwitchViewMode(0); // SPYViewModes.spyViewGrid = 0

// Now switch back to the document opened first
objDoc1.SetActiveDocument();

The JScript code listed above is available in the sample file DocumentAccess.js (see Example Files).

Iteration
The JScript listing below shows how to iterate through the open documents. It is assumed that you have already started the application and opened some documents as shown in the previous sections.

// go through all open documents using a JScript Enumerator
bRequiresSaving = false;
for (var iterDocs = new Enumerator(objSpy.Documents); !iterDocs.atEnd(); iterDocs.moveNext())
{
    if (iterDocs.item().IsModified)
        bRequiresSaving = true;

    var strErrorText = new Array(1);
    var nErrorNumber = new Array(1);
    var errorData = new Array(1);

    if (!iterDocs.item().IsValid(strErrorText, nErrorNumber, errorData))
    {
        var text = strErrorText;
        // access that XMLData object only if filled in
        if (errorData[0] !== null)
            text += "(" + errorData[0].Name + "/" + errorData[0].TextValue + ")";

        WScript.Echo("Document ": " + iterDocs.item().Name + " validation error[" + nErrorNumber + ": " + text);
    }
    else
    {
        // The COM call succeeded and the document is valid.
        WScript.Echo("Document ": " + iterDocs.item().Name + " is valid.");
    }
}

// go through all open documents using index-based access to the document collection
for (i = objSpy.Documents.Count; i > 0; i--)
    objSpy.Documents.Item(i).Close(false);
The JScript code listed above is available in the sample file `DocumentAccess.js` (see Example Files).

**Error Handling**

The Application API returns errors in two different ways:

- The HRESULT returned by every API method
- The IErrorInfo interface of the Application API

Every API method returns an HRESULT. This return value gives the caller information about errors during execution of the method. If the call was successful, the return value is S_OK. The HRESULT option is commonly used in C/C++ programs.

However, programming languages such as VisualBasic and scripting languages (and other high-level development environments) don't give the programmer access to the HRESULT return of a COM call. Such languages use the IErrorInfo interface, which is also supported by the Application API. If an error occurs, the Application API creates a new object that implements the IErrorInfo interface. The information provided by the IErrorInfo interface is imported by the development environment into its own error-handling mechanism.

For example, the JScript code listing below causes an error to be thrown by incorrectly declaring an array. Additional information about the error object is provided by its properties number and description.

```javascript
try {
    var arr = new Array(-1);
} catch (err) {
    WScript.Echo("Error : (" + (err.number & 0xffff) + ")" + err.description);
}
```

**Events**

COM specifies that a client must register itself at a server for callbacks using the connection point mechanism. The automation interface for XMLSpy defines the necessary event interfaces. The way to connect to those events depends on the programming language you use in your client. The following code listing shows how this is done using JScript.

The method `WScript.ConnectObject` is used to receive events.

```javascript
// The event-handler function
function DocEvent_OnBeforeCloseDocument(objDocument) {
    WScript.Echo("Received event - before closing document");
}

// Create or connect to XMLSpy (or Authentic Desktop)
try {
    // Create the environment and XMLSpy (or Authentic Desktop)
    objWshShell = WScript.CreateObject("WScript.Shell");
    objFSO = WScript.CreateObject("Scripting.FileSystemObject");
    objSpy = WScript.GetObject('', "XMLSpy.Application");
    /* objSpy = WScript.CreateObject('XMLSpy.Application'); */
}
```

// If only Authentic Desktop is installed (and XMLSpy is not installed) use:
// objSpy = WScript.GetObject("", "AuthenticDesktop.Application")

} catch(err)
    { WScript.Echo ("Can't create WScript.Shell object or XMLSpy"); }

// Create document object and connect to its events
objSpy.Visible = true;
majorVersionYear = objSpy.MajorVersion + 1998
docPath = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\Documents\Altova\XMLSpy" + majorVersionYear + "\Examples\ExpReport.xml"
objDoc = objSpy.Documents.OpenFile (docPath, false);
WScript.ConnectObject(objDoc, "DocEvent_");

// Keep running while waiting for the event
// In the meanwhile close this document in XMLSpy (or Authentic Desktop) manually
WScript.Echo ("Sleeping for 10 seconds ...");
WScript.Sleep (10000);
objDoc = null;
WScript.Echo ("Stopped listening for event");
objSpy.Quit();

**Import and Export of Data**

Before you implement your import and export tasks with the Application API, it is good practice to test the connections, parameters, SQL queries and so on in XMLSpy. In this way you are able to verify the results and make quick adjustments to import or export parameters.

Most of the methods for importing and exporting data are placed in the **Application** object, the remaining functions are accessible via the **Document** interface.

There is some preparatory work necessary, before the actual import or export can be started. Every import/export job consists of two parts. You need to define a connection to your data and the specific behaviour for the import/export process.

In case of an import, the connection is either a database, a text-file or a Word document. The behaviour is basically which data (columns) should be imported in XMLSpy.

In case of an export, the connection is either a database or a text file. Specify which data (elements of the XML file) and additional parameters (for example, automatic key generation or number of sub-levels) to use from the XML-structure.

The properties in the **DatabaseConnection**, **TextImportExportSettings** and **ExportSettings** interfaces have default values. See the corresponding descriptions in the **Interfaces** chapter for further information.

The sub-sections of this section describe each of these operations in detail.

- Import from Database
- Export to Database
- Import from Text
- Export to Text
Given below are the steps to establish a connection to an existing database for import:

1. Use a `DatabaseConnection` object and set the properties:
   
   The method `Application.GetDatabaseSettings` returns a new object for a database connection:
   
   ```
   objImpSettings = objSpy.GetDatabaseSettings();
   ```
   
   You have to set either an **ADO connection string**,
   
   ```
   objImpSettings.ADOConnection = strADOConnection
   ```
   
   or the **path** to an existing database file:
   
   ```
   objImpSettings.File = strExampleFolder + "Tutorial\Company.mdb";
   ```
   
   To complete the settings you create a **SQL select statement** to define the data to be queried:
   
   ```
   objImpSettings.SQLSelect = "SELECT * FROM Address";
   ```
   
2. Call `Application.GetDatabaseImportElementList` to get a collection of the resulting columns of the SQL query:
   
   ```
   objElementList = objSpy.GetDatabaseImportElementList(objImpSettings);
   ```
   
   This collection gives you the opportunity to control which columns should be imported and specify the datatype of the new elements. Each item of the collection represents one column to import. If you remove an item, the corresponding column will not be imported. You can additionally modify the `ElementList.Item.ElementKind` property to set the datatype of the XML elements for each column.

   Please consider that `GetDatabaseImportElementList()` executes the SQL query and could initiate a time-consuming call. To avoid this, it is possible to pass a null-pointer as the second parameter to `ImportFromDatabase();` this imports all columns as plain XML elements.

3. Start the import with `Application.ImportFromDatabase`:
   
   ```
   objImpDocFromDB = objSpy.ImportFromDatabase(objImpSettings, objElementList);
   ```

   // Locate examples via USERPROFILE shell variable.
   objWshShell = WScript.CreateObject("WScript.Shell");
   majorVersionYear = objSpy.MajorVersion + 1998
   strExampleFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\My Documents\Altova\XMLSpy" + majorVersionYear + "\Examples\";

   try
   {
   // specify the source of data import
   objImpSettings = objSpy.GetDatabaseSettings();
   objImpSettings.File = strExampleFolder + "Tutorial\Company.mdb";
   objImpSettings.SQLSelect = "SELECT * FROM Address";
   
   // column filter
   objElementList = objSpy.GetDatabaseImportElementList(objImpSettings);
   
   // import into a new XML file
The JScript code listed above is available in the sample file ImportExport.js (see Example Files).

To export data to a database, carry out the steps below:

1. Use a **DatabaseConnection** object and set the necessary properties. All properties except **SQLSelect** are important for the export. **ADOConnection** or **File** defines the target for the output. You need to set only one of them.

2. Fill an **ExportSettings** object with the required values. These properties are the same options as those available in the export dialog of XMLSpy. Select the menu option **Convert | Export to Text files/Database** to see the options and try a combination of export settings. After that it is easy to transfer these settings to the properties of the interface.

   Call **Application.GetExportSettings** to get an **ExportSettings** object:

   ```javascript
   objExpSettings = objSpy.GetExportSettings();
   ```

3. Build an element list with **Document.GetExportElementList**. The element list enables you to eliminate XML elements from the export process. It also gives you information about the record and field count in the **RecordCount** and **FieldCount** properties. Set the **ExportSettings.ElementList** property to this collection. It is possible to set the element list to null/Nothing (default) to export all elements.

4. Call **Document.ExportToDatabase** to execute the export. The description of the **ExportToDatabase** method contains also a code example for a database export.

   ```javascript
   // set the behaviour of the export with ExportSettings
   objExpSettings = objSpy.GetExportSettings();

   // set the destination with DatabaseConnection
   objDB = objSpy.GetDatabaseSettings();
   objDB.CreateMissingTables = true;
   objDB.CreateNew = true;
   objDB.File = "C:\Temp\Export.mdb";

   try
   {
     objImpDocFromDB.ExportToDatabase(objImpDocFromDB.RootElement, objExpSettings, objDB);
   }
   catch (err) {
   }
   ```
The JScript code listed above is available in the sample file ImportExport.js (see Example Files).

Importing data from a text file is similar to the import from a database. You must use other interfaces (described in steps 1 to 3 below) with different methods and properties:

1. Use a TextImportExportSettings object and set the properties:
   The method Application.GetTextImportExportSettings returns a new object to specify a text file for import.

   ```javascript
   objImpSettings = objSpy.GetTextImportExportSettings();
   ```
   
   You have to set at least the ImportFile property to the path of the file for the import. Another important property is HeaderRow. Set it to False if the text file does not contain a leading line as a header row.

   ```javascript
   objImpSettings.ImportFile = strExampleFolder + "Tutorial\Shapes.txt";
   ```

2. Call Application.GetTextImportElementList to get a collection of all columns inside the text file:

   ```javascript
   objElementList = objSpy.GetTextImportElementList(objImpSettings);
   ```

3. Start the import with Application.ImportFromText.

   ```javascript
   objImpDocFromText = objSpy.ImportFromText(objImpSettings, objElementList);
   ```

   ```javascript
   try {
       // specify the source of data import
       objImpSettings = objSpy.GetTextImportExportSettings();
       objImpSettings.ImportFile = strExampleFolder + "Tutorial\Shapes.txt";
       objImpSettings.HeaderRow = false;

       // column filter
       objElementList = objSpy.GetTextImportElementList(objImpSettings);

       // import into a new XML file
       objImpDocFromText = objSpy.ImportFromText(objImpSettings, objElementList);
   } catch (err) {
       WScript.Echo("Error importing from text file.\n\n" + "Error: " + (err.number & 0xffff) + "\n" + "Description: " + err.description);
   }
   ```

   The JScript code listed above is available in the sample file ImportExport.js (see Example Files).
To export data to text, carry out the steps below:

1. Use a `TextImportExportSettings` object and set the necessary properties.
2. Fill an `ExportSettings` object with the required values. See Item 2 in Export to database.
4. Call `Document.ExportToText` to execute the export.

```javascript
objExpSettings = objSpy.GetExportSettings();
objExpSettings.ElementList =
objImpDocFromText.GetExportElementList(objImpDocFromText.RootElement,
objExpSettings);

objTextExp = objSpy.GetTextImportExportSettings();
objTextExp.HeaderRow = true;
objTextExp.DestinationFolder = "C:\Temp";

try
{
    objImpDocFromText.ExportToText(objImpDocFromText.RootElement,
    objExpSettings, objTextExp);
} catch(err)
{
    WScript.Echo("Error exporting to text.\n\n" +
    "Error: " + (err.number & 0xffff) + "\n" +
    "Description: " + err.description);
}
```

The JScript code listed above is available in the sample file `ImportExport.js` (see Example Files).

**VBScript**

VBScript is syntactically different than JScript but works in the same way. This section contains a listing showing how events are used with VBScript and an example.

For information about other functionality, refer to the JScript examples listed below:

- Start application or attach to a running instance
- Simple document access
- Iteration
- Error handling
- Import and export of data

**Events**

COM specifies that a client must register itself at a server for callbacks using the connection point mechanism. The automation interface for XMLSpy defines the necessary event interfaces. The way to connect to those events depends on the programming language you use in your client. The following code listing shows how this is done using VBScript.

The method `WScript.ConnectObject` is used to receive events.
To run this code, paste it into a file with .vbs extension, and either double-click in Windows Explorer, or run it from a command prompt.

```vbs
' the event handler function
    Call WScript.Echo("received event - before closing document")
End Function

' create or connect to XmlSpy
Set objWshShell = WScript.CreateObject("WScript.Shell")
Set objFSO = WScript.CreateObject("Scripting.FileSystemObject")
Set objSpy = WScript.GetObject("", "XMLSpy.Application")
' If only Authentic is installed (and XMLSpy is not installed) use:
' Set objSpy = WScript.GetObject("", "AuthenticDesktop.Application")
' If only XMLSpy 64-bit is installed, use:
' Set objSpy = WScript.GetObject("", "XMLSpy_x64.Application")

' create document object and connect to its events
objSpy.Visible = True

' Find out user's personal folder and locate one of the installed examples.
personalFolder = objWshShell.ExpandEnvironmentStrings("%UserProfile%")
majorVersionYear = objSpy.MajorVersion + 1998
xmlspyExamplesFolder = personalFolder & "\Documents\Altova\XMLSpy" & majorVersionYear & "\Examples\"
docPath = xmlspyExamplesFolder & "ExpReport.xml"

' open a document
Set objDoc = objSpy.Documents.OpenFile (docPath, False)
Call WScript.ConnectObject(objDoc, "DocEvent_")

' keep running while waiting on the event
' in the meantime close the document in XMLSPY manually
Call WScript.Echo("sleeping for 10 seconds ...")
Call WScript.Sleep (10000)

Set objDoc = Nothing
Call WScript.Echo("stopped listening for event")
Call objSpy.Quit
```

**Note:** For 32-bit XMLSpy, the registered name, or programmatic identifier (ProgId) of the COM object is XMLSpy.Application. For 64-bit XMLSpy, the name is XMLSpy_x64.Application.

**Example: Using Events**

Authentic View supports event connection on a per-object basis. Implementation of this feature is based on COM connection points and is available in environments that support this mechanism.

The following example is a VBScript code example that shows how to use events from within a VBScript project.

```vbs
' ----------
' VBScript example that demonstrates how to use events.----------
```
' Event handler for OnSelectionChanged event of AuthenticView
Function AuthenticViewEvent_OnSelectionChanged(objAuthenticRange)
    If objAuthenticRange.FirstTextPosition <>
        objAuthenticRange.LastTextPosition Then
        Call WScript.Echo("Selection: " & objAuthenticRange.Text & vbNewLine &
            vbNewLine & "Close this dialog.")
    Else
        Call WScript.Echo("Cursor position: " &
            objAuthenticRange.FirstTextPosition & vbNewLine & vbNewLine &
            "Close this dialog.")
    End If
End Function

' Start/access XMLSpy and connect to its automation interface.
Set WshShell = WScript.CreateObject("WScript.Shell")
Set objSpy = GetObject("", "XMLSpy.Application")
' Make the UI of XMLSpy visible.
objSpy.Visible = True

' Find out user's personal folder and locate one of the installed XMLSpy
examples.
personalFolder = WshShell.ExpandEnvironmentStrings("%UserProfile%")
majorVersionYear = objSpy.MajorVersion + 1998
xmlspyExamplesFolder = personalFolder & "\Documents\Altova\XMLSpy" &
    majorVersionYear & "\Examples" docPath = xmlspyExamplesFolder & "ExpReport.xml"

' Create object to access windows file system and test if the our document
exists.
Set fso = CreateObject("Scripting.FileSystemObject")
If fso.FileExists(docPath) Then
    ' open the document
    Call objSpy.Documents.OpenFile(docPath, False)
    set objDoc = objSpy.ActiveDocument

    ' switch active document to authentic view
    objDoc.SwitchViewMode 4 ' spyViewAuthentic

    ' Register for connection point events on the authentic view of the active
document.
    ' Any function with a valid event name prefixed with "AuthenticViewEvent_
    will
    ' be called when the corresponding event gets triggered on the specified
    object.
    set objView = objDoc.AuthenticView
    Call WScript.ConnectObject(objView, "AuthenticViewEvent_")
    Call WScript.Echo("Events are connected." & vbNewLine & vbNewLine &
        "Now set or move the cursor in XMLSpy." & vbNewLine &
        vbNewLine & "Close this
dialog to shut down XMLSpy.")

    ' To disconnect from the events delete the reference to the object.
    set objView = Nothing
Else
    Call WScript.Echo("The file " & docPath & ", does not exist.")
End If

' shut down XMLSpy when this script ends
objSpy.Visible = False
C#

The C# programming language can be used to access the Application API functionality. You could use Visual Studio 2010/2012/2013/2015/2017 to create the C# code, saving it in a Visual Studio project. Create the project as follows:

1. In Microsoft Visual Studio, add a new project using File | New | Project.
2. Add a reference to the XMLSpy Type Library by clicking Project | Add Reference. The Add Reference dialog appears. Browse for the XMLSpy Type Library component, which is located in the XMLSpy application folder, and add it.
3. Enter the code you want.
4. Compile the code and run it.

Example C# project

Your XMLSpy package contains an example C# project, which is located in the C# folder of the API Examples folder:

<table>
<thead>
<tr>
<th>Windows 7, Windows 8,</th>
<th>C:\Users&lt;username&gt;\Documents\Altova\XMLSpy2019\</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 10</td>
<td></td>
</tr>
</tbody>
</table>

You can compile and run the project from within Visual Studio 2010/2012/2013/2015/2017.

The code listing below shows how basic application functionality can be used. This code is similar to the example C# project in the API Examples folder of your application package, but might differ slightly.

Platform configuration

If you have a 64-bit operating system and are using a 32-bit installation of XMLSpy, you must 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…>).

What the code listing below does

The example code listing below creates a simple user interface (screenshot below) with buttons that invoke basic XMLSpy operations:
Overview

Application API

Start XMLSpy: Starts XMLSpy, which is registered as an automation server, or activates the application if it is already running.

Open OrgChart.pxf: Locates one of the example documents installed with XMLSpy and opens it. If this document is already open it becomes the active document.

OnDocumentOpened Event On/Off: Shows how to listen to XMLSpy events. When turned on, a message box will pop up after a document has been opened.

Open ExpReport.xml: Opens another example document.

Toggle View Mode: Changes the view of all open documents between Text View and Authentic View. The code shows how to iterate through open documents.

Validate: Validates the active document and shows the result in a message box. The code shows how to handle errors and COM output parameters.

Shut down XMLSpy: Stops XMLSpy.

You can modify the code (of the code listing below or of the example C# project in the API Examples folder) in any way you like and run it.

Compiling and running the example

In the API Examples folder, double-click the file AutomateXMLSpy_VS2008.sln or the file AutomateXMLSpy_VS2010.sln (to open in Visual Studio 2010/2012/2013/2015/2017). Alternatively the file can be opened from within Visual Studio (with File | Open | Project/Solution). To compile and run the example, select Debug | Start Debugging or Debug | Start Without Debugging.

Code listing of the example

Given below is the C# code listing of the basic functionality of the form (Form1.cs) created in the AutomateXMLSpy example. Note that the code listed below might differ slightly from the code in the API Examples form. The listing below is commented for ease of understanding. Parts of the code are also presented separately in the sub-sections of this section, according to the Application API functionality they access.
The code essentially consists of a series of handlers for the buttons in the user interface shown in the screenshot above.

```csharp
namespace WindowsFormsApplication2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // An instance of XMLSpy accessed via its automation interface
        XMLSpyLib.Application XMLSpy;

        // Location of examples installed with XMLSpy
        String strExamplesFolder;

        private void Form1_Load(object sender, EventArgs e)
        {
            // Locate examples installed with XMLSpy
            // REMARK: You might need to adapt this if you have a different major version of the product
            strExamplesFolder = Environment.GetEnvironmentVariable("USERPROFILE") + "\My Documents\Altova\XMLSpy2012\Examples\";
        }

        // Handler for the "Start XMLSpy" button
        private void StartXMLSpy_Click(object sender, EventArgs e)
        {
            if (XMLSpy == null)
            {
                Cursor.Current = Cursors.WaitCursor;

                // If no XMLSpy instance is open, create one and make it visible
                XMLSpy = new XMLSpyLib.Application();
                XMLSpy.Visible = true;
                Cursor.Current = Cursors.Default;
            }
            else
            {
                // If an instance of XMLSpy is already running, make sure it's visible
                if (!XMLSpy.Visible)
                {
                    XMLSpy.Visible = true;
                }
            }
        }

        // Handler for the "Open OrgChart.pxf" button
        private void openOrgChart_Click(object sender, EventArgs e)
        {
            // Make sure there's a running XMLSpy instance, and that it's visible
            StartXMLSpy_Click(null, null);
        }
    }
}
```
// Open one of the example files installed with the product
XMLSpy.Documents.OpenFile(strExamplesFolder + "OrgChart.pxf",
false);

// Handler for the "Open ExpReport.xml" button
private void openExpReport_Click(object sender, EventArgs e)
{
    // Make sure there's a running XMLSpy instance, and that it's
    // open one of the sample files installed with the product.
false);
}

// Handler for the "Toggle View Mode" button
private void toggleView_Click(object sender, EventArgs e)
{
    // Make sure there's a running XMLSpy instance, and that it's
    StartXMLSpy_Click(null, null);

    // Iterate through all open documents and toggle view between Text
    View and Authentic View
    foreach (XMLSpyLib.Document doc in XMLSpy.Documents)
    {
        if (doc.CurrentViewMode == XMLSpyLib.SPYViewModes.spyViewText)
            doc.SwitchViewMode(XMLSpyLib.SPYViewModes.spyViewAuthentic);
        else
            doc.SwitchViewMode(XMLSpyLib.SPYViewModes.spyViewText);
    }

    // Handler for the "Shutdown XMLSpy" button
    // Shut down the application instance by explicitly releasing the COM
    object
    private void shutdownXMLSpy_Click(object sender, EventArgs e)
    {
        if (XMLSpy != null)
        {
            // Allow shutdown of XMLSpy by releasing the UI
            XMLSpy.Visible = false;

            // Explicitly release the COM object
            try
            {
                while (System.Runtime.InteropServices.Marshal.ReleaseComObject(XMLSpy) > 0) ;
            }
            finally
            {
                // Disallow subsequent access to this object
                XMLSpy = null;
            }
        }
    }

    // Handler for button "Validate"
    private void validate_Click(object sender, EventArgs e)
    {

// COM errors are returned to C# as exceptions. We use a try/catch block to handle them.
try
{
    // Method 'IsValid' is one of the few functions that uses output parameters
    // Use 'object' type for these parameters
    object strErrorText = "";
    object nErrorNumber = 0;
    object errorData = null;

    if (!XMLSpy.ActiveDocument.IsValid(ref strErrorText, ref nErrorNumber, ref errorData))
    {
        // The COM call succeeded but the document is not valid
        // A detailed description of the problem is returned in
        strErrorText, nErrorNumber and errorData
        listBoxMessages.Items.Add("Document " +
        XMLSpy.ActiveDocument.Name + " is not valid.");
        ListBoxMessages.Items.Add("\tErrorText : " + strErrorText);
        listBoxMessages.Items.Add("\tErrorNumber: " + nErrorNumber);
        listBoxMessages.Items.Add("\tElement    : " + (errorData !=
        null ? ((XMLSpyLib.XMLData)errorData).TextValue : "null");
    }
    else
    {
        // The COM call succeeded and the document is valid
        listBoxMessages.Items.Add("Document " +
        XMLSpy.ActiveDocument.Name + " is valid.");
    }
    catch (Exception ex)
    {
        // The COM call was not successful
        // Probably no application instance has been started or no
document is open.
        listBoxMessages.Items.Add("Error validating active document: " +
        ex.Message);
    }
}

// Event handler for OnDocumentOpened event
private void handleOnDocumentOpened(XMLSpyLib.Document i_ipDocument)
{
    MessageBox.Show("Document " + i_ipDocument.Name + " was opened!");
}

// Remember if the event handler is currently registered.
private bool bEventHandlerIsRegistered = false;

// Handler for button 'OnDocuemntOpened Event On/Off
private void toggleOnDocumentOpenedEvent_Click(object sender, EventArgs e)
{
    if (XMLSpy != null)
    {
        if (bEventHandlerIsRegistered)
            XMLSpy.OnDocumentOpened += new
Add Reference to XMLSpy API

Add the application's type library as a reference in a .NET project as follows: With the .NET project open, click Project | Add Reference. Then browse for the type library, which is called XMLSpy.tlb, and is located in the XMLSpy application folder.

Then declare a variable to access the XMLSpy API:

```csharp
// An instance of XMLSpy is accessed via its automation interface.
XMLSpyLib.Application XMLSpy;
```

Application Startup and Shutdown

In the code snippets below, the methods StartXMLSpy_Click and ShutdownXMLSpy_Click are those assigned to buttons in the AutomateXMLSpy example that, respectively, start up and shut down the application. This example is located in the C# folder of the API Examples folder (see the file Form1.cs):

<table>
<thead>
<tr>
<th>Windows 7, Windows 8, Windows 10</th>
<th>C:\Users&lt;username&gt;\Documents\Altova\XMLSpy2019\</th>
</tr>
</thead>
</table>

You can compile and run the project from within Visual Studio 2010/2012/2013/2015/2017.

Starting XMLSpy

The following code snippet from the AutomateXMLSpy example shows how to start up the application.

```csharp
// Handler for the "Start XMLSpy" button
private void StartXMLSpy_Click(object sender, EventArgs e) {
    if (XMLSpy == null) {
        Cursor.Current = Cursors.WaitCursor;

        if (false) {
            XMLSpy = new XMLSpyLib.Application();
            XMLSpy.Visible = true;
        } else 
            XMLSpy = new XMLSpyLib.Application();
    }
}
```
Shutting down XMLSpy

The following code snippet from the AutomateXMLSpy example shows how to shut down the application.

```csharp
// Handler for the "Shutdown XMLSpy" button
// Shut down the application instance by explicitly releasing the COM object
private void shutdownXMLSpy_Click(object sender, EventArgs e)
{
    if (XMLSpy != null)
    {
        // Allow shutdown of XMLSpy by releasing the UI
        XMLSpy.Visible = false;

        // Explicitly release COM object
        try
        {
            while (System.Runtime.InteropServices.Marshal.ReleaseComObject(XMLSpy) > 0) ;
        }
        finally
        {
            // Disallow subsequent access to this object
            XMLSpy = null;
        }
    }
}
```

Opening Documents

The code snippets below (from the AutomateXMLSpy example) show how two files are opened via two separate methods assigned to two buttons in the user interface. Both methods use the same Application API access mechanism: XMLSpy.Documents.OpenFile(string, boolean).

The AutomateXMLSpy example (see the file Form1.cs) is located in the C# folder of the API Examples folder:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 7, Windows 8, Windows 10</td>
<td>C:\Users&lt;username&gt;\Documents\Altova\XMLSpy2019\</td>
</tr>
</tbody>
</table>

You can compile and run the project from within Visual Studio 2010/2012/2013/2015/2017.
Code snippet

// Handler for the "Open OrgChart.pxf" button
private void openOrgChart_Click(object sender, EventArgs e)
{
    // Make sure there's a running XMLSpy instance, and that it's visible
    StartXMLSpy_Click(null, null);

    // Open a file from the Examples folder installed with the product
    XMLSpy/Documents.OpenFile(strExamplesFolder + "OrgChart.pxf",
    false);
}

// Handler for the "Open ExpReport.xml" button
private void openExpReport_Click(object sender, EventArgs e)
{
    // Make sure there's a running XMLSpy instance, and that it's visible
    StartXMLSpy_Click(null, null);

    // Open a file from the Examples folder installed with the product
    false);
}

The file opened last will be the active file.

Iterating through Open Documents

The code snippet below (from the AutomateXMLSpy example; see the file Form1.cs) shows how to iterate through open documents. A condition is then tested within the iteration loop, and the document view is switched between Text View and Authentic View.

// Handler for the "Toggle view mode" button
private void toggleView_Click(object sender, EventArgs e)
{
    // Make sure there's a running XMLSpy instance, and that it's visible
    StartXMLSpy_Click(null, null);

    // Iterate through open documents and toggle current view between text and authentic view.
    foreach (XMLSpyLib.Document doc in XMLSpy/Documents)
    {
        if (doc.CurrentViewMode == XMLSpyLib.SPYViewModes.spyViewText)
            doc.SwitchViewMode(XMLSpyLib.SPYViewModes.spyViewAuthentic);
        else
            doc.SwitchViewMode(XMLSpyLib.SPYViewModes.spyViewText);
    }
}

The AutomateXMLSpy example example is located in the C# folder of the API Examples folder.
Errors and COM Output Parameters

The code snippet below (from the AutomateXMLSpy example) shows how to handle errors and COM output parameters. The method XMLSpy.ActiveDocument.IsValid(ref strErrorText, ref nErrorNumber, ref errorData) uses output parameters that are used, in the code snippet below, to generate an error-message text.

The AutomateXMLSpy example (see the file Form1.cs) is located in the C# folder of the API Examples folder:

Code snippet

```csharp
// Handler for button "Validate"
private void validate_Click(object sender, EventArgs e)
{
    // COM errors are returned to C# as exceptions. We use a try/catch block to handle them.
    try
    {
        // Method 'IsValid' is one of the few functions that uses output parameters
        // Use 'object' type for these parameters
        object strErrorText = "";
        object nErrorNumber = 0;
        object errorData = null;

        if (!XMLSpy.ActiveDocument.IsValid(ref strErrorText, ref nErrorNumber, ref errorData))
        {
            // The COM call succeeded but the document is not valid
            // A detailed description of the problem is returned in strErrorText, nErrorNumber and errorData
            listBoxMessages.Items.Add("Document " + XMLSpy.ActiveDocument.Name + " is not valid.");
            listBoxMessages.Items.Add("\tErrorText : " + strErrorText);
            listBoxMessages.Items.Add("\tErrorNumber: " + nErrorNumber);
            listBoxMessages.Items.Add("\tElement    : " + (errorData != null ? ((XMLSpyLib.XMLData)errorData).TextValue : "null"));
        }
        else
        {
            // The COM call succeeded and the document is valid
        }
    }
    catch
    {
        // Handle exceptions
    }
}
```
listBoxMessages.Items.Add("Document " + XMLSpy.ActiveDocument.Name + " is valid.");
}
}
catch (Exception ex)
{
    // The COM call was not successful
    // Probably no application instance has been started or no
    // document is open.
    listBoxMessages.Items.Add("Error validating active document: " + ex.Message);
}
}

**Events**

The code snippet below (from the AutomateXMLSpy example) lists the code for two event
handlers. The AutomateXMLSpy example (see the file Form1.cs) is located in the C# folder of the
API Examples folder:

You can compile and run the project from within Visual Studio 2010/2012/2013/2015/2017.

```csharp
// Event handler for OnDocumentOpened event
private void handleOnDocumentOpened(XMLSpyLib.Document i_ipDocument)
{
    MessageBox.Show("Document " + i_ipDocument.Name + " was opened!");
}

// Remember if the event handler is currently registered.
private bool bEventHandlerIsRegistered = false;

// Handler for button 'OnDocumentOpened Event On/Off
private void toggleOnDocumentOpenedEvent_Click(object sender, EventArgs e)
{
    if (XMLSpy != null)
    {
        if (bEventHandlerIsRegistered)
            XMLSpy.OnDocumentOpened -= new
            XMLSpyLib._IApplicationEvents_OnDocumentOpenedEventHandler(handleOnDocumentOpened);
        else
            XMLSpy.OnDocumentOpened += new
            XMLSpyLib._IApplicationEvents_OnDocumentOpenedEventHandler(handleOnDocumentOpened);
        bEventHandlerIsRegistered = !bEventHandlerIsRegistered;
    }
}
```
Java

The Application API can be accessed from Java code. To allow accessing the XMLSpy automation server directly from Java code, the libraries listed below must reside in the classpath. They are installed in the folder: JavaAPI in the XMLSpy application folder.

- **AltovaAutomation.dll**: a JNI wrapper for Altova automation servers (AltovaAutomation_x64.dll in the case of 64-bit versions)
- **AltovaAutomation.jar**: Java classes to access Altova automation servers
- **XMLSpyAPI.jar**: Java classes that wrap the XMLSpy automation interface
- **XMLSpyAPI_JavaDoc.zip**: a Javadoc file containing help documentation for the Java API

**Note:** In order to use the Java API, the DLL and Jar files must be on the Java Classpath.

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 of the example Java project, see the section, [Example Java Project](#).

Rules for mapping the Application API names to Java

The rules for mapping between the Application API and the Java wrapper are as follows:

- **Classes and class names**
  For every interface of the XMLSpy 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. 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:

  Application: Java class to access the application
  ApplicationEvents: Events interface for the Application
  ApplicationEventsDefaultHandler: Default handler for ApplicationEvents

**Exceptions to mapping rules**

There are some exceptions to the rules listed above. These are listed below:
This section explains how some basic XMLSpy functionality can be accessed from Java code. It is organized into the following sub-sections:

- Example Java Project
- Application Startup and Shutdown
- Simple Document Access
- Iterations
- Use of Out-Parameters
- Event Handlers

Example Java Project

The XMLSpy installation package contains an example Java project, located in the Java folder of the API Examples folder:

| Windows 7, Windows 8, Windows 10 | C:Users<username>Documents\Altova\XMLSpy2019\ |

This folder contains Java examples for the XMLSpy API. 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. If you are using a 64-bit version of the application, some filenames contain _x64 in the name. These filenames are indicated with (_x64).

- AltovaAutomation(_x64).dll: Java-COM bridge: DLL part
- AltovaAutomation.jar: Java-COM bridge: Java library part
- XMLSpyAPI.jar: Java classes of the XMLSpy API
- RunXMLSpy.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
What the example does
The example starts up XMLSpy and performs a few operations, including opening and closing documents. When done, XMLSpy stays open. You must close it manually.

- **Start XML Spy**: Starts XMLSpy, which is registered as an automation server, or activates XMLSpy if it is already running.
- **Open OrgChart.pxf**: Locates one of the example documents installed with XMLSpy and opens it.
- **Iteration and Changing the View Mode**: Changes the view of all open documents to Text View. The code also shows how to iterate through open documents.
- **Iteration, validation, output parameters**: Validates the active document and shows the result in a message box. The code shows how to use output parameters.
- **Event Handling**: Shows how to handle XMLSpy events.
- **Shut down XMLSpy**: Shuts down XMLSpy.

You can modify the example in any way you like and run it.

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:

```bash
buildAndRun.bat "<Path-to-the-Java-bin-folder>"
```

The Java binary folder must be that of a JDK 1.5 or later installation on your computer.

Press the **Return** key. The Java source in `RunXMLSpy.java` will be compiled and then executed.

Loading the example in Eclipse
Open Eclipse and use the **Import | 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 `RunXMLSpy` 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 `RunXMLSpy.java` is listed below with comments.
// Access general JAVA-COM bridge classes
import com.altova.automation.libs.*;

// Access XMLSpy Java-COM bridge
import com.altova.automation.XMLSpy.*;
import com.altova.automation.XMLSpy.Enums.SPYViewModes;

/**
 * An example that starts XMLSpy COM server and performs view operations on it
 * Feel free to extend
 */
public class RunXMLSpy {
  public static void main(String[] args) {
    // An instance of the application.
    Application xmlSpy = null;

    // Instead of COM error handling, use Java exception mechanism
    try {
      // Start XMLSpy as COM server
      xmlSpy = new Application();

      // COM servers start up invisible, so make it visible
      xmlSpy.setVisible(true);

      // Locate samples installed with the product
      String strExamplesFolder = System.getenv("USERPROFILE") + "\My Documents\Altova\XMLSpy2012\Experiments";

      // Open two example files
      xmlSpy.getDocuments().openFile(strExamplesFolder + "OrgChart.pxf", false);
      xmlSpy.getDocuments().openFile(strExamplesFolder + "ExpReport.xml", false);

      // Iterate through open documents and set view mode to 'Text'.
      for (Document doc:xmlSpy.getDocuments())
        if (doc.getCurrentViewMode() != SPYViewModes.spyViewText)
          doc.switchViewMode(SPYViewModes.spyViewText);

    } catch (Exception e) {
      // An alternative iteration mode is index-based
      // COM indices are typically zero-based
      Documents documents = xmlSpy.getDocuments();
      for (int i = 1; i <= documents.getCount(); i++)
        {;
          Document doc = documents.getItem(i);

          // Validation is one of the few methods to have output parameters.
          // The class JVariant is the correct type for parameters in these cases.
          // To get values back mark them with the by-reference flag.
          JVariant validationErrorText = new JVariant.JStringVariant("");
055 validationErrorText.setByRefFlag();
056 JVariant validationErrorCount = new
057 JVariant.JIntVariant(0);
058 validationErrorCount.setByRefFlag();
059 JVariant validationErrorXMLData = new
060 JVariant.JIDispatchVariant(0);
061 validationErrorXMLData.setByRefFlag();
062 if (!doc.isValid(validationErrorText, validationErrorCount,
063 validationErrorXMLData))
064 System.out.println("Document" + doc.getName() + " is not
065 wellformed = " + validationErrorText.getStringValue());
066 else
067 System.out.println("Document" + doc.getName() + " is
068 wellformed.");
069 }
070 }
071 // The following lines attach to the document events using a default
072 implementation
073 // for the events and override one of its methods.
074 // If you want to override all document events it is better to derive
075 your listener class
076 // from DocumentEvents and implement all methods of this interface.
077 Document doc = xmlSpy.getActiveDocument();
078 doc.addListener(new
079 DocumentEventsDefaultHandler()
080 {
081 @Override
082 public boolean
083 onBeforeCloseDocument(Document i_ipDoc) throws AutomationException
084 {
085 System.out.println("Document
086 " + i_ipDoc.getName() + " requested closing.");
087 // Allow closing of document
088 return true;
089 }
090 doc.close(true);
091 doc = null;
092 System.out.println("Watch XMLSpy!");
093 }
094 });
095 catch (AutomationException e)
096 {
097 // e.printStackTrace();
098 }
099 finally
100 {
101 // Make sure that XMLSpy can shut down properly.
102 if (xmlSpy != null)
103 xmlSpy.dispose();
104 }
105 }
Application Startup and Shutdown

The code listings below show how the application can be started up and shut down.

Application startup
Before starting up the application, the appropriate classes must be imported (see below).

```java
import com.altova.automation.libs.*;
import com.altova.automation.XMLSpy.*;
import com.altova.automation.XMLSpy.Enums.SPYViewModes;

public class RunXMLSpy {
    public static void main(String[] args) {
        // An instance of the application.
        Application xmlSpy = null;
        try {
            // Start XMLSpy as COM server
            xmlSpy = new Application();
            // COM servers start up invisible, so make it visible
            xmlSpy.setVisible(true);
        } catch (
```
// Make sure that XMLSpy can shut down properly.
if (xmlSpy != null)
    xmlSpy.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 XMLSpy!");

---

Simple Document Access

The code listing below shows how to open a document.

// Locate samples installed with the product
String strExamplesFolder = System.getenv("USERPROFILE") + "\My Documents\Altova\XMLSpy2012\Examples\";

// Open file
xmlSpy.getDocuments().openFile(strExamplesFolder + "OrgChart.pxf", false);

---

Iterations

The listing below shows how to iterate through open documents.

// Iterate through open documents and set view mode to 'Text'.
for (Document doc:xmlSpy.getDocuments())
    if ( doc.getCurrentViewMode() != SPYViewModes.spyViewText)
        doc.switchViewMode(SPYViewModes.spyViewText);

// An alternative iteration mode is index-based
// COM indices are typically zero-based
Documents documents = xmlSpy.getDocuments();
for (int i = 1; i <= documents.getCount(); i++)
    { Document doc = documents.getItem(i);
      ...
    }

---

Use of Out-Parameters

The code listing below iterates through open documents and validates each of them. For each validation, a message is generated using the output parameters of the Validation method.

// An alternative iteration mode is index-based
// COM indices are typically zero-based
Documents documents = xmlSpy.getDocuments();
for (int i = 1; i <= documents.getCount(); i++)
05      i++)
06     {
07         Document doc = documents.getItem(i);
08
09     // Validation is one of the few methods to have output parameters.
10     // The class JVariant is the correct type for parameters in these cases.
11     // To get values back mark them with the by-reference flag.
12     JVariant validationErrorText = new
13     JVariant.JStringVariant("");
14     validationErrorText.setByRefFlag();
15     JVariant validationErrorCount = new
16     JVariant.JIntVariant(0);
17     validationErrorCount.setByRefFlag();
18     JVariant validationErrorXMLData = new
19     JVariant.JIDispatchVariant(0);
20     validationErrorXMLData.setByRefFlag();
21     if (!doc.isValid(validationErrorText,
22         validationErrorCount, validationErrorXMLData))
23         System.out.println("Document
24         " + doc.getName() + " is not wellformed - " +
25         validationErrorText.getStringValue());
26     else
27         System.out.println("Document
28         " + doc.getName() + " is wellformed.");
29     }

Event Handlers

The listing below shows how to listen for and use events.

01     // The following lines attach to the document events using a default
02     // implementation
03     // for the events and override one of its methods.
04     // If you want to override all document events it is better to derive your
05     // listener class
06     // from DocumentEvents and implement all methods of this interface.
07     Document doc = xmlSpy.getActiveDocument();
08     doc.addListener(new DocumentEventsDefaultHandler()
09     {
10         @Override
11         public boolean
12             onBeforeCloseDocument(Document i_ipDoc) throws AutomationException
13             {
14                 System.out.println("Document " + i_ipDoc.getName() + " requested
15                     closing.");
16             }

© 2019 Altova GmbH
3.1.3 The DOM and XMLData

The XMLData interface gives you full access to the XML structure behind the current document with less methods than DOM and is much simpler. The XMLData interface is a minimalist approach to reading and modifying existing, or newly created XML data. You might however, want to use a DOM tree because you can access one from an external source or you just prefer the MSXML DOM implementation.

The `ProcessDOMNode()` and `ProcessXMLDataNode()` functions provided below convert any segments of an XML structure between XMLData and DOM.

To use the `ProcessDOMNode()` function:
- pass the root element of the DOM segment you want to convert in `objNode` and
- pass the plugin object with the `CreateChild()` method in `objCreator`

To use the `ProcessXMLDataNode()` function:
- pass the root element of the XMLData segment in `objXMLData` and
- pass the DOMDocument object created with MSXML in `xmlDoc`

```
// DOM To XMLData conversion
Function ProcessDOMNode(objNode, objCreator)
{
    var objRoot;
    objRoot = CreateXMLDataFromDOMNode(objNode, objCreator);
    If(objRoot) {
        If((objNode.nodeValue != null) && (objNode.nodeValue.length > 0))
            objRoot.TextValue = objNode.nodeValue;
        // add attributes
        If(objNode.attributes) {
            var Attribute;
            var oNodeList = objNode.attributes;
            For(var i = 0; i < oNodeList.length; i++) {
                Attribute = oNodeList.item(i);
                var newNode;
                newNode = ProcessDOMNode(Attribute, objCreator);
                objRoot.AppendChild(newNode);
            }
        }
        If(objNode.hasChildNodes) {
            try {
                // add children
```
```javascript
var Item;
oNodeList = objNode.childNodes;

For(var i = 0; i < oNodeList.length; i++) {
    Item = oNodeList.item(i);
    var newNode;
    newNode = ProcessDOMNode(Item, objCreator);
    objRoot.appendChild(newNode);
}
}

function CreateXMLDataFromDOMNode(objNode, objCreator)
{
    var bSetName = true;
    var bSetValue = true;
    var nKind = 4;

    switch(objNode.nodeType) {
    Case 2:nKind = 5;break;
    Case 3:nKind = 6;bSetName = false;break;
    Case 4:nKind = 7;bSetName = false;break;
    Case 8:nKind = 8;bSetName = false;break;
    Case 7:nKind = 9;break;
    }
    var objNew = null;
    objNew = objCreator.CreateChild(nKind);

    If(bSetName)
        objNew.Name = objNode.nodeName;
    If(bSetValue (objNode.nodeValue != null))
        objNew.TextValue = objNode.nodeValue;

    Return objNew;
}
```
objRoot.appendChild(xmlDoc.createTextNode(objXMLData.TextValue));

If (objXMLData.HasChildren) {
    try {
        var objChild;
        objChild = objXMLData.GetFirstChild(-1);

        While (True) {
            If (objChild) {
                var newNode;
                newNode = ProcessXMLDataNode(objChild, xmlDoc);

                If (newNode.nodeType == 2) {
                    // child node is an attribute
                    objRoot.attributes.setNamedItem(newNode);
                }
                Else
                    objRoot.appendChild(newNode);
            }
            objChild = objXMLData.GetNextChild();
        }
    }
    catch (err) {
    }
}
Return objRoot;

Function CreateDOMNodeFromXMLData (objXMLData, xmlDoc) {
    switch (objXMLData.Kind) {
        Case 4: Return xmlDoc.createElement(objXMLData.Name);
        Case 5: Return xmlDoc.createAttribute(objXMLData.Name);
        Case 6: Return xmlDoc.createTextNode(objXMLData.TextValue);
        Case 7: Return xmlDoc.createCDATASection(objXMLData.TextValue);
        Case 8: Return xmlDoc.createComment(objXMLData.TextValue);
        Case 9: Return xmlDoc.createProcessingInstruction(objXMLData.Name, objXMLData.TextValue);
    }
    Return xmlDoc.createElement(objXMLData.Name);
}

Function IsTextNodeEnabled (objNode) {
    switch (objNode.nodeType) {
        Case 1:
        Case 2:
        Case 5:
        Case 6:
        Case 11: Return True;
    }
    Return False;
3.1.4 Obsolete: Authentic View Row operations

If the schema on which an XML document is based specifies that an element is repeatable, such a structure can be represented in Authentic View as a table. When represented as a table, rows and their contents can be manipulated individually, thereby allowing you to manipulate each of the repeatable elements individually. Such row operations would be performed by an external script.

If an external script is to perform row operations then two steps must occur:

- The first step checks whether the cursor is currently in a row using a property. Such a check could be, for example, `IsRowInsertEnabled`, which returns a value of either `TRUE` or `FALSE`.
- If the return value is `TRUE` then a row method, such as `RowAppend`, can be called. (`RowAppend` has no parameters and returns no value.)

The following is a list of properties and methods available for table operations. Each property returns a `BOOL`, and the methods have no parameter.

<table>
<thead>
<tr>
<th>Property</th>
<th>Method</th>
<th>Table operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsRowInsertEnabled</td>
<td><code>RowInsert</code></td>
<td>Insert row operation</td>
</tr>
<tr>
<td></td>
<td><code>AuthenticRange.InsertRow</code></td>
<td></td>
</tr>
<tr>
<td>IsRowAppendEnabled</td>
<td><code>RowAppend</code></td>
<td>Append row operation</td>
</tr>
<tr>
<td></td>
<td><code>AuthenticRange.AppendRow</code></td>
<td></td>
</tr>
<tr>
<td>IsRowDeleteEnabled</td>
<td><code>RowDelete</code></td>
<td>Delete row operation</td>
</tr>
<tr>
<td></td>
<td><code>AuthenticRange.DeleteRow</code></td>
<td></td>
</tr>
<tr>
<td>IsRowMoveUpEnabled</td>
<td><code>RowMoveUp</code></td>
<td>Move XML data up one row</td>
</tr>
<tr>
<td></td>
<td><code>AuthenticRange.MoveRowUp</code></td>
<td></td>
</tr>
<tr>
<td>IsRowMoveDownEnabled</td>
<td><code>RowMoveDown</code></td>
<td>Move XML data down one row</td>
</tr>
<tr>
<td></td>
<td><code>AuthenticRange.MoveRowDown</code></td>
<td></td>
</tr>
<tr>
<td>IsRowDuplicateEnabled</td>
<td><code>RowDuplicate</code></td>
<td>Duplicate currently selected row</td>
</tr>
<tr>
<td></td>
<td><code>AuthenticRange.DuplicateRow</code></td>
<td></td>
</tr>
</tbody>
</table>

3.1.5 Obsolete: Authentic View Editing operations

When XML data is displayed as data in Authentic View, it is possible to manipulate individual elements using standard editing operations such as cut, copy, and paste. However, not all XML data nodes can be edited. So, in order to carry out an editing operation, first a property is used to test whether editing is possible, and then a method is called to perform the editing operation.

The only method that does not have a test is the method `EditSelectAll`, which automatically selects all elements displayed in the document.

The following is a list of properties and methods that perform editing operations. Each property returns a `BOOL`, and the methods have no parameter.
<table>
<thead>
<tr>
<th>Property</th>
<th>Method</th>
<th>Editing operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsEditUndoEnabled</td>
<td><code>EditUndo</code></td>
<td>Undo an editing operation</td>
</tr>
<tr>
<td></td>
<td><code>AuthenticView.Undo</code></td>
<td></td>
</tr>
<tr>
<td>IsEditRedoEnabled</td>
<td><code>EditRedo</code></td>
<td>Redo an editing operation</td>
</tr>
<tr>
<td></td>
<td><code>AuthenticView.Redo</code></td>
<td></td>
</tr>
<tr>
<td>IsEditCopyEnabled</td>
<td><code>EditCopy</code></td>
<td>Copy selected text to the clipboard</td>
</tr>
<tr>
<td></td>
<td><code>AuthenticRange.Copy</code></td>
<td></td>
</tr>
<tr>
<td>IsEditCutEnabled</td>
<td><code>EditCut</code></td>
<td>Cut selected text to the clipboard</td>
</tr>
<tr>
<td></td>
<td><code>AuthenticRange.Cut</code></td>
<td></td>
</tr>
<tr>
<td>IsEditPasteEnabled</td>
<td><code>EditPaste</code></td>
<td>Paste from clipboard to current cursor position</td>
</tr>
<tr>
<td></td>
<td><code>AuthenticRange.Paste</code></td>
<td></td>
</tr>
<tr>
<td>IsEditClearEnabled</td>
<td><code>EditClear</code></td>
<td>Clear selected text from XML document</td>
</tr>
<tr>
<td></td>
<td><code>AuthenticRange.Delete</code></td>
<td></td>
</tr>
</tbody>
</table>
3.2 Interfaces

Object Hierarchy

Application
  SpyProject
    SpyProjectItems
      SpyProjectItem
  Documents
    Document
      GridView
      AuthenticView
        AuthenticDataTransfer (previously DocEditDataTransfer)
      OldAuthenticView (previously DocEditView, now obsolete, superseded by
      AuthenticView and AuthenticRange)
      AuthenticSelection (previously DocEditSelection, now obsolete,
      superseded by AuthenticRange)
      AuthenticEvent (previously DocEditEvent, now obsolete)
    AuthenticDataTransfer (previously DocEditDataTransfer)
  TextView
  XMLData
Dialogs
  CodeGeneratorDlg
  FileSelectionDlg
  SchemaDocumentationDlg
  GenerateSampleXMLDlg
  DTDSchemaGeneratorDlg
  FindInFilesDlg
DatabaseConnection
ExportSettings
TextImportExportSettings
ElementList
  ElementListItem

Enumerations

Description
This chapter contains the reference of the XMLSpy 1.5 Type Library.

Most of the given examples are written in VisualBasic. These code snippets assume that there is
a variable defined and set, called **objSpy of type Application**. There are also some code
samples written in JavaScript.

3.2.1 Application

See also

Methods
GetDatabaseImportElementList
GetDatabaseSettings
GetDatabaseTables
ImportFromDatabase
CreateXMLSchemaFromDBStructure

GetTextImportElementList
GetTextImportExportSettings
ImportFromText

ImportFromWord

ImportFromSchema

GetExportSettings

NewProject
OpenProject

AddMacroMenuItem
ClearMacroMenu

ShowForm

ShowApplication

URLDelete
URLMakeDirectory

AddXSLT_XQParameter
GetXSLT_XQParameterCount
GetXSLT_XQParameterName
GetXSLT_XQParameterXPath
RemoveXSLT_XQParameter

FindInFiles

Quit

Properties
Application
Parent

ActiveDocument
Documents

CurrentProject

Dialogs

WarningNumber
WarningText

Status
MajorVersion
MinorVersion
Edition
Application is the root for all other objects. It is the only object you can create by `CreateObject` (VisualBasic) or other similar COM related functions.

Example

```vbscript
Dim objSpy As Application
Set objSpy = CreateObject("XMLSpy.Application")
```

Events

**OnBeforeOpenDocument**

See also

**Event:** `OnBeforeOpenDocument(objDialog as FileSelectionDlg)`

Description

This event gets fired whenever a document gets opened via the OpenFile or OpenURL menu command. It is sent after a document file has been selected but before the document gets opened. The file selection dialog object is initialized with the name of the selected document file. You can modify this selection. To continue the opening of the document leave the `FileSelectionDlg.DialogAction` property of `io_objDialog` at its default value `spyDialogOK`. To abort the opening of the document set this property to `spyDialogCancel`.

Examples

Given below are examples of how this event can be scripted.

**XMLSpy scripting environment - VBScript:**

```vbscript
Function On_BeforeOpenDocument(objDialog)
End Function
```

**XMLSpy scripting environment - JScript:**

```javascript
function On_BeforeOpenDocument(objDialog)
{
}
```

**XMLSpy IDE Plugin:**

```csharp
IXMLSpyPlugIn.OnEvent(26, ...) // nEventId = 26
```

**OnBeforeOpenProject**

See also

**Event:** `OnBeforeOpenProject(objDialog as FileSelectionDlg)`

Description
This event gets fired after a project file has been selected but before the project gets opened. The file selection dialog object is initialized with the name of the selected project file. You can modify this selection. To continue the opening of the project leave the `FileSelectionDlg.DialogResult` property of `io_objDialog` at its default value `spyDialogOK`. To abort the opening of the project set this property to `spyDialogCancel`.

**Examples**

Given below are examples of how this event can be scripted.

**XMLSpy scripting environment - VBScript:**

```vbscript
Function On_BeforeOpenProject(objDialog)
End Function
```

**XMLSpy scripting environment - JScript:**

```javascript
function On_BeforeOpenProject(objDialog)
{
}
```

**XMLSpy IDE Plugin:**

```csharp
IXMLSpyPlugin.OnEvent(25, ...) // nEventId = 25
```

---

**OnDocumentOpened**

**See also**

**Event:** `OnDocumentOpened(objDocument as Document)`

**Description**

This event gets fired whenever a document opens in XMLSpy. This can happen due to opening a file with the OpenFile or OpenURL dialog, creating a new file or dropping a file onto XMLSpy. The new document gets passed as parameter. The operation cannot be canceled.

**Examples**

Given below are examples of how this event can be scripted.

**XMLSpy scripting environment - VBScript:**

```vbscript
Function On_OpenDocument(objDocument)
End Function
```

**XMLSpy scripting environment - JScript:**

```javascript
function On_OpenDocument(objDocument)
{
}
```

**XMLSpy IDE Plugin:**

```csharp
IXMLSpyPlugin.OnEvent(7, ...) // nEventId = 7
```
**OnProjectOpened**

See also

*Event:* `OnProjectOpened(objProject as SpyProject)`

**Description**

This event gets fired whenever a project gets opened in XMLSpy. The new project gets passed as parameter.

**Examples**

Given below are examples of how this event can be scripted.

*XMLSpy scripting environment - VBScript:*

```vbnet
Function On_OpenProject(objProject)
End Function
```

*XMLSpy scripting environment - JScript:*

```javascript
function On_OpenProject(objProject)
{
}
```

*XMLSpy IDE Plugin:*

```csharp
IXMLSpyPlugIn.OnEvent(6, ...) // nEventId = 6
```

**ActiveDocument**

See also

*Property:* `ActiveDocument as Document`

**Description**

Reference to the active document. If no document is open, `ActiveDocument` is null (nothing).

**Errors**

1111  The application object is no longer valid.
1100  Invalid address for the return parameter was specified.

**AddMacroMenuItem**

See also

*Method:* `AddMacroMenuItem(strMacro as String, strDisplayText as String)`

**Description**

adds a menu item to the *Tools* menu. This new menu item invokes the macro defined by *strMacro*. See also "Calling macros from XMLSpy".

**Errors**

1111  The application object is no longer valid.
AddXSLT_XQParameter

**Method:** AddXSLT_XQParameter(name as String, XPath as String)

**Description**
Adds an XSLT or XQuery parameter. The parameter's name and value are the two arguments of the method.

**Errors**
- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.
- 1124 The XPath expression is not set.
- 1125 Not a QName.
- 1126 The specified XPath is not valid. Reason for invalidity appended.
- 1127 A parameter with the submitted name already exists.

Application

**See also**

**Property:** Application as Application (read-only)

**Description**
Accesses the XMLSpy application object.

**Errors**
- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

ClearMacroMenu

**See also**

**Method:** ClearMacroMenu()

**Return Value**
None

**Description**
Removes all menu items from the Tools menu. See also Running macros.

**Errors**
- 1111 The application object is no longer valid.

CreateXMLSchemaFromDBStructure

**See also**
Method: `CreateXMLSchemaFromDBStructure(pImportSettings as DatabaseConnection, pTables as ElementList)`

Description
`CreateXMLSchemaFromDBStructure` creates from a database specified in `pImportSettings` for the defined tables in `pTables` new XML Schema document(s) describing the database tables structure.

The parameter `pTables` specifies which table structures the XML Schema document should contain. This parameter can be NULL, specifying that all table structures will be exported.

See also [GetDatabaseTables](#).

Errors
- 1112 Invalid database specified.
- 1120 Database import failed.

CurrentProject
See also

Property: `CurrentProject as SpyProject`

Description
Reference to the active document. If no project is open, `CurrentProject` is null (nothing).

Errors
- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

Dialogs
See also

Property: `Dialogs as Dialogs` (read-only)

Description
Access the built-in dialogs of XMLSpy.

Errors
- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

Documents
See also

Property: `Documents as Documents`

Description
Collection of all open documents.
Errors
1111 The application object is no longer valid.
1100 Invalid address for the return parameter was specified.

Edition
See also

Property: Edition as String

Description
Returns the edition of the application, for example Altova XMLSpy Enterprise Edition for the Enterprise edition.

Errors
1111 The application object is no longer valid.
1100 Invalid address for the return parameter was specified.

FindInFiles
See also

Method: FindInFiles(pSettings as FindInFilesDlg) as FindInFilesResults

Description
Returns a FindInFilesResults object containing information about the files that matched the specified settings.

Errors
1111 The application object is no longer valid.
1100 Invalid address for the return parameter was specified.

GetDatabaseImportElementList
See also

Method: GetDatabaseImportElementList(pImportSettings as DatabaseConnection) as ElementList

Description
The function returns a collection of ElementListItems where the properties ElementList.Item.Name contain the names of the fields that can be selected for import and the properties ElementList.Item.ElementKind are initialized either to spyXMLDataAttr or spyXMLDataElement, depending on the value passed in DatabaseConnection.AsAttributes. This list serves as a filter to what finally gets imported by a future call to ImportFromDatabase. Use ElementList.RemoveElement to exclude fields from import.

Properties mandatory to be filled out for the database connection are one of DatabaseConnection.File, DatabaseConnection.ADOConnection and DatabaseConnection.ODBCConnection, as well as DatabaseConnection.SQLSelect. Use the property DatabaseConnection.AsAttributes to initialize
ElementListItem.ElementKind of the resulting element list to either spyXMLDataAttr or spyXMLDataElement, respectively.

Example
See example at ImportFromDatabase.

Errors
1111  The application object is no longer valid.
1100  Invalid parameter or invalid address for the return parameter was specified.
1107  Import from database failed.
1112  Invalid database specified.
1114  Select statement is missing.
1119  database element list import failed.

GetDatabaseSettings
See also

Method: GetDatabaseSettings() as DatabaseConnection

Description
GetDatabaseSettings creates a new object of database settings. The object is used to specify database connection parameters for the methods GetDatabaseTables, GetDatabaseImportElementList, ImportFromDatabase, ImportFromSchema and ExportToDatabase.

Example
See example of ImportFromDatabase.

Errors
1111  The application object is no longer valid.
1100  Invalid address for the return parameter was specified.

GetDatabaseTables
See also

Method: GetDatabaseTables(pImportSettings as DatabaseConnection) as ElementList

Description
GetDatabaseTables reads the table names from the database specified in pImportSettings. Properties mandatory to be filled out for the database connection are one of DatabaseConnection.File, DatabaseConnection.ADOConnection and DatabaseConnection.ODBCConnection. All other properties are ignored.
The function returns a collection of ElementListItems where the properties ElementListItem.Name contain the names of tables stored in the specified database. The remaining properties of ElementListItem are unused.

Errors
1111  The application object is no longer valid.
1100  Invalid parameter or invalid address for the return parameter was specified.
1112  Invalid database specified.
Error while reading database table information.
Database table query failed.

Example

```vba
Dim objImpSettings As DatabaseConnection
Set objImpSettings = objSpy.GetDatabaseSettings
objImpSettings.ADOConnection = TxtADO.Text

'store table names in list box
ListTables.Clear

Dim objList As ElementList
Dim objItem As ElementListItem
On Error GoTo ErrorHandler
Set objList = objSpy.GetDatabaseTables(objImpSettings)

For Each objItem In objList
    ListTables.AddItem objItem.Name
Next
```

GetExportSettings

See also

**Method:** GetExportSettings() as [ExportSettings](#) (read-only)

**Description**
GetExportSettings creates a new object of common export settings. This object is used to pass the parameters to the export functions and defines the behaviour of the export calls. See also the export functions from [Document](#) and the examples at [Import and Export](#).

Errors

1111 The application object is no longer valid.
1100 Invalid address for the return parameter was specified.

GetTextImportElementList

See also

**Method:** GetTextImportElementList(pImportSettings as [TextImportExportSettings](#)) as [ElementList](#)

**Description**
GetTextImportElementList retrieves importing information about the text-file as specified in pImportSettings. The function returns a collection of ElementListItem where the properties ElementListItem.Name contain the names of the fields found in the file. The values of remaining properties are undefined.

If the text-file does not contain a column header, set pImportSettings.HeaderRow to False. The resulting element list will contain general column names like 'Field1' and so on.

See also [Import and export of data](#).
Errors

1111 The application object is no longer valid.
1100 Invalid parameter or invalid address for the return parameter was specified.
1107 Import from database failed.
1115 Error during text element list import. Cannot create parser for import file.
1116 Error during text element list import.

Example

```vba
' ---------------------------------------------------------
' VBA client code fragment - import selected fields from text file
' ---------------------------------------------------------
Dim objImpSettings As TextImportExportSettings
Set objImpSettings = objSpy.GetTextImportExportSettings

objImpSettings.ImportFile = "C:\ImportMe.txt"
objImpSettings.HeaderRow = False

Dim objList As ElementList
Set objList = objSpy.GetTextImportElementList(objImpSettings)

' exclude first column
objList.RemoveItem 1

Dim objImpDoc As Document
On Error Resume Next
Set objImpDoc = objSpy.ImportFromText(objImpSettings, objList)
CheckForError
```

**GetTextImportExportSettings**

*See also*

Method: `GetTextImportExportSettings()` as `TextImportExportSettings` (read-only)

Description

`GetTextImportExportSettings` creates a new object of common import and export settings for text files. See also the example for `Application.GetTextImportElementList` and `Import and Export`.

Errors

1111 The application object is no longer valid.
1100 Invalid address for the return parameter was specified.

**GetXSLT_XQParameterCount**

Method: `GetXSLT_XQParameterCount()` as `Long`

Description

Returns the number of XSLT and XQuery parameters.

Errors
1111  The application object is no longer valid.
1100  Invalid address for the return parameter was specified.

GetXSLT_XQParameterName

*Method:* GetXSLT_XQParameterName(index as Long) as String

*Description*
Returns the name of the XSLT or XQuery parameter identified by the supplied index.

*Errors*
1111  The application object is no longer valid.
1100  Invalid address for the return parameter was specified.

GetXSLT_XQParameterXPath

*Method:* GetXSLT_XQParameterXPath(index as Long) as String

*Description*
Returns the XPath expression of the XSLT or XQuery parameter identified by the supplied index.

*Errors*
1111  The application object is no longer valid.
1100  Invalid address for the return parameter was specified.

ImportFromDatabase

*See also*

*Method:* ImportFromDatabase(pImportSettings as DatabaseConnection, pElementList as ElementList) as Document

*Return Value*
Creates a new document containing the data imported from the database.

*Description*
ImportFromDatabase imports data from a database as specified in pImportSettings and creates a new document containing the data imported from the database. Properties mandatory to be filled out are one of DatabaseConnection.File, DatabaseConnection.ADOConnection or DatabaseConnection.ODBCConnection and DatabaseConnection.SQLSelect. Additionally, you can use DatabaseConnection.AsAttributes, DatabaseConnection.ExcludeKeys, DatabaseConnection.IncludeEmptyElements and NumberDateTimeFormat to further parameterize import.

The parameter pElementList specifies which fields of the selected data gets written into the newly created document, and which are created as elements and which as attributes. This parameter can be NULL, specifying that all selected fields will be imported as XML elements.
See `GetDatabaseSettings` and `GetDatabaseImportElementList` for necessary steps preceding any import of data from a database.

**Errors**
- 1111 The application object is no longer valid.
- 1100 Invalid parameter or invalid address for the return parameter was specified.
- 1107 Import from database failed.
- 1112 Invalid database specified.
- 1114 Select statement is missing.
- 1117 Transformation to XML failed.
- 1120 Database import failed.

**Example**

```vba
Dim objImpSettings As DatabaseConnection
Set objImpSettings = objSpy.GetDatabaseSettings

objImpSettings.ADOConnection = strADOConnection
objImpSettings.SQLSelect = "SELECT * FROM MyTable"

Dim objDoc As Document
On Error Resume Next
Set objDoc = objSpy.ImportFromDatabase(objImpSettings, objSpy.GetDatabaseImportElementList(objImpSettings))
' CheckForError here
```

**ImportFromSchema**

**See also**

**Method:** `ImportFromSchema(pImportSettings as DatabaseConnection, strTable as String, pSchemaDoc as Document) as Document`

**Return Value**
Creates a new document filled with data from the specified database as specified by the schema definition in `pSchemaDoc`.

**Description**
`ImportFromSchema` imports data from a database specified in `pImportSettings`. Properties mandatory to be filled out are one of `DatabaseConnection.File`, `DatabaseConnection.ADOConnection` or `DatabaseConnection.ODBCConnection`. Additionally, you can use `DatabaseConnection.AsAttributes`, `DatabaseConnection.ExcludeKeys` and `NumberDateTimeFormat` to further parameterize import. All other properties get ignored.

`ImportFromSchema` does not use and explicit SQL statement to select the data. Instead, it expects a structure definition of the document to create in form of an XML schema document in `pSchemaDoc`. From this definition the database select statement is automatically deduced. Specify in `strTable` the table name of the import root that will become the root node in the new document.

See `GetDatabaseSettings` and `GetDatabaseTables` for necessary steps preceding an
import from a database based on a schema definition. To create the schema definition file use command 'create database schema' from the 'convert' menu of XMLSpy.

Errors
1111 The application object is no longer valid.
1100 Invalid parameter or invalid address for the return parameter was specified.
1107 Import from database failed.
1112 Invalid database specified.
1120 Database import failed.
1121 Could not create validator for the specified schema.
1122 Failed parsing schema for database import.

ImportFromText

See also

Method: ImportFromText(pImportSettings as TextImportExportSettings, pElementList as ElementList) as Document

Description
ImportFromText imports the text file as specified in pImportSettings. The parameter pElementList can be used as import filter. Either pass the list returned by a previous call to GetTextImportElementList or null to import all columns. To avoid import of unnecessary columns use ElementList.RemoveElement to remove the corresponding field names from pElementList before calling ImportFromText. The method returns the newly created document containing the imported data. This document is the same as the active document of XMLSpy.

See also Import and export of data.

Errors
1111 The application object is no longer valid.
1100 Invalid parameter or invalid address for the return parameter was specified.
1107 Import from text file failed.
1117 Transformation to XML failed.

Example
' ---------------------------------------------------------
' VBA client code fragment - import from text file
' ---------------------------------------------------------
Dim objImpSettings As TextImportExportSettings
Set objImpSettings = objSpy.GetTextImportExportSettings

objImpSettings.ImportFile = strFileName
objImpSettings.HeaderRow = False

Dim objImpDoc As Document
On Error Resume Next
Set objImpDoc = objSpy.ImportFromText(objImpSettings, objSpy.GetTextImportElementList(objImpSettings))

CheckForError
**ImportFromWord**

See also

**Method:** ImportFromWord(strFile as String) as Document

**Description**
ImportFromWord imports the MS-Word Document strFile into a new XML document.

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1100</td>
<td>Invalid parameter or invalid address for the return parameter was specified. Import from document failed.</td>
</tr>
</tbody>
</table>

**IsAPISupported**

See also

**Property:** IsAPISupported as Boolean

**Description**
Returns whether the API is supported in this version or not.

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1100</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

**MajorVersion**

See also

**Property:** MajorVersion as Integer

**Description**
Returns the application version's major number, for example 15 for 2013 versions, and 16 for 2014 versions.

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1100</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>

**MinorVersion**

See also

**Property:** MinorVersion as Integer

**Description**
Returns the application version's minor number.

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>The application object is no longer valid.</td>
</tr>
<tr>
<td>1100</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
**NewProject**

*See also*

*Method:* `NewProject(strPath as String, bDiscardCurrent as Boolean)`

*Description*

`NewProject` creates a new project. If there is already a project open that has been modified and `bDiscardCurrent` is false, then `NewProject()` fails.

*Errors*

- **1111** The application object is no longer valid.
- **1102** A project is already open but `bDiscardCurrent` is true.
- **1103** Creation of new project failed.

**OpenProject**

*See also*

*Method:* `OpenProject(strPath as String, bDiscardCurrent as Boolean, bDialog as Boolean)`

*Parameters*

- **strPath**
  Path and file name of the project to open. Can be empty if `bDialog` is true.

- **bDiscardCurrent**
  Discard currently open project and possible lose changes.

- **bDialog**
  Show dialogs for user input.

*Return Value*

None

*Description*

`OpenProject` opens an existing project. If there is already a project open that has been modified and `bDiscardCurrent` is false, then `OpenProject()` fails.

*Errors*

- **1111** The application object is no longer valid.
- **1100** Invalid parameter or invalid address for the return parameter was specified.
- **1101** Cannot open specified project.
- **1102** A project is already open but `bDiscardCurrent` is true.

**Parent**

*See also*

*Property:* `Parent` as `Application` (read-only)
Description
Accesses the XMLSpy application object.

Errors
1111 The application object is no longer valid.
1100 Invalid address for the return parameter was specified.

Quit
See also

Method: Quit()

Return Value
None

Description
This method terminates XMLSpy. All modified documents will be closed without saving the changes. This is also true for an open project.

If XMLSpy was automatically started as an automation server by a client program, the application will not shut down automatically when your client program shuts down if a project or any document is still open. Use the Quit method to ensure automatic shut-down.

Errors
1111 The application object is no longer valid.

ReloadSettings
See also

Method: ReloadSettings

Return Value

Description
The application settings are reloaded from the registry.

Available with TypeLibrary version 1.5

Errors
1111 The application object is no longer valid.

RemoveXSLT_XQParameter

Method: RemoveXSLT_XQParameter(index as Long)

Description
Removes the XSLT or XQuery parameter identified by the supplied index.

Errors
1111  The application object is no longer valid.
1100  Invalid address for the return parameter was specified.

RunMacro
See also

Method: RunMacro (strMacro as String)

Return Value

Description
Calls the specified macro either from the project scripts (if present) or from the global scripts.

Available with TypeLibrary version 1.5

Errors
1111  The application object is no longer valid.

ScriptingEnvironment
See also

Property: ScriptingEnvironment as IUnknown (read-only)

Description
Reference to any active scripting environment. This property makes it possible to access the TypeLibrary of the XMLSpyFormEditor.exe application which is used as the current scripting environment.

Available with TypeLibrary version 1.5

Errors
1111  The application object is no longer valid.
1100  Invalid address for the return parameter was specified.

ServicePackVersion
See also

Property: ServicePackVersion as Long

Description
Returns the Service Pack version number of the application. Eg: 1 for 2010 R2 SP1

Errors
1111  The application object is no longer valid.
1100  Invalid address for the return parameter was specified.

ShowApplication
See also
Method: ShowApplication(bShow as Boolean)

Return Value
None

Description
The method shows (bShow = True) or hides (bShow = False) XMLSpy.

Errors
1110 The application object is no longer valid.

ShowFindInFiles

See also

Method: ShowFindInFiles(pSettings as FindInFilesDlg) as Boolean

Return Value
Returns false if the user pressed the Cancel button, true otherwise.

Description
Displays the FindInFiles dialog preset with the given settings. The user modifications of the settings are stored in the passed dialog object.

Errors
1111 The application object is no longer valid.
1100 Invalid parameter or invalid address for the return parameter was specified.

ShowForm

See also

Method: ShowForm(strFormName as String) as Long

Return Value
Returns zero if the user pressed a Cancel button or the form calls TheView.Cancel().

Description
Displays the form strFormName.

Forms, event handlers and macros can be created with the Scripting Environment. Select "Switch to scripting environment" from the Tools menu to invoke the Scripting Environment.

Errors
1111 The application object is no longer valid.
1100 Invalid parameter or invalid address for the return parameter was specified.

Status

See also

Property: Status as ENUMApplicationStatus
**Application API Interfaces**

**Description**
Returns the current status of the running application.

**Errors**
- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

**URLDelete**

**See also**

**Method:** `URLDelete (strURL as String, strUser as String, strPassword as String)`

**Return Value**
None

**Description**
The method deletes the file at the URL `strURL`.

**Errors**
- 1111 The application object is no longer valid.
- 1109 Error deleting file at specified URL.

**URLMakeDirectory**

**See also**

**Method:** `URLMakeDirectory (strURL as String, strUser as String, strPassword as String)`

**Return Value**
None

**Description**
The method creates a new directory at the URL `strURL`.

**Errors**
- 1111 The application object is no longer valid.
- 1100 Invalid parameter specified.

**Visible**

**See also**

**Property:** `Visible as VARIANT_BOOL`

**Description**
Sets or gets the visibility attribute of XMLSpy. This standard automation property makes usage of `ShowApplication` obsolete.

**Errors**
- 1110 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.
**WarningNumber**

**See also**

**Property:** `WarningNumber` as integer

**Description**
Some methods fill the property `WarningNumber` with additional information if an error occurs.

Currently just **Documents.OpenFile** fills this property.

**Errors**
- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

**WarningText**

**See also**

**Property:** `WarningText` as String

**Description**
Some methods fill the property `WarningText` with additional information if an error occurs.

Currently just **Documents.OpenFile** fills this property.

**Errors**
- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

### 3.2.2 AuthenticContextMenu

The context menu interface provides the mean for the user to customize the context menus shown in Authentic. The interface has the methods listed in this section.

**CountItems**

**Method:** `CountItems()` nItems as long

**Return Value**
Returns the number of menu items.

**Errors**
- 2501 Invalid object.

**DeleteItem**

**Method:** `DeleteItem(IndexPosition as long)`

**Return Value**
Deletes the menu item that has the index position submitted in the first parameter.
Errors
2501 Invalid object
2502 Invalid index

GetItemText
Method: GetItemText(IndexPosition as long) MenuItemName as string

Return Value
Gets the name of the menu item located at the index position submitted in the first parameter.

Errors
2501 Invalid object
2502 Invalid index

InsertItem
Method: InsertItem(IndexPosition as long, MenuItemName as string, MacroName as string)

Return Value
Inserts a user-defined menu item at the position in the menu specified in the first parameter and having the name submitted in the second parameter. The menu item will start a macro, so a valid macro name must be submitted.

Errors
2501 Invalid object
2502 Invalid index
2503 No such macro
2504 Internal error

SetItemText
Method: SetItemText(IndexPosition as long, MenuItemName as string)

Return Value
Sets the name of the menu item located at the index position submitted in the first parameter.

Errors
2501 Invalid object
2502 Invalid index

3.2.3 AuthenticDataTransfer

Renamed from DocEditDataTransfer to AuthenticDataTransfer

The DocEditView object is renamed to OldAuthenticView.
DocEditSelection is renamed to AuthenticSelection.
DocEditEvent is renamed to AuthenticEvent.
DocEditDataTransfer is renamed to AuthenticDataTransfer.

Their usage—except for AuthenticDataTransfer—is no longer recommended. We will continue to support existing functionality for a yet undefined period of time but no new features will be added to these interfaces. All functionality available up to now in DocEditView, DocEditSelection, DocEditEvent and DocEditDataTransfer is now available via AuthenticView, AuthenticRange and AuthenticDataTransfer. Many new features have been added.

For examples on migrating from DocEdit to Authentic see the description of the different methods and properties of the different DocEdit objects.

See also

Methods

gedata

Properties

dropEffect
ownDrag
type

Description

The events OnDragOver and OnBeforeDrop provide information about the object being dragged with an instance of type AuthenticDataTransfer. It contains a description of the dragged object and its content. The latter is available either as string or a pointer to a COM object supporting the IUnknown interface.

dropEffect

See also

Property: dropEffect as long

Description
The property stores the drop effect from the default event handler. You can set the drop effect if you change this value and return TRUE for the event handler (or set AuthenticEvent.cancelBubble to TRUE if you are still using the now obsolete AuthenticEvent interface).

Errors

2101 Invalid address for the return parameter was specified.

gedata

See also
**Method:** `getData()` as Variant

**Description**
Retrieve the data associated with the dragged object. Depending on `AuthenticDataTransfer.type`, that data is either a string or a COM interface pointer of type `IUnknown`.

**Errors**
- 2101 Invalid address for the return parameter was specified.

**ownDrag**

**See also**

**Property:** `ownDrag` as Boolean (read-only)

**Description**
The property is `TRUE` if the current dragging source comes from inside Authentic View.

**Errors**
- 2101 Invalid address for the return parameter was specified.

**type**

**See also**

**Property:** `type` as String (read-only)

**Description**
Holds the type of data you get with the `DocEditDataTransfer.getData` method.

Currently supported data types are:
- `OWN` data from Authentic View itself
- `TEXT` plain text
- `UNICODETEXT` plain text as UNICODE

**Errors**
- 2101 Invalid address for the return parameter was specified.

### 3.2.4 AuthenticEventContext

The `EventContext` interface gives access to many properties of the context in which a macro is executed.

#### EvaluateXPath

**Method:** `EvaluateXPath(strExpression as string)` as `strValue as string`

**Return Value**
The method evaluates the XPath expression in the context of the node within which the event was triggered and returns a string.
Description
EvaluateXPath() executes an XPath expressions with the given event context. The result is returned as string, in the case of a sequence it is a space-separated string.

Errors
- 2201 Invalid object.
- 2202 No context.
- 2209 Invalid parameter.
- 2210 Internal error.
- 2211 XPath error.

GetEventContextType
Method: GetEventContextType() Type as AuthenticEventContextType enumeration

Return Value
Returns the context node type.

Description
GetEventContextType allows the user to determine whether the macro is in an XML node or in an XPath atomic item context. The enumeration AuthenticEventContextType is defined as follows:

authenticEventContextXML,
authenticEventContextAtomicItem,
authenticEventContextOther

If the context is a normal XML node, the GetXMLNode() function gives access to it (returns NULL if not).

Errors
- 2201 Invalid object.
- 2202 No context.
- 2209 Invalid parameter.

GetNormalizedTextValue
Method: GetNormalizedTextValue() strValue as string

Return Value
Returns the value of the current node as string

Errors
- 2201 Invalid object.
- 2202 No context.
- 2203 Invalid context
- 2209 Invalid parameter.

GetVariableValue
Method: GetVariableValue(strName as string) strValue as string

Return Value
Gets the value of the variable submitted as the parameter.

**Description**

GetVariableValue gets the variable’s value in the scope of the context.

```javascript
nZoom = parseInt( AuthenticView.EventContext.GetVariableValue( 'Zoom' ) );
if ( nZoom > 1 )
{
    AuthenticView.EventContext.SetVariableValue( 'Zoom', nZoom - 1 );
}
```

**Errors**

- 2201 Invalid object.
- 2202 No context.
- 2204 No such variable in scope
- 2205 Variable cannot be evaluated
- 2206 Variable returns sequence
- 2209 Invalid parameter

---

**GetXMLNode**

*Method:* GetXMLNode() Node as XMLData object

**Return Value**

Returns the context XML node or NULL

**Errors**

- 2201 Invalid object.
- 2202 No context.
- 2203 Invalid context
- 2209 Invalid parameter.

---

**IsAvailable**

*Method:* IsAvailable() as Boolean

**Return Value**

Returns true if EventContext is set, false otherwise.

**Errors**

- 2201 Invalid object.

---

**SetVariableValue**

*Method:* SetVariableValue(strName as string, strValue as string)

**Return Value**

Sets the value (second parameter) of the variable submitted in the first parameter.

**Description**

SetVariableValue sets the variable’s value in the scope of the context.
```javascript
nZoom = parseInt( AuthenticView.EventContext.GetVariableValue( 'Zoom' ) );
if ( nZoom > 1 )
{
    AuthenticView.EventContext.SetVariableValue( 'Zoom', nZoom - 1 );
}
```

**Errors**

- **2201** Invalid object.
- **2202** No context.
- **2204** No such variable in scope
- **2205** Variable cannot be evaluated
- **2206** Variable returns sequence
- **2207** Variable read-only
- **2208** No modification allowed

### 3.2.5 AuthenticRange

**See also**

The first table lists the properties and methods of AuthenticRange that can be used to navigate through the document and select specific portions.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Clone</td>
</tr>
<tr>
<td>FirstTextPosition</td>
<td>CollapsToBegin</td>
</tr>
<tr>
<td>FirstXMLElement</td>
<td>CollapsToEnd</td>
</tr>
<tr>
<td>FirstXMLElementOffset</td>
<td>ExpandTo</td>
</tr>
<tr>
<td>LastTextPosition</td>
<td>Goto</td>
</tr>
<tr>
<td>LastXMLElement</td>
<td>GotoNext</td>
</tr>
<tr>
<td>LastXMLElementOffset</td>
<td>GotoPrevious</td>
</tr>
<tr>
<td>Parent</td>
<td>IsEmpty</td>
</tr>
<tr>
<td></td>
<td>IsEqual</td>
</tr>
<tr>
<td></td>
<td>MoveBegin</td>
</tr>
<tr>
<td></td>
<td>MoveEnd</td>
</tr>
<tr>
<td></td>
<td>NextCursorPosition</td>
</tr>
<tr>
<td></td>
<td>PreviousCursorPosition</td>
</tr>
</tbody>
</table>

The following table lists the content modification methods, most of which can be found on the right/button mouse menu.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Edit operations</th>
<th>Dynamic table operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Copy</td>
<td>AppendRow</td>
</tr>
<tr>
<td></td>
<td>Cut</td>
<td>DeleteRow</td>
</tr>
<tr>
<td></td>
<td>Delete</td>
<td>DuplicateRow</td>
</tr>
<tr>
<td></td>
<td>IsCopyEnabled</td>
<td>InsertRow</td>
</tr>
<tr>
<td></td>
<td>IsCutEnabled</td>
<td>IsFirstRow</td>
</tr>
<tr>
<td></td>
<td>IsDeleteEnabled</td>
<td>IsInDynamicTable</td>
</tr>
<tr>
<td></td>
<td>IsPasteEnabled</td>
<td>IsLastRow</td>
</tr>
<tr>
<td></td>
<td>Paste</td>
<td>MoveRowDown</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MoveRowUp</td>
</tr>
</tbody>
</table>

The following methods provide the functionality of the Authentic entry helper windows for range objects.
Operations of the entry helper windows

<table>
<thead>
<tr>
<th>Elements</th>
<th>Attributes</th>
<th>Entities</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPerformActionWith</td>
<td>GetElementAttributeValue</td>
<td>GetEntityNames</td>
</tr>
<tr>
<td>CanPerformAction</td>
<td>GetElementAttributeValue</td>
<td></td>
</tr>
<tr>
<td>PerformAction</td>
<td>GetElementAttributeValue</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetElementAttributeNames</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetElementHierarchy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HasElementAttribute</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IsTextStateApplied</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SetElementAttributeValue</td>
<td></td>
</tr>
</tbody>
</table>

Description

AuthenticRange objects are the 'cursor' selections of the automation interface. You can use them to point to any cursor position in the Authentic view, or select a portion of the document. The operations available for AuthenticRange objects then work on this selection in the same way, as the corresponding operations of the user interface do with the current user interface selection. The main difference is that you can use an arbitrary number of AuthenticRange objects at the same time, whereas there is exactly one cursor selection in the user interface.

To get to an initial range object use AuthenticView.Selection, to obtain a range corresponding with the current cursor selection in the user interface. Alternatively, some trivial ranges are accessible via the read/write properties AuthenticView.DocumentBegin, AuthenticView.DocumentEnd, and AuthenticView.WholeDocument. The most flexible method is AuthenticView.Goto, which allows navigation to a specific portion of the document within one call. For more complex selections, combine the above, with the various navigation methods on range objects listed in the first table on this page.

Another method to select a portion of the document is to use the position properties of the range object. Two positioning systems are available and can be combined arbitrarily:

- **Absolute** text cursor positions, starting with position 0 at the document beginning, can be set and retrieved for the beginning and end of a range. For more information see FirstTextPosition and LastTextPosition. This method requires complex internal calculations and should be used with care.

- **The XMLData** element and a text position inside this element, can be set and retrieved for the beginning and end of a range. For more information see FirstXMLData, FirstXMLDataOffset, LastXMLData, and LastXMLDataOffset. This method is very efficient but requires knowledge on the underlying document structure. It can be used to locate XMLData objects and perform operations on them otherwise not accessible through the user interface.

Modifications to the document content can be achieved by various methods:

- The **Text** property allows you to retrieve the document text selected by the range object. If set, the selected document text gets replaced with the new text.
- The standard document edit functions Cut, Copy, Paste and Delete.
- Table operations for tables that can grow dynamically.
- Methods that map the functionality of the Authentic entry helper windows.
- Access to the XMLData objects of the underlying document to modify them directly.

AppendRow

See also
**Method:** AppendRow() as Boolean

**Description**
If the beginning of the range is inside a dynamic table, this method inserts a new row at the end of the selected table. The selection of the range is modified to point to the beginning of the new row. The function returns true if the append operation was successful, otherwise false.

**Errors**
- 2001: The authentic range object or its related view object is no longer valid.
- 2005: Invalid address for the return parameter was specified.

**Examples**
```
' ---------------------------------------------------------
' Scripting environment - VBScript
' Append row at end of current dynamically growable table
' ---------------------------------------------------------
Dim objRange
' we assume that the active document is open in authentic view mode
Set objRange = Application.ActiveDocument.AuthenticView.Selection

' check if we can insert something
If objRange.IsInDynamicTable Then
    objRange.AppendRow
    ' objRange points to beginning of new row
    objRange.Select
End If
```

**Application**

**See also**

**Property:** Application as Application (read-only)

**Description**
Accesses the XMLSpy application object.

**Errors**
- 2001: The authentic range object or its related view object is no longer valid.
- 2005: Invalid address for the return parameter was specified.

**CanPerformAction**

**See also**

**Method:** CanPerformAction (eAction as SPYAuthenticActions, strElementName as String) as Boolean

**Description**
CanPerformAction and its related methods enable access to the entry-helper functions of Authentic. This function allows easy and consistent modification of the document content, without having to know exactly where the modification will take place. The beginning of the range object is used to locate the next valid location where the specified action can be performed. If the location can be found, the method returns True, otherwise it returns False.
HINT: To find out all valid element names for a given action, use CanPerformActionWith.

Errors
- 2001 The authentic range object or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.
- 2007 Invalid action was specified.

Examples
See PerformAction.

CanPerformActionWith
See also

Method: CanPerformActionWith (eAction as SPYAuthenticActions, out_arrElementNames as Variant)

Description
PerformActionWith and its related methods, enable access to the entry-helper functions of Authentic. These function allows easy and consistent modification of the document content without having to know exactly where the modification will take place.

This method returns an array of those element names that the specified action can be performed with.

HINT: To apply the action use CanPerformActionWith.

Errors
- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.
- 2007 Invalid action was specified.

Examples
See PerformAction.

Clone
See also

Method: Clone() as AuthenticRange

Description
Returns a copy of the range object.

Errors
- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

CollapsToBegin
See also

Method: CollapsToBegin() as AuthenticRange
Description
Sets the end of the range object to its begin. The method returns the modified range object.

Errors
- 2001: The authentic range object, or its related view object is no longer valid.
- 2005: Invalid address for the return parameter was specified.

CollapsToEnd
See also

Method: CollapsToEnd() as AuthenticRange

Description
Sets the beginning of the range object to its end. The method returns the modified range object.

Errors
- 2001: The authentic range object, or its related view object is no longer valid.
- 2005: Invalid address for the return parameter was specified.

Copy
See also

Method: Copy() as Boolean

Description
Returns False if the range contains no portions of the document that may be copied.
Returns True if text, and in case of fully selected XML elements the elements as well, has been copied to the copy/paste buffer.

Errors
- 2001: The authentic range object or its related view object is no longer valid.
- 2005: Invalid address for the return parameter was specified.

Cut
See also

Method: Cut() as Boolean

Description
Returns False if the range contains portions of the document that may not be deleted.
Returns True after text, and in case of fully selected XML elements the elements as well, has been deleted from the document and saved in the copy/paste buffer.

Errors
- 2001: The authentic range object, or its related view object is no longer valid.
- 2005: Invalid address for the return parameter was specified.
Delete

See also

Method: Delete() as Boolean

Description
Returns *False* if the range contains portions of the document that may not be deleted.
Returns *True* after text, and in case of fully selected XML elements the elements as well, has been deleted from the document.

Errors
- 2001 The authentic range object or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

DeleteRow

See also

Method: DeleteRow() as Boolean

Description
If the beginning of the range is inside a dynamic table, this method deletes the selected row. The selection of the range gets modified to point to the next element after the deleted row. The function returns *true*, if the delete operation was successful, otherwise *false*.

Errors
- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

Examples
```
' ---------------------------------------------------------
' Scripting environment - VBScript
' Delete selected row from dynamically growing table
' ---------------------------------------------------------
Dim objRange
' we assume that the active document is open in authentic view mode
Set objRange = Application.ActiveDocument.AuthenticView.Selection

' check if we are in a table
If objRange.IsInDynamicTable Then
    objRange.DeleteRow
End If
```

DuplicateRow

See also

Method: DuplicateRow() as Boolean

Description
If the beginning of the range is inside a dynamic table, this method inserts a duplicate of the current row after the selected one. The selection of the range gets modified to point to the beginning of the new row. The function returns *true* if the duplicate operation was successful,
otherwise false.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

Examples

```vbscript
' Scripting environment - VBScript
' duplicate row in current dynamically growable table
' ---------------------------------------------------------
Dim objRange
' we assume that the active document is open in authentic view mode
Set objRange = Application.ActiveDocument.AuthenticView.Selection

' check if we can insert something
If objRange.IsInDynamicTable Then
    objRange.DuplicateRow
    ' objRange points to beginning of new row
    objRange.Select
End If
```

EvaluateXPath

**Method:** `EvaluateXPath` *(strExpression as string) strValue as string*

**Return Value**
The method returns a string

**Description**
`EvaluateXPath()` executes an XPath expressions with the context node being the beginning of
the range selection. The result is returned as string, in the case of a sequence it is a space-
separated string. If XML context node is irrelevant, the user may provide any node, like
`AuthenticView.XMLDataRoot`.

Errors

- 2001 Invalid object
- 2005 Invalid parameter
- 2008 Internal error
- 2202 Missing context node
- 2211 XPath error

ExpandTo

**See also**

**Method:** `ExpandTo` *(eKind as SPYAuthenticElementKind), as AuthenticRange*

**Description**
Selects the whole element of type `eKind`, that starts at, or contains, the first cursor position
of the range. The method returns the modified range object.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
2003 Range expansion would be beyond end of document.
2005 Invalid address for the return parameter was specified.

**FirstTextPosition**

**See also**

*Property:* **FirstTextPosition** as Long

**Description**
Set or get the left-most text position index of the range object. This index is always less or equal to **LastTextPosition**. Indexing starts with 0 at document beginning, and increments with every different position that the text cursor can occupy. Incrementing the test position by 1, has the same effect as the cursor-right key. Decrementing the test position by 1 has the same effect as the cursor-left key.

If you set **FirstTextPosition** to a value greater than the current **LastTextPosition**, **LastTextPosition** gets set to the new **FirstTextPosition**.

HINT: Use text cursor positions with care, since this is a costly operation compared to XMLData based cursor positioning.

**Errors**
2001 The authentic range object, or its related view object is not valid.
2005 Invalid address for the return parameter was specified.
2006 A text position outside the document was specified.

**Examples**

```
' ---------------------------------------
' Scripting environment - VBScrip
' ---------------------------------------
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

nDocEndPosition = objAuthenticView.DocumentEnd.FirstTextPosition

' let's create a range that selects the whole document
' in an inefficient way
Dim objRange
' we need to get a (any) range object first
Set objRange = objAuthenticView.DocumentBegin
objRange.FirstTextPosition = nDocStartPosition
objRange.LastTextPosition = nDocEndPosition

' let's check if we got it right
If objRange.isEqual(objAuthenticView.WholeDocument) Then
    MsgBox "Test using direct text cursor positioning was ok"
Else
    MsgBox "Ooops!"
End If
```
FirstXMLData

See also

**Property:** `FirstXMLData` as `XMLData`

**Description**
Set or get the first `XMLData` element in the underlying document that is partially, or completely selected by the range. The exact beginning of the selection is defined by the `FirstXMLDataOffset` attribute.

Whenever you set `FirstXMLData` to a new data object, `FirstXMLDataOffset` gets set to the first cursor position inside this element. Only `XMLData` objects that have a cursor position may be used. If you set `FirstXMLData` / `FirstXMLDataOffset` selects a position greater than the current `LastXMLData` / `LastXMLDataOffset`, the latter gets moved to the new start position.

HINT: You can use the `FirstXMLData` and `LastXMLData` properties, to directly access and manipulate the underlying XML document in those cases where the methods available with the `AuthenticRange` object are not sufficient.

**Errors**
- 2001  The authentic range object, or its related view object is not valid.
- 2005  Invalid address for the return parameter was specified.
- 2008  Internal error
- 2009  The `XMLData` object cannot be accessed.

**Examples**
```vbnet
' -----------------------------------------------
' Scripting environment - VBScript
' show name of currently selected XMLData element
' -----------------------------------------------
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

Dim objXMLData
Set objXMLData = objAuthenticView.Selection.FirstXMLData
' authentic view adds a 'text' child element to elements
' of the document which have content. So we have to go one
' element up.
Set objXMLData = objXMLData.Parent
MsgBox "Current selection selects element " & objXMLData.Name
```

FirstXMLDataOffset

See also

**Property:** `FirstXMLDataOffset` as Long

**Description**
Set or get the cursor position offset inside `FirstXMLData` element for the beginning of the range. Offset positions are based on the characters returned by the `Text` property, and start with 0. When setting a new offset, use -1 to set the offset to the last possible position in the element.
The following cases require specific attention:

- The textual form of entries in Combo Boxes, Check Boxes and similar controls can be different from what you see on screen. Although the data offset is based on this text, there only two valid offset positions, one at the beginning and one at the end of the entry. An attempt to set the offset to somewhere in the middle of the entry, will result in the offset being set to the end.

- The textual form of XML Entities might differ in length from their representation on the screen. The offset is based on this textual form.

If \texttt{FirstXMLData/FirstXMLDataOffset} selects a position after the current \texttt{LastXMLData/LastXMLDataOffset}, the latter gets moved to the new start position.

**Errors**

- 2001 The authentic range object, or its related view object is not valid.
- 2005 Invalid offset was specified.
- 2006 Invalid address for the return parameter was specified.

**Examples**

```vbscript
' ---------------------------------------------
' Scripting environment - VBScript
' Select the complete text of an XMLData element
' using XMLData based selection and ExpandTo
' ---------------------------------------------
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

' first we use the XMLData based range properties
' to select all text of the first XMLData element
' in the current selection
Dim objRange
Set objRange = objAuthenticView.Selection
objRange.FirstXMLDataOffset = 0 ' start at beginning of element text
objRange.LastXMLData = objRange.FirstXMLData ' select only one element
objRange.LastXMLDataOffset = -1 ' select till its end

' the same can be achieved with the ExpandTo method
Dim objRange2
Set objRange2 = objAuthenticView.Selection.ExpandTo(spyAuthenticTag)

' were we successful?
If objRange.IsEqual(objRange2) Then
    objRange.Select()
Else
    MsgBox "Oops"
End If
```

**GetElementAttributeNames**

**See also**

- **Method**: \texttt{GetElementAttributeNames(strElementName as String, out_arrAttributeNames as Variant)}
**Description**
Retrieve the names of all attributes for the enclosing element with the specified name. Use the element/attribute pairs, to set or get the attribute value with the methods `GetElementAttributeValue` and `SetElementAttributeValue`.

**Errors**
- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid element name was specified.
  Invalid address for the return parameter was specified.

**Examples**
See `SetElementAttributeValue`.

### GetElementAttributeValue

**See also**

**Method:** `GetElementAttributeValue`(strElementName as String, strAttributeName as String) as String

**Description**
Retrieve the value of the attribute specified in `strAttributeName`, for the element identified with `strElementName`. If the attribute is supported but has no value assigned, the empty string is returned. To find out the names of attributes supported by an element, use `GetElementAttributeNames`, or `HasElementAttribute`.

**Errors**
- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid element name was specified.
  Invalid attribute name was specified.
  Invalid address for the return parameter was specified.

**Examples**
See `SetElementAttributeValue`.

### GetElementHierarchy

**See also**

**Method:** `GetElementHierarchy`(out_arrElementNames as Variant)

**Description**
Retrieve the names of all XML elements that are parents of the current selection. Inner elements get listed before enclosing elements. An empty list is returned whenever the current selection is not inside a single `XMLData` element.

The names of the element hierarchy, together with the range object uniquely identify `XMLData` elements in the document. The attributes of these elements can be directly accessed by `GetElementAttributeNames`, and related methods.

**Errors**
- 2001 The authentic range object, or its related view object is no longer valid.
2005  Invalid address for the return parameter was specified.

C# Examples
--------------------------------------------
C#  
--------------------------------------------

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            XMLSpyLib.Application app = new XMLSpyLib.Application();
            app.ShowApplication(true);

            XMLSpyLib.AuthenticView view = app.ActiveDocument.AuthenticView;
            XMLSpyLib.AuthenticRange range = view.DocumentBegin;

            object o = null;
            range.GetElementHierarchy(ref o);

            object[] elements = (object[])o;

            foreach (string e in elements)
            {
                Console.WriteLine(e);
            }
        }
    }
}

Also see: SetElementAttributeValue.

GetEntityNames

See also

Method: GetEntityNames(out_arrEntityNames as Variant)

Description
Retrieve the names of all defined entities. The list of retrieved entities is independent of the current selection, or location. Use one of these names with the InsertEntity function.

Errors
2001   The authentic range object, or its related view object is no longer valid.
2005   Invalid address for the return parameter was specified.

Examples
See: GetElementHierarchy and InsertEntity.
**GetVariableValue**

*Method:* `GetVariableValue(strName as string) strVal as string`

**Return Value**
Gets the value of the variable named as the method's parameter.

**Errors**
- 2001: Invalid object.
- 2202: No context.
- 2204: No such variable in scope.
- 2205: Variable cannot be evaluated.
- 2206: Variable returns sequence.
- 2209: Invalid parameter.

**Goto**

*See also*

*Method:* `Goto(eKind as SPYAuthenticElementKind, nCount as Long, eFrom as SPYAuthenticDocumentPosition) as AuthenticRange`

**Description**
Sets the range to point to the beginning of the `nCount` element of type `eKind`. The start position is defined by the parameter `eFrom`.

Use positive values for `nCount` to navigate to the document end. Use negative values to navigate to the beginning of the document. The method returns the modified range object.

**Errors**
- 2001: The authentic range object, or its related view object is no longer valid.
- 2004: Target lies before begin of document.
- 2005: Invalid element kind specified.
- 2009: Invalid start position specified.
- 2010: Invalid address for the return parameter was specified.

**GotoNext**

*See also*

*Method:* `GotoNext(eKind as SPYAuthenticElementKind) as AuthenticRange`

**Description**
Sets the range to the beginning of the next element of type `eKind`. The method returns the modified range object.

**Errors**
- 2001: The authentic range object, or its related view object is no longer valid.
- 2005: Invalid element kind specified.
- 2009: Invalid address for the return parameter was specified.
Examples

' -------------------------------
' Scripting environment - VBScript
' Scan through the whole document word-by-word
' -------------------------------
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

Dim objRange
Set objRange = objAuthenticView.DocumentBegin
Dim bEndOfDocument
bEndOfDocument = False

On Error Resume Next
While Not bEndOfDocument
    objRange.GotoNext(spyAuthenticWord).Select
    If ((Err.number - vbObjectError) = 2003) Then
        bEndOfDocument = True
        Err.Clear
    ElseIf (Err.number <> 0) Then
        Err.Raise ' forward error
    End If
End If
GotoNextCursorPosition

See also

Method: GotoNextCursorPosition() as AuthenticRange

Description
Sets the range to the next cursor position after its current end position. Returns the modified object.

Errors
  2001 The authentic range object, or its related view object is no longer valid.
  2003 Target lies after end of document.
  2005 Invalid address for the return parameter was specified.

GotoPrevious

See also

Method: GotoPrevious(eKind as SPYAuthenticElementKind) as AuthenticRange

Description
Sets the range to the beginning of the element of type eKind which is before the beginning of the current range. The method returns the modified range object.

Errors
  2001 The authentic range object, or its related view object is no longer valid.
  2004 Target lies before beginning of document.
2005  Invalid element kind specified.
       Invalid address for the return parameter was specified.

Examples

' ----------------------------------------------------------
' Scripting environment - VBScript
' Scan through the whole document tag-by-tag
' ----------------------------------------------------------
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

Dim objRange
Set objRange = objAuthenticView.DocumentEnd
Dim bEndOfDocument
bBeginOfDocument = False
On Error Resume Next
While Not bBeginOfDocument
   objRange.GotoPrevious(spyAuthenticTag).Select
   If ((Err.number - vbObjecterror) = 2004) Then
      bBeginOfDocument = True
      Err.Clear
   ElseIf (Err.number <> 0) Then
      Err.Raise ' forward error
   End If
End If
Wend

GotoPreviousCursorPosition

See also

Method: GotoPreviousCursorPosition() as AuthenticRange

Description
Set the range to the cursor position immediately before the current position. Returns the modified object.

Errors

2001  The authentic range object, or its related view object is no longer valid.
2004  Target lies before begin of document.
2005  Invalid address for the return parameter was specified.

HasElementAttribute

See also

Method: HasElementAttribute (strElementName as String, strAttributeName as String) as Boolean

Description
Tests if the enclosing element with name strElementName, supports the attribute specified in strAttributeName.
Errors
2001 The authentic range object, or its related view object is no longer valid.
2005 Invalid element name was specified.
Invalid address for the return parameter was specified.

InsertEntity
See also

Method: InsertEntity(strEntityName as String)

Description
Replace the ranges selection with the specified entity. The specified entity must be one of the entity names returned by GetEntityNames.

Errors
2001 The authentic range object, or its related view object is no longer valid.
2005 Unknown entry name was specified.

Examples
' ---------------------------------------------------------
' Scripting environment - VBScript
' Insert the first entity in the list of available entities
' ---------------------------------------------------------
Dim objRange
' we assume that the active document is open in authentic view mode
Set objRange = Application.ActiveDocument.AuthenticView.Selection

' first we get the names of all available entities as they
' are shown in the entry helper of XMLSpy
Dim arrEntities
objRange.GetEntityNames arrEntities

' we insert the first one of the list
If UBound(arrEntities) >= 0 Then
    objRange.InsertEntity arrEntities(0)
Else
    MsgBox "Sorry, no entities are available for this document"
End If

InsertRow
See also

Method: InsertRow() as Boolean

Description
If the beginning of the range is inside a dynamic table, this method inserts a new row before the current one. The selection of the range, gets modified to point to the beginning of the newly inserted row. The function returns true if the insert operation was successful, otherwise false.

Errors
2001 The authentic range object, or its related view object is no longer valid.
2005 Invalid address for the return parameter was specified.
Examples
' -------------------------------------------------------------
' Scripting environment - VBScript
' Insert row at beginning of current dynamically growing table
' -------------------------------------------------------------
Dim objRange
' we assume that the active document is open in authentic view mode
Set objRange = Application.ActiveDocument.AuthenticView.Selection

' check if we can insert something
If objRange.IsInDynamicTable Then
  objRange.InsertRow
  ' objRange points to beginning of new row
  objRange.Select
End If

IsCopyEnabled
See also

Property: IsCopyEnabled as Boolean (read-only)

Description
Checks if the copy operation is supported for this range.

Errors
  2001  The authentic range object, or its related view object is no longer valid.
  2005  Invalid address for the return parameter was specified.

IsCutEnabled
See also

Property: IsCutEnabled as Boolean (read-only)

Description
Checks if the cut operation is supported for this range.

Errors
  2001  The authentic range object, or its related view object is no longer valid.
  2005  Invalid address for the return parameter was specified.

IsDeleteEnabled
See also

Property: IsDeleteEnabled as Boolean (read-only)

Description
Checks if the delete operation is supported for this range.

Errors
  2001  The authentic range object, or its related view object is no longer valid.
2005 Invalid address for the return parameter was specified.

**IsEmpty**

See also

*Method:* `IsEmpty()` as Boolean

**Description**

Tests if the first and last position of the range are equal.

**Errors**

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

**IsEqual**

See also

*Method:* `IsEqual (objCmpRange as AuthenticRange)` as Boolean

**Description**

Tests if the start and end of both ranges are the same.

**Errors**

- 2001 One of the two range objects being compared, is invalid.
- 2005 Invalid address for a return parameter was specified.

**IsFirstRow**

See also

*Property:* `IsFirstRow` as Boolean (read-only)

**Description**

Test if the range is in the first row of a table. Which table is taken into consideration depends on the extend of the range. If the selection exceeds a single row of a table, the check is if this table is the first element in an embedding table. See the entry helpers of the user manual for more information.

**Errors**

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

**IsInDynamicTable**

See also

*Method:* `IsInDynamicTable()` as Boolean

**Description**

Test if the whole range is inside a table that supports the different row operations like 'insert', 'append', duplicate, etc.
Errors
2001 The authentic range object, or its related view object is no longer valid.
2005 Invalid address for the return parameter was specified.

IsLastRow
See also

Property: IsLastRow as Boolean (read-only)

Description
Test if the range is in the last row of a table. Which table is taken into consideration depends on the extend of the range. If the selection exceeds a single row of a table, the check is if this table is the last element in an embedding table. See the entry helpers of the user manual for more information.

Errors
2001 The authentic range object, or its related view object is no longer valid.
2005 Invalid address for the return parameter was specified.

IsPasteEnabled
See also

Property: IsPasteEnabled as Boolean (read-only)

Description
Checks if the paste operation is supported for this range.

Errors
2001 The authentic range object, or its related view object is no longer valid.
2005 Invalid address for the return parameter was specified.

IsSelected

Property: IsSelected as Boolean

Description
Returns true() if selection is present. The selection range still can be empty: that happens when e.g. only the cursor is set.

IsTextStateApplied
See also

Method: IsTextStateApplied (i_strElementName as String) as Boolean

Description
Checks if all the selected text is embedded into an XML Element with name i_strElementName. Common examples for the parameter i_strElementName are "strong", "bold" or "italic".
Errors
2001 The authentic range object, or its related view object is no longer valid.
2005 Invalid address for the return parameter was specified.

LastTextPosition

See also

Property: LastTextPosition as Long

Description
Set or get the rightmost text position index of the range object. This index is always greater or equal to FirstTextPosition. Indexing starts with 0 at the document beginning, and increments with every different position that the text cursor can occupy. Incrementing the test position by 1, has the same effect as the cursor-right key. Decreasing the test position by 1 has the same effect as the cursor-left key.

If you set LastTextPosition to a value less then the current FirstTextPosition, FirstTextPosition gets set to the new LastTextPosition.

HINT: Use text cursor positions with care, since this is a costly operation compared to XMLData based cursor positioning.

Errors
2001 The authentic range object, or its related view object is not valid.
2005 Invalid address for the return parameter was specified.
2006 A text position outside the document was specified.

Examples
' ---------------------------------------
' Scripting environment - VBScript
' ---------------------------------------
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView
nDocEndPosition = objAuthenticView.DocumentEnd.FirstTextPosition

' let's create a range that selects the whole document
' in an inefficient way
Dim objRange
' we need to get a (any) range object first
Set objRange = objAuthenticView.DocumentBegin
objRange.FirstTextPosition = nDocStartPosition
objRange.LastTextPosition = nDocEndPosition

' let's check if we got it right
If objRange isEqual(objAuthenticView.WholeDocument) Then
    MsgBox "Test using direct text cursor positioning was ok"
Else
    MsgBox "Oops!"
End If
**LastXMLData**

**See also**

**Property:** LastXMLData as XMLData

**Description**
Set or get the last XMLData element in the underlying document that is partially or completely selected by the range. The exact end of the selection is defined by the LastXMLDataOffset attribute.

Whenever you set LastXMLData to a new data object, LastXMLDataOffset gets set to the last cursor position inside this element. Only XMLData objects that have a cursor position may be used. If you set LastXMLData/LastXMLDataOffset, select a position less then the current FirstXMLData/FirstXMLDataOffset, the latter gets moved to the new end position.

HINT: You can use the FirstXMLData and LastXMLData properties to directly access and manipulate the underlying XML document in those cases, where the methods available with the AuthenticRange object are not sufficient.

**Errors**

- **2001** The authentic range object, or its related view object is not valid.
- **2005** Invalid address for the return parameter was specified.
- **2008** Internal error
- **2009** The XMLData object cannot be accessed.

**LastXMLDataOffset**

**See also**

**Property:** LastXMLDataOffset as Long

**Description**
Set or get the cursor position inside LastXMLData element for the end of the range.

Offset positions are based on the characters returned by the Text property and start with 0. When setting a new offset, use -1 to set the offset to the last possible position in the element. The following cases require specific attention:

- The textual form of entries in Combo Boxes, Check Boxes and similar controls can be different from what you see on the screen. Although, the data offset is based on this text, there only two valid offset positions, one at the beginning and one at the end of the entry. An attempt to set the offset to somewhere in the middle of the entry, will result in the offset being set to the end.
- The textual form of XML Entities might differ in length from their representation on the screen. The offset is based on this textual form.

If LastXMLData/LastXMLDataOffset selects a position before FirstXMLData/FirstXMLDataOffset, the latter gets moved to the new end position.

**Errors**

- **2001** The authentic range object, or its related view object is not valid.
Invalid offset was specified.
Invalid address for the return parameter was specified.

Examples

`---------------------------------------------`
'Scripting environment - VBScript
'Select the complete text of an XMLData element
'use XMLData based selection and ExpandTo
`---------------------------------------------`
Dim objAuthenticView
'we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

'first we use the XMLData based range properties
'to select all text of the first XMLData element
'in the current selection
Dim objRange
Set objRange = objAuthenticView.Selection
objRange.FirstXMLDataOffset = 0  'start at beginning of element text
objRange.LastXMLData = objRange.FirstXMLData 'select only one element
objRange.LastXMLDataOffset = -1  'select till its end

'the same can be achieved with the ExpandTo method
Dim objRange2
Set objRange2 = objAuthenticView.Selection.ExpandTo(spyAuthenticTag)

'were we successful?
If objRange.IsEqual(objRange2) Then
    objRange.Select()
Else
    MsgBox "Ooops"
End If

MoveBegin

See also

Method: MoveBegin (eKind as SPYAuthenticElementKind, nCount as Long) as AuthenticRange

Description
Move the beginning of the range to the beginning of the nCount element of type eKind. Counting starts at the current beginning of the range object.

Use positive numbers for nCount to move towards the document end, use negative numbers to move towards document beginning. The end of the range stays unmoved, unless the new beginning would be larger than it. In this case, the end is moved to the new beginning. The method returns the modified range object.

Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>The authentic range object, or its related view object is no longer valid.</td>
</tr>
<tr>
<td>2003</td>
<td>Target lies after end of document.</td>
</tr>
<tr>
<td>2004</td>
<td>Target lies before beginning of document.</td>
</tr>
<tr>
<td>2005</td>
<td>Invalid element kind specified.</td>
</tr>
<tr>
<td></td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
MoveEnd

See also

Method: MoveEnd (eKind as SPYAuthenticElementKind, nCount as Long) as AuthenticRange

Description
Move the end of the range to the begin of the nCount element of type eKind. Counting starts at the current end of the range object.

Use positive numbers for nCount to move towards the document end, use negative numbers to move towards document beginning. The beginning of the range stays unmoved, unless the new end would be less than it. In this case, the beginning gets moved to the new end. The method returns the modified range object.

Errors
2001 The authentic range object, or its related view object is no longer valid.
2003 Target lies after end of document.
2004 Target lies before begin of document.
2005 Invalid element kind specified.
Invalid address for the return parameter was specified.

MoveRowDown

See also

Method: MoveRowDown() as Boolean

Description
If the beginning of the range is inside a dynamic table and selects a row which is not the last row in this table, this method swaps this row with the row immediately below. The selection of the range moves with the row, but does not otherwise change. The function returns true if the move operation was successful, otherwise false.

Errors
2001 The authentic range object or its related view object is no longer valid.
2005 Invalid address for the return parameter was specified.

MoveRowUp

See also

Method: MoveRowUp() as Boolean

Description
If the beginning of the range is inside a dynamic table and selects a row which is not the first row in this table, this method swaps this row with the row above. The selection of the range moves with the row, but does not change otherwise. The function returns true if the move operation was successful, otherwise false.

Errors
2001 The authentic range object, or its related view object is no longer valid.
Invalid address for the return parameter was specified.

**Parent**

See also

*Property*: `Parent as AuthenticView` (read-only)

**Description**
Access the view that owns this range object.

**Errors**
- 2001: The authentic range object, or its related view object is no longer valid.
- 2005: Invalid address for the return parameter was specified.

**Paste**

See also

*Method*: `Paste()` as Boolean

**Description**
Returns `False` if the copy/paste buffer is empty, or its content cannot replace the current selection.

Otherwise, deletes the current selection, inserts the content of the copy/paste buffer, and returns `True`.

**Errors**
- 2001: The authentic range object, or its related view object is no longer valid.
- 2005: Invalid address for the return parameter was specified.

**PerformAction**

See also

*Method*: `PerformAction(eAction as SPYAuthenticActions, strElementName as String)` as Boolean

**Description**
PerformAction and its related methods, give access to the entry-helper functions of Authentic. This function allows easy and consistent modification of the document content without a need to know exactly where the modification will take place. The beginning of the range object is used to locate the next valid location where the specified action can be performed. If no such location can be found, the method returns `False`. Otherwise, the document gets modified and the range points to the beginning of the modification.

**HINT**: To find out element names that can be passed as the second parameter use `CanPerformActionWith`.

**Errors**
- 2001: The authentic range object, or its related view object is no longer valid.
- 2005: Invalid address for the return parameter was specified.
- 2007: Invalid action was specified.
Examples

' ' -------------------------------
' Scripting environment - VBScript
' Insert the innermost element
' ' -------------------------------
Dim objRange
' we assume that the active document is open in authentic view mode
Set objRange = Application.ActiveDocument.AuthenticView.Selection

' we determine the elements that can be inserted at the current position
Dim arrElements()
objRange.CanPerformActionWith spyAuthenticInsertBefore, arrElements

' we insert the first (innermost) element
If UBound(arrElements) >= 0 Then
    objRange.PerformAction spyAuthenticInsertBefore, arrElements(0)
    ' objRange now points to the beginning of the inserted element
    ' we set a default value and position at its end
    objRange.Text = "Hello"
    objRange.ExpandTo(spyAuthenticTag).CollapsToEnd().Select
Else
    MsgBox "Can't insert any elements at current position"
End If

Select
See also

Method: Select()

Description
Makes this range the current user interface selection. You can achieve the same result using:
' objRange.Parent.Selection = objRange'

Errors
2001 The authentic range object or its related view object is no longer valid.

Examples

' ' -------------------------------
' Scripting environment - VBScript
' ' -------------------------------
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

' set current selection to end of document
objAuthenticView.DocumentEnd.Select()

SelectNext
See also

Method: SelectNext (eKind as SPYAuthenticElementKind) as AuthenticRange
Description
Selects the element of type \texttt{eKind} after the current end of the range. The method returns the modified range object.

Errors
2001 The authentic range object, or its related view object is no longer valid.
2003 Target lies after end of document.
2005 Invalid element kind specified.
Invalid address for the return parameter was specified.

Examples
' --------------------------------------------
' Scripting environment - VBScrip
' Scan through the whole document word-by-word
' --------------------------------------------
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

Dim objRange
Set objRange = objAuthenticView.DocumentBegin
Dim bEndOfDocument
bEndOfDocument = False
On Error Resume Next
While Not bEndOfDocument
    objRange.SelectNext(spyAuthenticWord).Select
    If ((Err.number - vbObjecterror) = 2003) Then
        bEndOfDocument = True
        Err.Clear
    ElseIf (Err.number <> 0) Then
        Err.Raise ' forward error
    End If
Wend

SelectPrevious

See also

\textit{Method:} \texttt{GotoPrevious(eKind as SPYAuthenticElementKind) as AuthenticRange}

Description
Selects the element of type \texttt{eKind} before the current beginning of the range. The method returns the modified range object.

Errors
2001 The authentic range object, or its related view object is no longer valid.
2004 Target lies before begin of document.
2005 Invalid element kind specified.
Invalid address for the return parameter was specified.

Examples
' --------------------------------------------
' Scripting environment - VBScript
' Scan through the whole document tag-by-tag
' ---------------------------------------------
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

Dim objRange
Set objRange = objAuthenticView.DocumentEnd
Dim bEndOfDocument
bBeginOfDocument = False
On Error Resume Next
While Not bBeginOfDocument
    objRange.SelectPrevious(spyAuthenticTag).Select
    If ((Err.number - vbObjecterror) = 2004) Then
        bBeginOfDocument = True
        Err.Clear
    ElseIf (Err.number <> 0) Then
        Err.Raise ' forward error
    End If
Wend

SetElementAttributeValue

See also

**Method:** SetElementAttributeValue (strElementName as String, strAttributeName as String, strAttributeValue as String)

**Description**
Set the value of the attribute specified in strAttributeName for the element identified with strElementName. If the attribute is supported but has no value assigned, the empty string is returned. To find out the names of attributes supported by an element, use GetElementAttributeNames, or HasElementAttribute.

**Errors**
- 2001 The authentic range object or its related view object is no longer valid.
- 2005 Invalid element name was specified.
- Invalid attribute name was specified.
- Invalid attribute value was specified.

**Examples**
' ---------------------------------------------
' Scripting environment - VBScript
' Get and set element attributes
' ---------------------------------------------
Dim objRange
' we assume that the active document is open in authentic view mode
Set objRange = Application.ActiveDocument.AuthenticView.Selection

' first we find out all the elements below the beginning of the range
Dim arrElements
objRange.GetElementHierarchy arrElements

If IsArray(arrElements) Then
    If UBound(arrElements) >= 0 Then
        ' we use the top level element and find out its valid attributes
        Dim arrAttrs()
        objRange.GetElementAttributeNames arrElements(0), arrAttrs

        If UBound(arrAttrs) >= 0 Then
            ' we retrieve the current value of the first valid attribute
            Dim strAttrVal
            strAttrVal = objRange.GetElementAttributeValue(arrElements(0), arrAttrs(0))

            msgbox "current value of " & arrElements(0) & "//" & arrAttrs(0) & " is: " & strAttrVal

            ' we change this value and read it again
            strAttrVal = "Hello"
            objRange.SetElementAttributeValue arrElements(0), arrAttrs(0), strAttrVal

            strAttrVal = objRange.GetElementAttributeValue(arrElements(0), arrAttrs(0))

            msgbox "new value of " & arrElements(0) & "//" & arrAttrs(0) & " is: " & strAttrVal
        End If
    End If
End If

SetFromRange

See also

Method: SetFromRange(objSrcRange as AuthenticRange)

Description
Sets the range object to the same beginning and end positions as objSrcRange.

Errors
2001 One of the two range objects, is invalid.
2005 Null object was specified as source object.

SetVariableValue

Method: SetVariableValue(strName as string, strValue as string)

Return Value
Sets the value (second parameter) of the variable named in the first parameter.

Errors
2201 Invalid object.
2202 No context.
Text

See also

**Property:** Text as String

Description
Set or get the textual content selected by the range object.

The number of characters retrieved are not necessarily identical, as there are text cursor positions between the beginning and end of the selected range. Most document elements support an end cursor position different to the beginning cursor position of the following element. Drop-down lists maintain only one cursor position, but can select strings of any length. In the case of radio buttons and check boxes, the text property value holds the string of the corresponding XML element.

If the range selects more then one element, the text is the concatenation of the single texts. XML entities are expanded so that '&' is expected as '&amp;'.

Setting the text to the empty string, does not delete any XML elements. Use Cut, Delete or PerformAction instead.

Errors
2001 The authentic range object or its related view object is no longer valid.
2005 Invalid address for a return parameter was specified.

3.2.6 AuthenticView

See also

<table>
<thead>
<tr>
<th>Properties</th>
<th>Methods</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Goto</td>
<td>OnBeforeCopy</td>
</tr>
<tr>
<td>AsXMLString</td>
<td>IsRedoEnabled</td>
<td>OnBeforeCut</td>
</tr>
<tr>
<td>DocumentBegin</td>
<td>IsUndoEnabled</td>
<td>OnBeforeDelete</td>
</tr>
<tr>
<td>DocumentEnd</td>
<td>Print</td>
<td>OnBeforeDrop</td>
</tr>
<tr>
<td>Event</td>
<td>Redo</td>
<td>OnBeforePaste</td>
</tr>
<tr>
<td>MarkupVisibility</td>
<td>Undo</td>
<td>OnDragOver</td>
</tr>
<tr>
<td>Parent</td>
<td>UpdateXMLInstanceEntities</td>
<td>OnKeyboardEvent</td>
</tr>
<tr>
<td>Selection</td>
<td></td>
<td>OnMouseEvent</td>
</tr>
<tr>
<td>XMLDataRoot</td>
<td></td>
<td>OnSelectionChanged</td>
</tr>
<tr>
<td>WholeDocument</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description
AuthenticView and its child objects AuthenticRange and AuthenticDataTransfer provide you with an interface for Authentic View, which allow easy and consistent modification of document
contents. These interfaces replace the following interfaces which are marked now as **obsolete**:

- **OldAuthenticView** (old name was **DocEditView**)
- **AuthenticSelection** (old name was **DocEditSelection**, superseded by **AuthenticRange**)
- **AuthenticEvent** (old name was **DocEditEvent**)

**AuthenticView** gives you easy access to specific features such as printing, the multi-level undo buffer, and the current cursor selection, or position.

**AuthenticView** uses objects of type **AuthenticRange** to make navigation inside the document straight-forward, and to allow for the flexible selection of logical text elements. Use the properties **DocumentBegin**, **DocumentEnd**, or **WholeDocument** for simple selections, while using the **Goto** method for more complex selections. To navigate relative to a given document range, see the methods and properties of the **AuthenticRange** object.

**Examples**

```
' ---------------------------------------
' XMLSpy scripting environment - VBScript
' secure access to authentic view object
' ---------------------------------------
Dim objDocument
Set objDocument = Application.ActiveDocument
If (Not objDocument Is Nothing) Then
    ' we have an active document, now check for view mode
    If (objDocument.CurrentViewMode <> spyViewAuthentic) Then
        If (Not objDocument.SwitchViewMode (spyViewAuthentic)) Then
            MsgBox "Active document does not support authentic view mode"
        Else
            ' now it is safe to access the authentic view object
            Dim objAuthenticView
            Set objAuthenticView = objDocument.AuthenticView
            ' now use the authentic view object
        End If
    Else
        ' now it is safe to access the authentic view object
        Dim objAuthenticView
        Set objAuthenticView = objDocument.AuthenticView
        ' now use the authentic view object
    End If
Else
    MsgBox "No document is open"
End If
```

**Events**

**OnBeforeCopy**

**See also**

**Event**: **OnBeforeCopy()** as **Boolean**

**Scripting environment - VBScript**:

```
Function On_AuthenticBeforeCopy()
    ' On_AuthenticBeforeCopy = False ' to disable operation
End Function
```
Scripting environment - JScript:

```javascript
function On_AuthenticBeforeCopy()
{
    // return false; /* to disable operation */
}
```

IDE Plugin:

```javascript
IXMLSpyPlugIn.OnEvent(21, ...) // nEventId = 21
```

Description
This event gets triggered before a copy operation gets performed on the document. Return `true` (or nothing) to allow copy operation. Return `false` to disable copying.

OnBeforeCut

See also

Event: OnBeforeCut() as Boolean

Scripting environment - VBScript:

```vbs
Function On_AuthenticBeforeCut()
    ' On_AuthenticBeforeCut = False ' to disable operation
End Function
```

Scripting environment - JScript:

```javascript
function On_AuthenticBeforeCut()
{
    // return false; /* to disable operation */
}
```

IDE Plugin:

```javascript
IXMLSpyPlugIn.OnEvent(20, ...) // nEventId = 20
```

Description
This event gets triggered before a cut operation gets performed on the document. Return `true` (or nothing) to allow cut operation. Return `false` to disable operation.

OnBeforeDelete

See also

Event: OnBeforeDelete() as Boolean

Scripting environment - VBScript:

```vbs
Function On_AuthenticBeforeDelete()
    ' On_AuthenticBeforeDelete = False ' to disable operation
End Function
```

Scripting environment - JScript:

```javascript
function On_AuthenticBeforeDelete()
{
    // return false; /* to disable operation */
```
1151

Application API

Interfaces

}
IDE Plugin:
IXMLSpyPlugIn.OnEvent (22, ...) / / nEv ent I d = 22
Description
This event gets triggered before a delete operation gets performed on the document. Return True
(or nothing) to allow delete operation. Return False to disable operation.
OnBeforeDrop
See also
Event: OnBeforeDrop (i_nXPos as Long, i_nYPos as Long, i_ipRange as
AuthenticRange, i_ipData as cancelBoolean
Scripting environment - VBScript:
Function On_AuthenticBeforeDrop(nXPos, nYPos, objRange, objData)
' On_AuthenticBeforeDrop = False ' to disable operation
End Function
Scripting environment - JScript:
function On_AuthenticBeforeDrop(nXPos, nYPos, objRange, objData)
{
// return false; /* to disable operation */
}
IDE Plugin:
IXMLSpyPlugIn.OnEvent (11, ...) / / nEv ent I d = 11
Description
This event gets triggered whenever a previously dragged object gets dropped inside the application
window. All event related information gets passed as parameters.
The first two parameters specify the mouse position at the time when the event occurred. The
parameter objRange passes a range object that selects the XML element below the mouse
position. The value of this parameter might be NULL. Be sure to check before you access the
range object. The parameter objData allows to access information about the object being
dragged.
Return False to cancle the drop operation. Return True (or nothing) to continue normal operation.
Examples
' ---------------------------------------------------------------------------' VB code snippet - connecting to object level events
' ---------------------------------------------------------------------------' access XMLSpy (without checking for any errors)
Dim objSpy As XMLSpyLib.Application
Set objSpy = GetObject("", "XMLSpy.Application")
' this is the event callback routine connected to the OnBeforeDrop
' event of object objView
Private Function objView_OnBeforeDrop(ByVal i_nXPos As Long, ByVal i_nYPos As

Altova XMLSpy 2019 Professional Edition

© 2019 Altova Gmb H


Long,

ByVal i_ipRange As IAuthenticRange,
ByVal i_ipData As IAuthenticDataTransfer) As Boolean

If (Not i_ipRange Is Nothing) Then
    MsgBox ("Dropping on content is prohibited");
    Return False;
Else
    Return True;
End If
End Function

' use VBA keyword WithEvents to connect to object-level event
Dim WithEvents objView As XMLSpyLib.AuthenticView
Set objView = objSpy.ActiveDocument.AuthenticView

' continue here with something useful ...
' and serve the windows message loop

OnBeforePaste

See also

Event: OnBeforePaste (objData as Variant, strType as String) as Boolean

Scripting environment - VBScript:
Function On_AuthenticBeforePaste (objData, strType)
    ' On_AuthenticBeforePaste = False ' to disable operation
End Function

Scripting environment - JScript:
function On_AuthenticBeforePaste (objData, strType)
{
    // return false; /* to disable operation */
}

IDE Plugin:
IXMLSpyPlugIn.OnEvent (19, ...)   // nEventId = 19

Description
This event gets triggered before a paste operation gets performed on the document. The parameter strType is one of "TEXT", "UNICODETEXT" or "IUNKNOWN". In the first two cases objData contains a string representation of the object that will be pasted. In the later case, objData contains a pointer to an IUnknown COM interface.

Return True (or nothing) to allow paste operation. Return False to disable operation.

OnBeforeSave

Event: OnBeforeSave (SaveAs flag) as Boolean

Description: OnBeforeSave gives the opportunity to e.g. warn the user about overwriting the
existing XML document, or to make the document read-only when specific circumstances are not met. The event will be fired before the file dialog is shown.

**OnDragOver**

**See also**

**Event:** OnDragOver (\(nXPos\) as Long, \(nYPos\) as Long, \(eMouseEvent\) as SPYMouseEvent, \(objRange\) as TAuthenticRange, \(objData\) as AuthenticDataTransfer) as Boolean

**Scripting environment - VBScript:**

```vbscript
Function On_AuthenticDragOver(nXPos, nYPos, eMouseEvent, objRange, objData)
    ' On_AuthenticDragOver = False  ' to disable operation
End Function
```

**Scripting environment - JScript:**

```javascript
function On_AuthenticDragOver(nXPos, nYPos, eMouseEvent, objRange, objData)
{
    // return false;  /* to disable operation */
}
```

**IDE Plugin:**

```vbnet
IXMLSpyPlugIn.OnEvent(10, ...) // nEventId = 10
```

**Description**

This event gets triggered whenever an object from within or outside of Authentic View gets dragged with the mouse over the application window. All event related information gets passed as parameters.

The first three parameters specify the mouse position, the mouse button status and the status of the virtual keys at the time when the event occurred. The parameter `objRange` passes a range object that selects the XML element below the mouse position. The value of this parameter might be `NULL`. Be sure to check before you access the range object. The parameter `objData` allows to access information about the object being dragged.

**Return** `False` to cancel the drag operation. Return `True` (or nothing) to continue normal operation.

**Examples**

```vbnet
' VB code snippet - connecting to object level events
' ---------------------------------------------------------------------
' access XMLSpy (without checking for any errors)
Dim objSpy As XMLSpyLib.Application
Set objSpy = GetObject("", "XMLSpy.Application")

' this is the event callback routine connected to the OnDragOver
' event of object objView
Private Function objView_OnDragOver(ByVal i_nXPos As Long, ByVal i_nYPos As Long,
                                   ByVal i_eMouseEvent As SPYMouseEvent, 
                                   ByVal i_eMouseEvent As SPYMouseEvent,
```

```vbnet
```
ByVal i_ipRange As IAuthenticRange,
ByVal i_ipData As IAuthenticDataTransfer)

As Boolean

If (((i_eMouseEvent And spyShiftKeyDownMask) <> 0) And
   (Not i_ipRange Is Nothing)) Then
   MsgBox ("Floating over element " &
     i_ipRange.FirstXMLData.Parent.Name);
   End If

   Return True;
End Function

' use VBA keyword WithEvents to connect to object-level event
Dim WithEvents objView As XMLSpyLib.AuthenticView
Set objView = objSpy.ActiveDocument.AuthenticView

' continue here with something useful ...
' and serve the windows message loop

OnKeyboardEvent

See also

Event: OnKeyboardEvent (eKeyEvent as SPYKeyEvent, nKeyCode as Long,
nVirtualKeyStatus as Long) as Boolean

Scripting environment - VBScript:
Function On_AuthenticKeyboardEvent(eKeyEvent, nKeyCode, nVirtualKeyStatus)
   ' On_AuthenticKeyboardEvent = True ' to cancel bubbling of event
   End Function

Scripting environment - JScript:
function On_AuthenticKeyboardEvent(eKeyEvent, nKeyCode, nVirtualKeyStatus)
{
   // return true; /* to cancel bubbling of event */
}

IDE Plugin:
IXMLSpyPlugIn.OnEvent(30, ...) // nEventId = 30

Description
This event gets triggered for WM_KEYDOWN, WM_KEYUP and WM_CHAR Windows messages.

The actual message type is available in the eKeyEvent parameter. The status of virtual keys is combined in the parameter nVirtualKeyStatus. Use the bit-masks defined in the enumeration datatype SPYVirtualKeyMask, to test for the different keys or their combinations.
**OnLoad Event:** `OnLoad()`

**Description:** `OnLoad` can be used e.g. to restrict some AuthenticView functionality, as shown in the example below:

```javascript
function On_AuthenticLoad( )
{
    // We are disabling all entry helpers in order to prevent user from manipulating XML tree
    AuthenticView.DisableElementEntryHelper();
    AuthenticView.DisableAttributeEntryHelper();

    // We are also disabling the markup buttons for the same purpose
    AuthenticView.SetToolbarButtonState( 'AuthenticMarkupSmall', authenticToolbarButtonDisabled );
    AuthenticView.SetToolbarButtonState( 'AuthenticMarkupLarge', authenticToolbarButtonDisabled );
    AuthenticView.SetToolbarButtonState( 'AuthenticMarkupMixed', authenticToolbarButtonDisabled );
}
```

In the example the status of the Markup Small, Markup Large, Markup Mixed toolbar buttons are manipulated with the help of button identifiers. See complete list.

**OnMouseEvent**

See also

**Event:** `OnMouseEvent(nXPos as Long, nYPos as Long, eMouseEvent as SPYMouseEvent, objRange as AuthenticRange)` as Boolean

**Scripting environment - VBScript:**

```vbscript
Function On_AuthenticMouseEvent( nXPos, nYPos, eMouseEvent, objRange )
    ' On_AuthenticMouseEvent = True  ' to cancel bubbling of event
End Function
```

**Scripting environment - JScript:**

```javascript
function On_AuthenticMouseEvent( nXPos, nYPos, eMouseEvent, objRange )
{
    // return true;  /* to cancel bubbling of event */
}
```

**IDE Plugin:**

```javascript
IXMLSpyPlugIn.OnEvent(31, ... // nEventId = 31
```

**Description**

This event gets triggered for every mouse movement and mouse button Windows message.

The actual message type and the mouse buttons status, is available in the `eMouseEvent` parameter. Use the bit-masks defined in the enumeration datatype `SPYMouseEvent` to test for the different messages, button status, and their combinations.
The parameter `objRange` identifies the part of the document found at the current mouse cursor position. The range objects always selects a complete tag of the document. (This might change in future versions, when a more precise positioning mechanism becomes available). If no selectable part of the document is found at the current position, the range object is `null`.

**OnSelectionChanged**

See also

**Event:** `OnSelectionChanged (objNewSelection as AuthenticRange)`

**Scripting environment - VBScript:**

Function `On_AuthenticSelectionChanged (objNewSelection)`
End Function

**Scripting environment - JScript:**

function `On_AuthenticSelectionChanged (objNewSelection)`
{
}

**IDE Plugin:**

`IXMLSpyPlugIn.OnEvent (23, ...) // nEventId = 23`

**Description**

This event gets triggered whenever the selection in the user interface changes.

**Examples**

```vbnet
' access XMLSpy (without checking for any errors)
Dim objSpy As XMLSpyLib.Application
Set objSpy = GetObject("", "XMLSpy.Application")

' this is the event callback routine connected to the OnSelectionChanged event of object objView
Private Sub objView_OnSelectionChanged (ByVal i_ipNewRange As XMLSpyLib.IAuthenticRange)
    MsgBox ("new selection: " & i_ipNewRange.Text)
End Sub

' use VBA keyword WithEvents to connect to object-level event
Dim WithEvents objView As XMLSpyLib.AuthenticView
Set objView = objSpy.ActiveDocument.AuthenticView

' continue here with something useful ...
' and serve the windows message loop
```

**OnToolBarButtonClick**

**Event:** `OnToolBarButtonClick (Button identifier)`

**Description:** `OnToolBarButtonClick` is fired when a toolbar button was clicked by user. The
parameter button identifier helps to determine which button was clicked. The list of predefined button identifiers is below:

- AuthenticPrint
- AuthenticPrintPreview
- AuthenticUndo
- AuthenticRedo
- AuthenticCut
- AuthenticCopy
- AuthenticPaste
- AuthenticClear
- AuthenticMarkupHide
- AuthenticMarkupLarge
- AuthenticMarkupMixed
- AuthenticMarkupSmall
- AuthenticValidate
- AuthenticChangeWorkingDBXMLCell
- AuthenticSave
- AuthenticSaveAs
- AuthenticReload
- AuthenticTableInsertRow
- AuthenticTableAppendRow
- AuthenticTableDeleteRow
- AuthenticTableInsertCol
- AuthenticTableAppendCol
- AuthenticTableDeleteCol
- AuthenticTableJoinCellRight
- AuthenticTableJoinCellLeft
- AuthenticTableJoinCellAbove
- AuthenticTableJoinCellBelow
- AuthenticTableSplitCellHorizontally
- AuthenticTableSplitCellVertically
- AuthenticTableAlignCellContentTop
- AuthenticTableCenterCellVertically
- AuthenticTableAlignCellContentBottom
- AuthenticTableAlignCellContentLeft
- AuthenticTableCenterCellContent
- AuthenticTableAlignCellContentRight
- AuthenticTableJustifyCellContent
- AuthenticTableInsertTable
- AuthenticTableDeleteTable
- AuthenticTableProperties
- AuthenticAppendRow
- AuthenticInsertRow
- AuthenticDuplicateRow
- AuthenticMoveRowUp
- AuthenticMoveRowDown
- AuthenticDeleteRow
- AuthenticDefineEntities
- AuthenticXMLSignature
For custom buttons the user might add his own identifiers. Please, note that the user must take care, as the identifiers are not checked for uniqueness. The same identifiers can be used to identify buttons in the Set/GetToolbarState() COM API calls. By adding code for different buttons, the user is in the position to completely redefine the AuthenticView toolbar behavior, adding own methods for table manipulation, etc.

**OnToolbarButtonExecuted**

*Event:* OnToolbarButtonExecuted (Button identifier)

*Description:* OnToolbarButtonClicked is fired when a toolbar button was clicked by user. The parameter button identifier helps to determine which button was clicked. See the list of predefined button identifiers.

OnToolbarButtonExecuted is fired after the toolbar action was executed. It is useful e.g. to add update code, as shown in the example below:

```javascript
// event fired when a toolbar button action was executed
function On_AuthenticToolbarButtonExecuted( varBtnIdentifier )
{
    // After whatever command user has executed - make sure to update toolbar button states
    UpdateOwnToolbarButtonStates();
}
```

In this case `UpdateOwnToolbarButtonStates` is a user function defined in the Global Declarations.

**OnUserAddedXMLNode**

*Event:* OnUserAddedXMLNode (XML node)

*Description:* OnUserAddedXMLNode will be fired when the user adds an XML node as a primary action. This happens in the situations, where the user clicks on

- auto-add hyperlinks (see example OnUserAddedXMLNode.sps)
- the Insert..., Insert After..., Insert Before... context menu items
- Append row, Insert row toolbar buttons
- Insert After..., Insert Before... actions in element entry helper (outside StyleVision)

The event doesn’t get fired on Duplicate row, or when the node was added externally (e.g. via COM API), or on Apply (e.g. Text State Icons), or when in XML table operations or in DB operations.

The event parameter is the XML node object, which was added giving the user an opportunity to manipulate the XML node added. An elaborate example for an event handler can be found in the OnUserAddedXMLNode.sps file.

**Application**

See also
**Property:** Application as Application (read-only)

**Description**
Accesses the XMLSpy application object.

**Errors**
- 2000 The authentic view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

**AsXMLString**

**See also**

**Property:** AsXMLString as String

**Description**
Returns or sets the document content as an XML string. Setting the content to a new value does not change the schema file or sps file in use. If the new XMLString does not match the actual schema file error 2011 gets returned.

**Errors**
- 2000 The authentic view object is no longer valid.
- 2011 AsXMLString was set to a value which is no valid XML for the current schema file.

**ContextMenu**

**Property:** ContextMenu() as ContextMenu

**Description**
The property ContextMenu gives access to customize the context menu. The best place to do it is in the event handler OnContextMenuActivated.

**Errors**
- 2000 Invalid object.
- 2005 Invalid parameter.

**CreateXMLNode**

**Method:** CreateXMLNode (nKind as SPYXMLDataKind) as XMLData

**Return Value**
The method returns the new XMLData object.

**Description**
To create a new XMLData object use the CreateXMLNode() method.

**Errors**
- 2000 Invalid object.
- 2012 Cannot create XML node.
DisableAttributeEntryHelper

**Method:** DisableAttributeEntryHelper()

**Description**
DisableAttributeEntryHelper() disables the attribute entry helper in XMLSpy, Authentic Desktop and Authentic Browser plug-in.

**Errors**
- 2000 Invalid object.

DisableElementEntryHelper

**Method:** DisableElementEntryHelper()

**Description**
DisableElementEntryHelper() disables the element entry helper in XMLSpy, Authentic Desktop and Authentic Browser plug-in.

**Errors**
- 2000 Invalid object.

DisableEntityEntryHelper

**Method:** DisableEntityEntryHelper()

**Description**
DisableEntityEntryHelper() disables the entity entry helper in XMLSpy, Authentic Desktop and Authentic Browser plug-in.

**Errors**
- 2000 Invalid object.

DocumentBegin

**See also**

**Property:** DocumentBegin as AuthenticRange (read-only)

**Description**
Retrieve a range object that points to the beginning of the document.

**Errors**
- 2000 The authentic view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

DocumentEnd

**See also**

**Property:** DocumentEnd as AuthenticRange (read-only)
Description
Retrieve a range object that points to the end of the document.

Errors
2000 The authentic view object is no longer valid.
2005 Invalid address for the return parameter was specified.

DoNotPerformStandardAction
Method: DoNotPerformStandardAction ()

Description
DoNotPerformStandardAction() serves as cancel bubble for macros, and stops further execution after macro has finished.

Errors
2000 Invalid object.

EvaluateXPath
Method: EvaluateXPath (XMLData as XMLData, strExpression as string) strValue as string

Return Value
The method returns a string

Description
EvaluateXPath() executes an XPath expressions with the given XML context node. The result is returned as string, in the case of a sequence it is a space-separated string.

Errors
2000 Invalid object.
2005 Invalid parameter.
2008 Internal error.
2013 XPath error.

Event
See also

Property: Event as AuthenticEvent (read-only)

Description
This property gives access to parameters of the last event in the same way as OldAuthenticView.event does. Since all events for the scripting environment and external clients are now available with parameters this Event property should only be used from within IDE-Plugins.

Errors
2000 The authentic view object is no longer valid.
2005 Invalid address for the return parameter was specified.
EventContext

Property: EventContext() as EventContext

Description
EventContext property gives access to the running macros context. See the EventContext interface description for more details.

Errors
2000  Invalid object.

GetToolbarButtonState

Method: GetToolbarButtonState (ButtonIdentifier as string) as AuthenticToolbarButtonState

Return Value
The method returns AuthenticToolbarButtonState

Description
Get/SetToolbarButtonState queries the status of a toolbar button, and lets the user disable or enable the button, identified via its button identifier (see list above). One usage is to disable toolbar buttons permanently. Another usage is to put SetToolbarButtonState in the OnSelectionChanged event handler, as toolbar buttons are updated regularly when the selection changes in the document.

Toolbar button states are given by the listed enumerations.

The default state means that the enable/disable of the button is governed by AuthenticView. When the user sets the button state to enable or disable, the button remains in that state as long as the user does not change it.

Errors
2000  Invalid object.
2005  Invalid parameter.
2008  Internal error.
2014  Invalid button identifier.

Goto

See also

Method: Goto (eKind as SPYAuthenticElementKind, nCount as Long, eFrom as SPYAuthenticDocumentPosition) as AuthenticRange

Description
Retrieve a range object that points to the beginning of the nCount element of type eKind. The start position is defined by the parameter eFrom. Use positive values for nCount to navigate to the document end. Use negative values to navigate towards the beginning of the document.

Errors
2000  The authentic view object is no longer valid.
2003 Target lies after end of document.
2004 Target lies before beginning of document.
2005 Invalid element kind specified.
The document position to start from is not one of \texttt{spyAuthenticDocumentBegin} or \texttt{spyAuthenticDocumentEnd}.
Invalid address for the return parameter was specified.

**Examples**

```
' -------------------------------
' Scripting environment - VBScript
' -------------------------------
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView
On Error Resume Next
Dim objRange
' goto beginning of first table in document
Set objRange = objAuthenticView.Goto (spyAuthenticTable, 1,
spyAuthenticDocumentBegin)
If (Err.number = 0) Then
  objRange.Select()
Else
  MsgBox "No table found in document"
End If
```

**IsRedoEnabled**

See also

**Property:** IsRedoEnabled as Boolean (read-only)

**Description**

True if redo steps are available and \textit{Redo} is possible.

**Errors**

2000 The authentic view object is no longer valid.
2005 Invalid address for the return parameter was specified.

**IsUndoEnabled**

See also

**Property:** IsUndoEnabled as Boolean (read-only)

**Description**

True if undo steps are available and \textit{Undo} is possible.

**Errors**

2000 The authentic view object is no longer valid.
2005 Invalid address for the return parameter was specified.
**MarkupVisibility**

*See also*

*Property:* **MarkupVisibility** as **SPYAuthenticMarkupVisibility**

**Description**
Set or get current visibility of markup.

**Errors**
- 2000  The authentic view object is no longer valid.
- 2005  Invalid enumeration value was specified.
- 2005  Invalid address for the return parameter was specified.

**Parent**

*See also*

*Property:* **Parent** as **Document** (read-only)

**Description**
Access the document shown in this view.

**Errors**
- 2000  The authentic view object is no longer valid.
- 2005  Invalid address for the return parameter was specified.

**Print**

*See also*

*Method:* **Print** *(bWithPreview as Boolean, bPromptUser as Boolean)*

**Description**
Print the document shown in this view. If **bWithPreview** is set to *True*, the print preview dialog pops up. If **bPromptUser** is set to *True*, the print dialog pops up. If both parameters are set to *False*, the document gets printed without further user interaction.

**Errors**
- 2000  The authentic view object is no longer valid.

**Redo**

*See also*

*Method:* **Redo()** as Boolean

**Description**
Redo the modification undone by the last undo command.
Errors
2000  The authentic view object is no longer valid.
2005  Invalid address for the return parameter was specified.

Selection
See also

Property: Selection as AuthenticRange

Description
Set or get current text selection in user interface.

Errors
2000  The authentic view object is no longer valid.
2002  No cursor selection is active.
2005  Invalid address for the return parameter was specified.

Examples
' ---------------------------------------
' Scripting environment - VBScript
' ---------------------------------------
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

' if we are the end of the document, re-start at the beginning
If (objAuthenticView.Selection.IsEqual(objAuthenticView.DocumentEnd)) Then
  objAuthenticView.Selection = objAuthenticView.DocumentBegin
Else
  objAuthenticView.Selection =
  objAuthenticView.Selection.GotoNextCursorPosition()
  ' or shorter:
  objAuthenticView.Selection.GotoNextCursorPosition().Select
End If

SetToolbarButtonState

Method: SetToolbarButtonState (ButtonIdentifier as string,
                            AuthenticToolbarButtonState state)

Description
Get/SetToolbarButtonState queries the status of a toolbar button, and lets the user disable or
enable the button, identified via its button identifier (see list above). One usage is to disable
toolbar buttons permanently. Another usage is to put SetToolbarButtonState in the
OnSelectionChanged event handler, as toolbar buttons are updated regularly when the selection
changes in the document.

Toolbar button states are given by the listed enumerations.

The default state means that the enable/disable of the button is governed by AuthenticView.
When the user sets the button state to enable or disable, the button remains in that state as long
as the user does not change it.
Errors
2000 Invalid object.
2008 Internal error.
2014 Invalid button identifier.

Undo
See also

Method: Undo() as Boolean

Description
Undo the last modification of the document from within this view.

Errors
2000 The authentic view object is no longer valid.
2005 Invalid address for the return parameter was specified.

UpdateXMLInstanceEntities
See also

Method: UpdateXMLInstanceEntities()

Description
Updates the internal representation of the declared entities, and refills the entry helper. In addition, the validator is reloaded, allowing the XML file to validate correctly. Please note that this may also cause schema files to be reloaded.

Errors
The method never returns an error.

Example
// -----------------------------------------
// Scripting environment - JavaScript
// -----------------------------------------
{
    var objDocType;
    objDocType = Application.ActiveDocument.DocEditView.XMLRoot.GetFirstChild(10);
    if(objDocType)
    {
        var objEntity = Application.ActiveDocument.CreateChild(14);
        objEntity.Name = "child";
        objEntity.TextValue = "SYSTEM \"child.xml\"";
        objDocType.AppendChild(objEntity);
    }
    Application.ActiveDocument.AuthenticView.UpdateXMLInstanceEntities();
}
}  

**WholeDocument**  
See also

*Property:* WholeDocument as **AuthenticRange** (read-only)

**Description**  
Retrieve a range object that selects the whole document.

**Errors**  
2000  The authentic view object is no longer valid.  
2005  Invalid address for the return parameter was specified.

**XMLDataRoot**  
See also

*Property:* XMLDataRoot as **XMLData** (read-only)

**Description**  
Returns or sets the top-level XMLData element of the current document. This element typically describes the document structure and would be of kind spyXMLDataXMLDocStruct, spyXMLDataXMLEntityDocStruct or spyXMLDataDTDDocStruct.

**Errors**  
2000  The authentic view object is no longer valid.  
2005  Invalid address for the return parameter was specified.

### 3.2.7 CodeGeneratorDlg

See also

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

**Properties and Methods**

Standard automation properties

*Application*  
*Parent*  

Programming language selection properties

*ProgrammingLanguage*  
*TemplateFileName*  

Settings for C++ code

*CPPSettings_DOMType*  
*CPPSettings_LibraryType*  
*CPPSettings_UseMFC*  
*CPPSettings_GenerateVC6ProjectFile*  
*CPPSettings_GenerateVSProjectFile*  

Settings for C# code
CSharpSettings_ProjectType

Dialog handling for above code generation properties
PropertySheetDialogAction

Output path selection properties
OutputPath
OutputPathDialogAction

Presentation of result
OutputResultDialogAction

Description
Use this object to configure the generation of program code for schema files. The method GenerateProgramCode expects a CodeGeneratorDlg as parameter to configure code generation as well as the associated user interactions.

Application

See also
Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Property: Application as Application (read-only)

Description
Access the XMLSpy application object.

Errors

2200  The object is no longer valid.
2201  Invalid address for the return parameter was specified.

CompatibilityMode (obsolete)

Property: CompatibilityMode as Boolean
Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description
Set to true to generate code compatible to XMLSpy 2005R3. Set to false to use newly added code-generation features.

Errors

2200  The object is no longer valid.
2201  Invalid action passed as parameter or an invalid address was specified for the return parameter.
CPPSettings_DOMType

Property: CPPSettings_DOMType as SPYDOMType
Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description
Defines one of the settings that configure generation of C++ code.

Errors
2200  The object is no longer valid.
2201  Invalid action passed as parameter or an invalid address was specified for the return parameter.

CPPSettings_GenerateVC6ProjectFile

Property: CPPSettings_GenerateVC6ProjectFile as Boolean
Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description
Defines one of the settings that configure generation of C++ code.

Errors
2200  The object is no longer valid.
2201  Invalid action passed as parameter or an invalid address was specified for the return parameter.

CPPSettings_GenerateGCCMakefile

Property: CPPSettings_GenerateGCCMakefile as Boolean
Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description
Creates makefiles to compile the generated code under Linux with GCC.

Errors
2200  The object is no longer valid.
2201  Invalid action passed as parameter or an invalid address was specified for the return parameter.

CPPSettings_GenerateVSProjectFile

Property: CSharpSettings_GenerateVSProjectFile as SPYProjectType
Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.
**Description**
Defines one of the settings that configure generation of C++ code. Only
spyVisualStudio2005Project (=4) and spyVisualStudio2008Project (=5) and
spyVisualStudio2010Project (=6) are valid project types.

**Errors**
2200  The object is no longer valid.
2201  Invalid action passed as parameter or an invalid address was specified for
the return parameter.

**CPPSettings_LibraryType**

*Property:*  CPPSettings_LibraryType as SPYLibType
Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other
version.

**Description**
Defines one of the settings that configure generation of C++ code.

**Errors**
2200  The object is no longer valid.
2201  Invalid action passed as parameter or an invalid address was specified for
the return parameter.

**CPPSettings_UseMFC**

*Property:*  CPPSettings_UseMFC as Boolean
Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other
version.

**Description**
Defines one of the settings that configure generation of C++ code.

**Errors**
2200  The object is no longer valid.
2201  Invalid action passed as parameter or an invalid address was specified for
the return parameter.

**CSharpSettings_ProjectType**

*Property:*  CSharpSettings_ProjectType as SPYProjectType
Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other
version.

**Description**
Defines the only setting to configure generation of C# code.

**Errors**
2200  The object is no longer valid.
2201  Invalid action passed as parameter or an invalid address was specified for
the return parameter.
OutputPath

Property: OutputPath as String
Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description
Selects the base directory for all generated code.

Errors
2200  The object is no longer valid.
2201  Invalid address for the return parameter was specified.

OutputPathDialogAction

Property: OutputPathDialogAction as SPYDialogAction
Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description
Defines how the sub-dialog for selecting the code generation output path gets handled. Set this value to spyDialogUserInput(2) to show the dialog with the current value of the OutputPath property as default. Use spyDialogOK(0) to hide the dialog from the user.

Errors
2200  The object is no longer valid.
2201  Invalid action passed as parameter or an invalid address was specified for the return parameter.

OutputResultDialogAction

Property: OutputResultDialogAction as SPYDialogAction
Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description
Defines how the sub-dialog that asks to show the result of the code generation process gets handled. Set this value to spyDialogUserInput(2) to show the dialog. Use spyDialogOK(0) to hide the dialog from the user.

Errors
2200  The object is no longer valid.
2201  Invalid action passed as parameter or an invalid address was specified for the return parameter.

Parent

See also
Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Property: Parent as Dialogs (read-only)
**Description**
Access the parent of the object.

**Errors**
- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

**ProgrammingLanguage**

PROPERTY: `ProgrammingLanguage` as `ProgrammingLanguage`

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

**Description**
Selects the output language for the code to be generated.

CAUTION: Setting this property to one of C++, C# or Java, changes the property `TemplateFileName` to the appropriate template file delivered with XMLSpy as well. If you want to generate C++, C# or Java code based on your own templates, set first the programming language and then select your template file.

**Errors**
- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

**PropertySheetDialogAction**

PROPERTY: `PropertySheetDialogAction` as `SPYDialogAction`

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

**Description**
Defines how the sub-dialog that configures the code generation process gets handled. Set this value to `spyDialogUserInput(2)` to show the dialog with the current values as defaults. Use `spyDialogOK(0)` to hide the dialog from the user.

**Errors**
- 2200 The object is no longer valid.
- 2201 Invalid action passed as parameter or an invalid address was specified for the return parameter.

**TemplateFileName**

PROPERTY: `TemplateFileName` as `String`

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

**Description**
Selects the code generation template file. XMLSpy comes with template files for C++, C# or Java in the SPL folder of your installation directory.

Setting this property to one of the code generation template files of your XMLSpy installation
automatically sets the ProgrammingLanguage property to its appropriate value.

Errors
- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

3.2.8 DatabaseConnection

See also

Properties for import and export
- File
- ADOConnection
- ODBCConnection

Properties for import only
- DatabaseKind
- SQLSelect
- AsAttributes
- ExcludeKeys
- IncludeEmptyElements
- NumberDateTimeFormat
- NullReplacement
- CommentIncluded

Properties for export only
- CreateMissingTables
- CreateNew
- TextFieldLen
- DatabaseSchema

Properties for XML Schema from DB Structure generation
- PrimaryKeys
- ForeignKeys
- UniqueKeys
- SchemaExtensionType
- SchemaFormat
- ImportColumnsType

Description
DatabaseConnection specifies the parameters for the database connection.

Please note that the properties of the DatabaseConnection interface are referring to the settings of the import and export dialogs of XMLSpy.

ADOConnection

See also

Property: ADOConnection as String

Description
The property `ADOConnection` contains a connection string. Either use this property or `ODBCConnection` or `File` to refer to a database.

**Errors**
No error codes are returned.

**Example**

```vba
Dim objSpyConn As DatabaseConnection
Set objSpyConn = objSpy.GetDatabaseSettings

Dim objADO As DataLinks
Set objADO = CreateObject("DataLinks")

If Not (objADO Is Nothing) Then
    Dim objConn As Connection
    Set objConn = objADO.PromptNew
    objSpyConn.ADOConnection = objConn.ConnectionString
End If
```

**AsAttributes**

**See also**

*Property:* `AsAttributes` as Boolean

**Description**
Set `AsAttributes` to true if you want to initialize all import fields to be imported as attributes. Default is false and will initialize all fields to be imported as elements. This property is used only in calls to `Application.GetDatabaseImportElementList`.

**Errors**
No error codes are returned.

**CommentIncluded**

**See also**

*Property:* `CommentIncluded` as Boolean

**Description**
This property tells whether additional comments are added to the generated XML. Default is true. This property is used only when importing from databases.

**Errors**
No error codes are returned.

**CreateMissingTables**

**See also**

*Property:* `CreateMissingTables` as Boolean
Description
If CreateMissingTables is true, tables which are not already defined in the export database will be created during export. Default is true. This property is used only when exporting to databases.

Errors
No error codes are returned.

CreateNew
See also

Property: CreateNew as Boolean

Description
Set CreateNew true if you want to create a new database on export. Any existing database will be overwritten. See also DatabaseConnection.File. Default is false. This property is used only when exporting to databases.

Errors
No error codes are returned.

DatabaseKind
See also

Property: DatabaseKind as SPYDatabaseKind

Description
Select the kind of database that gets access. The default value is spyDB_Unspecified(7) and is sufficient in most cases. This property is used only when importing from databases.

Errors
No error codes are returned.

DatabaseSchema
See also

Property: DatabaseSchema as String

Description
This property specifies the Schema used for export in Schema aware databases. Default is "". This property is used only when exporting to databases.

Errors
No error codes are returned.

ExcludeKeys
See also
**Property:** ExcludeKeys as Boolean

**Description**
Set ExcludeKeys to true if you want to exclude all key columns from the import data. Default is false. This property is used only when importing from databases.

**Errors**
No error codes are returned.

**File**

**See also**

**Property:** File as String

**Description**
The property File sets the path for the database during export or import. This property can only be used in conjunction with a Microsoft Access database. Either use this property or ODBCConnection or ADOConnection to refer to the database.

See also Import and Export.

**Errors**
No error codes are returned.

**ForeignKeys**

**See also**

**Property:** ForeignKeys as Boolean

**Description**
Specifies whether the Foreign Keys constraint is created or not. Default is true. This property is used only when creating a XML Schema from a DB structure.

**Errors**
No error codes are returned.

**ImportColumnsType**

**See also**

**Property:** ImportColumnsType as SPYImportColumnsType

**Description**
Defines if column information from the DB is saved as element or attribute in the XML Schema. Default is as element. This property is used only when creating a XML Schema from a DB structure.

**Errors**
No error codes are returned.
IncludeEmptyElements

See also

Property: IncludeEmptyElements as Boolean

Description
Set IncludeEmptyElements to false if you want to exclude all empty elements. Default is true. This property is used only when importing from databases.

Errors
No error codes are returned.

NullReplacement

See also

Property: NullReplacement as String

Description
This property contains the text value that is used during import for empty elements (null values). Default is "". This property is used only when importing from databases.

Errors
No error codes are returned.

NumberDateTimeFormat

See also

Property: NumberDateTimeFormat as SPYNumberDateTimeFormat

Description
The property NumberDateTimeFormat sets the format of numbers and date- and time-values. Default is spySystemLocale. This property is used only when importing from databases.

Errors
No error codes are returned.

ODBCConnection

See also

Property: ODBCConnection as String

Description
The property ODBCConnection contains a ODBC connection string. Either use this property or ADOConnection or File to refer to a database.

Errors
No error codes are returned.
PrimaryKeys
See also

*Property*: `PrimaryKeys` as `Boolean`

**Description**
Specifies whether the Primary Keys constraint is created or not. Default is true. This property is used only when creating a XML Schema from a DB structure.

**Errors**
No error codes are returned.

SchemaExtensionType
See also

*Property*: `SchemaExtensionType` as `SPYSchemaExtensionType`

**Description**
Defines the Schema extension type used during the Schema generation. This property is used only when creating a XML Schema from a DB structure.

See also [Create XML Schema from DB Structure](#).

**Errors**
No error codes are returned.

SchemaFormat
See also

*Property*: `SchemaFormat` as `SPYSchemaFormat`

**Description**
Defines the Schema format used during the Schema generation. This property is used only when creating a XML Schema from a DB structure.

See also [Create XML Schema from DB Structure](#).

**Errors**
No error codes are returned.

SQLSelect
See also

*Property*: `SQLSelect` as `String`

**Description**
The SQL query for the import is stored in the property `SQLSelect`. This property is used only when importing from databases. See also [Import and Export](#).
Errors
No error codes are returned.

**TextFieldLen**

See also

*Property:* `TextFieldLen` as long

**Description**
The property `TextFieldLen` sets the length for created text fields during the export. Default is 255. This property is used only when exporting to databases.

Errors
No error codes are returned.

**UniqueKeys**

See also

*Property:* `UniqueKeys` as Boolean

**Description**
Specifies whether the Unique Keys constraint is created or not. Default is true. This property is used only when creating an XML Schema from a DB structure.

Errors
No error codes are returned.

### 3.2.9 Dialogs

See also

**Properties and Methods**

Standard automation properties
- `Application`
- `Parent`

Various dialog objects
- `CodeGeneratorDlg`
- `FileSelectionDlg`
- `SchemaDocumentationDlg`
- `GenerateSampleXMLDlg`
- `DTDSchemaGeneratorDlg`
- `FindInFilesDlg`

**Description**
The Dialogs object provides access to different built-in dialogs of XMLSpy. These dialog objects allow to initialize the fields of user dialogs before they get presented to the user or allow to simulate complete user input by your program.
**Application**

See also

*Property:* Application as Application (read-only)

**Description**
Access the XMLSpy application object.

**Errors**
2300 The object is no longer valid.
2301 Invalid address for the return parameter was specified.

**CodeGenDlg**

See also
Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

*Property:* CodeGeneratorDlg as CodeGeneratorDlg (read-only)

**Description**
Get a new instance of a code generation dialog object. You will need this object to pass the necessary parameters to the code generation methods. Initial values are taken from last usage of the code generation dialog.

**Errors**
2300 The Dialogs object or one of its parents is no longer valid.
2301 Invalid address for the return parameter was specified.

**FileSelectionDlg**

See also

*Property:* FileSelectionDlg as FileSelectionDlg (read-only)

**Description**
Get a new instance of a file selection dialog object.

File selection dialog objects are passed to you with the some events that signal opening or saving of documents and projects.

**Errors**
2300 The Dialogs object or one of its parents is no longer valid.
2301 Invalid address for the return parameter was specified.

**Parent**

See also

*Property:* Parent as Application (read-only)
Description
Access the XMLSpy application object.

Errors
2300  The object is no longer valid.
2301  Invalid address for the return parameter was specified.

SchemaDocumentationDlg
See also

Property: SchemaDocumentationDlg as SchemaDocumentationDlg (read-only)

Description

Errors
2300  The Dialogs object or one of its parents is no longer valid.
2301  Invalid address for the return parameter was specified.

GenerateSampleXMLDlg
See also

Property: GenerateSampleXMLDlg as GenerateSampleXMLDlg (read-only)

Description
Get a new instance of a dialog object that parameterizes generation of a sample XML based on a W3C schema or DTD. See GenerateSampleXML for its usage.

Errors
2300  The Dialogs object or one of its parents is no longer valid.
2301  Invalid address for the return parameter was specified.

DTDSchemaGeneratorDlg
See also

Property: DTDSchemaGeneratorDlg as DTDSchemaGeneratorDlg (read-only)

Description
Get a new instance of a dialog object that parameterizes generation of a schema or DTD. See Document.GenerateDTDOrSchemaEx for its usage.

Errors
2300  The Dialogs object or one of its parents is no longer valid.
2301  Invalid address for the return parameter was specified.

FindInFilesDlg
See also

Property: FindInFilesDlg as FindInFilesDlg (read-only)
**Description**
Get a new instance of a dialog object that parameterizes the search (or replacement) of strings in files. See Application.FindInFiles for its usage.

**Errors**
- **2300** The Dialogs object or one of its parents is no longer valid.
- **2301** Invalid address for the return parameter was specified.

### 3.2.10 Document

**See also**

**Properties and Methods**

Standard automation properties
- Application
- Parent

Various document properties and methods
- SetActiveDocument
- Encoding
- SetEncoding (obsolete)
- Suggestions

XML validation
- IsValid
- SetExternalIsValid

Document conversion and transformation
- AssignDTD
- AssignSchema
- AssignXSL
- AssignXSLFO
- ConvertDTDOrSchema
- ConvertDTDOrSchemaEx
- GenerateDTDOrSchema
- GenerateDTDOrSchemaEx
- FlattenDTDOrSchema
- CreateSchemaDiagram
- ExecuteXQuery
- TransformXSL
- TransformXSLEx
- TransformXSLFO
- GenerateProgramCode (Enterprise Edition only)
- GenerateSchemaDocumentation
- GenerateSampleXML
- ConvertToWSDL20

Document export
- GetExportElementList
- ExportToText
- ExportToDatabase
CreateDBStructureFromXMLSchema
GetDBStructureList

File saving and naming
FullName
Name
Path
GetPathName (obsolete)
SetPathName (obsolete)
Title
IsModified
Saved
SaveAs
Save
SaveInString
SaveToURL
Close

View access
CurrentViewMode
SwitchViewMode
AuthenticView
GridView
DocEditView (obsolete)

Access to XMLData
RootElement
DataRoot
CreateChild
UpdateViews
StartChanges
EndChanges
UpdateXMLData

Description
Document objects represent XML documents opened in XMLSpy.

Use one of the following properties to access documents that are already open XMLSpy:
Application.ActiveDocument
Application.Documents

Use one of the following methods to open a new document in XMLSpy:
Documents.OpenFile
Documents.OpenURL
Documents.OpenURLDialog
Documents.NewFile
Documents.NewFileFromText
SpyProjectItem.Open
Application.ImportFromDatabase
Application.ImportFromSchema
Application.ImportFromText
Application.ImportFromWord
Document.ConvertDTDOrSchema
Events

**OnBeforeSaveDocument**

See also

**Event:** `OnBeforeSaveDocument(objDocument as Document, objDialog as FileSelectionDlg)

**XMLSpy scripting environment - VBScript:**

```vbnet
Function On_BeforeSaveDocument(objDocument, objDialog)
    End Function

' old handler - now obsolete
' return string to save to new file name
' return empty string to cancel save operation
' return nothing to save to original name
Function On_SaveDocument(objDocument, strFilePath)
End Function
```

**XMLSpy scripting environment - JScript:**

```javascript
function On_BeforeSaveDocument(objDocument, objDialog) {
}

// old handler - now obsolete
// return string to save to new file name
// return empty string to cancel save operation
// return nothing to save to original name
function On_SaveDocument(objDocument, strFilePath) {
}
```

**XMLSpy IDE Plugin:**

```csharp
IXMLSpyPlugIn.OnEvent(27, ...) // nEvent Id = 27
```

**Description**

This event gets fired on any attempt to save a document. The file selection dialog object is initialized with the name chosen for the document file. You can modify this selection. To continue saving the document leave the `FileSelectionDlg.DialogAction` property of `io_objDialog` at its default value `spyDialogOK`. To abort saving of the document set this property to `spyDialogCancel`.

**OnBeforeCloseDocument**

See also

**Event:** `OnBeforeCloseDocument(objDocument as Document) as Boolean`
**XMLSpy scripting environment - VBScript:**

Function `On_BeforeCloseDocument(objDocument)`

' On_BeforeCloseDocument = False ' to prohibit closing of document
End Function

**XMLSpy scripting environment - JScript:**

function `On_BeforeCloseDocument(objDocument)`

{  
  // return false; /* to prohibit closing of document */
}

**XMLSpy IDE Plugin:**

IXMLSpyPlugIn.OnEvent(28, ...) // nEventId = 28

**Description**

This event gets fired on any attempt to close a document. To prevent the document from being closed return false.

**OnBeforeValidate**

**See also**

**Event:** `OnBeforeValidate(objDocument as Document, bOnLoading as Boolean, bOnCommand as Boolean)` as Boolean

**XMLSpy scripting environment - VBScript:**

Function `On_BeforeValidate(objDocument, bOnLoading, bOnCommand)`

On_BeforeValidate = bCancelDefaultValidation 'set by the script if necessary
End Function

**XMLSpy scripting environment - JScript:**

function `On_BeforeValidate(objDocument, bOnLoading, bOnCommand)`

{  
  return bCancelDefaultValidation //set by the script if necessary
}

**XMLSpy IDE Plugin:**

IXMLSpyPlugIn.OnEvent(32, ...) // nEventId = 32

**Description**

This event gets fired before the document is validated. It is possible to suppress the default validation by returning false from the event handler. In this case the script should also set the validation result using the `SetExternalsValid` method.

`bOnLoading` is true if the event is raised on the initial validation on loading the document.

`bOnCommand` is true whenever the user selected the Validate command from the Toolbar or menu.
Available with TypeLibrary version 1.5

**OnCloseDocument**

See also

**Event:** `OnCloseDocument(objDocument as Document)`

**XMLSpy scripting environment - VBScript:**
```
Function On_CloseDocument(objDocument)
End Function
```

**XMLSpy scripting environment - JScript:**
```
function On_CloseDocument(objDocument)
{
}
```

**XMLSpy IDE Plugin:**
```
IXMLSpyPlugIn.OnEvent(8, ...) // nEventId = 8
```

**Description**
This event gets fired as a result of closing a document. Do not modify the document from within this event.

**OnViewActivation**

See also

**Event:** `OnViewActivation(objDocument as Document, eViewMode as SPYViewModes, bActivated as Boolean)`

**XMLSpy scripting environment - VBScript:**
```
Function On_ViewActivation(objDocument, eViewMode, bActivated)
End Function
```

**XMLSpy scripting environment - JScript:**
```
function On_ViewActivation(objDocument, eViewMode, bActivated)
{
}
```

**XMLSpy IDE Plugin:**
```
IXMLSpyPlugIn.OnEvent(29, ...) // nEventId = 29
```

**Description**
This event gets fired whenever a view of a document becomes visible (i.e. becomes the active view) or invisible (i.e. another view becomes the active view or the document gets closed). However, the first view activation event after a document gets opened cannot be received, since there is no document object to get the event from. Use the `Application.OnDocumentOpened` event instead.
Application

See also

Property: Application as Application (read-only)

Description
Accesses the XMLSpy application object.

Errors
1400 The object is no longer valid.
1407 Invalid address for the return parameter was specified.

AssignDTD

See also

Method: AssignDTD(strDTDFile as String, bDialog as Boolean)

Description
The method places a reference to the DTD file "strDTDFile" into the document. Note that no error occurs if the file does not exist, or is not accessible. If bDialog is true XMLSpy presents a dialog to set the file.

Errors
1400 The object is no longer valid.
1409 You are not allowed to assign a DTD to the document.

AssignSchema

See also

Method: AssignSchema(strSchemaFile as String, bDialog as Boolean)

Description
The method places a reference to the schema file "strSchemaFile" into the document. Note that no error occurs if the file does not exist or is not accessible. If bDialog is true XMLSpy presents a dialog to set the file.

Errors
1400 The object is no longer valid.
1409 You are not allowed to assign a schema file to the document.

AssignXSL

See also

Method: AssignXSL(strXSLFile as String, bDialog as Boolean)

Description
The method places a reference to the XSL file "strXSLFile" into the document. Note that no error occurs if the file does not exist or is not accessible. If bDialog is true XMLSpy presents a dialog to set the file.
Errors
1400 The object is no longer valid.
1409 You are not allowed to assign an XSL file to the document.

AssignXSLFO
See also

Method: AssignXSLFO (strXSLFOFile as String, bDialog as Boolean)

Description
The method places a reference to the XSLFO file "strXSLFile" into the document. Note that no error occurs if the file does not exist or is not accessible. If bDialog is true XMLSpy presents a dialog to set the file.

Errors
1400 The object is no longer valid.
1409 You are not allowed to assign an XSL file to the document.

AsXMLString
See also

Property: AsXMLString as String

Description
This property can be used to get or set the document content.

Errors
1400 The document object is no longer valid.
1404 Cannot create XMLData object.
1407 View mode cannot be switched.

AuthenticView
See also

Method: AuthenticView as AuthenticView (read-only)

Description
Returns an object that gives access to properties and methods specific to Authentic view. The object returned is only valid if the current document is opened in Authentic view mode. The lifetime of an object ends with the next view switch. Any attempt to access objects or any of its children afterwards will result in an error indicating that the object is invalid.

AuthenticView and DocEditView both provide automation access to the Authentic view mode of XMLSpy. Functional overlap is intentional. A future version of Authentic View will include all functionality of DocEditView and its sub-objects, thereby making usage of DocEditView obsolete.

Errors
1400 The object is no longer valid.
1417 Document needs to be open in authentic view mode.

Examples
' -----------------------------------------------------------
' XMLSpy scripting environment - VBScript
' secure access to authentic view object
' -----------------------------------------------------------
Dim objDocument
Set objDocument = Application.ActiveDocument
If (Not objDocument Is Nothing) Then
    ' we have an active document, now check for view mode
    If (objDocument.CurrentViewMode <> spyViewAuthentic) Then
        If (Not objDocument.SwitchViewMode (spyViewAuthentic)) Then
            MsgBox "Active document does not support authentic view
            mode"
    Else
        ' now it is safe to access the authentic view object
        Dim objAuthenticView
        Set objAuthenticView = objDocument.AuthenticView
        ' now use the authentic view object
    End If
Else
    MsgBox "No document is open"
End If

Close

See also

Method: Close (bDiscardChanges as Boolean)

Description
To close the document call this method. If bDiscardChanges is true and the document is modified, the document will be closed but not saved.

Errors
  1400 The object is no longer valid.
  1401 Document needs to be saved first.

ConvertDTDOrSchema

See also

Method: ConvertDTDOrSchema (nFormat as SPYDTDSchemaFormat, nFrequentElements as SPYFrequentElements)

Parameters
nFormat
Sets the schema output format to DTD or W3C.

nFrequentElements
Create complex elements as elements or complex types.

**Description**  
ConvertDTDOrSchema takes an existing schema format and converts it into a different format. For a finer tuning of DTD/XSD conversion, use ConvertDTDOrSchemaEx.

**Errors**  
1400 The object is no longer valid.
1412 Error during conversion. In the case of DTD to DTD or XSD to XSD conversion, the following errors are returned: DTD to DTD conversion is not supported. Please use function FlattenDTDOrSchema instead and Schema to schema conversion is not supported. Please use function FlattenDTDOrSchema instead.

**ConvertDTDOrSchemaEx**  
See also

*Method:* ConvertDTDOrSchemaEx (nFormat as SPYDTDSchemaFormat, nFrequentElements as SPYFrequentElements, sOutputPath as String, nOutputPathDialogAction as SPYDialogAction)

**Parameters**  
nFormat
Sets the schema output format to DTD, or W3C.

nFrequentElements
Create complex elements as elements or complex types.

sOutputPath
The file path for the newly generated file.

nOutputPathDialogAction
Defines the dialog interaction for this call.

**Description**  
ConvertDTDOrSchemaEx takes an existing schema format and converts it into a different format.

**Errors**  
1400 The object is no longer valid.
1412 Error during conversion. In the case of DTD to DTD or XSD to XSD conversion, the following errors are returned: DTD to DTD conversion is not supported. Please use function FlattenDTDOrSchema instead and Schema to schema conversion is not supported. Please use function FlattenDTDOrSchema instead.
ConvertToWSDL20

**Method:** ConvertToWSDL20 *(sFilePath as String, bShowDialogs as Boolean)*

**Parameters**

**sFilePath**
This specifies the file name of the converted WSDL. In case the source WSDL includes files which also must be converted, then only the directory part of the given path is used and the file names are generated automatically.

**bShowDialogs**
Defines whether file/folder selection dialogs are shown.

**Description**
Converts the WSDL 1.1 document to a WSDL 2.0 file. It will also convert any referenced WSDL files that are referenced from within this document. Note that this functionality is limited to WSDL View only. See Document.CurrentViewMode and SPYViewModes.

**Errors**
1400 The document object is no longer valid.
1407 Invalid parameters have been passed or an empty file name has been specified as output target.
1417 The document is not opened in WSDL view, maybe it is not an `.wsdl` file.
1421 Feature is not available in this edition.
1433 WSDL 1.1 to WSDL 2.0 conversion failed.

CreateChild

See also

**Method:** CreateChild *(nKind as SPYXMLDataKind) as XMLData*  

**Return Value**
The method returns the new XMLData object.

**Description**
To create a new XMLData object use the CreateChild() method.

**Errors**
1400 The object is no longer valid.
1404 Cannot create XMLData object.
1407 Invalid address for the return parameter was specified.

CreateDBStructureFromXMLSchema

See also

**Method:** CreateDBStructureFromXMLSchema *(pDatabase as DatabaseConnection, pTables as ElementList, bDropTableWithExistingName as Boolean) as String*

**Description**
CreateDBStructureFromXMLSchema exports the given tables to the specified database.
function returns the SQL statements that were necessary to perform the changes.

See also GetDBStructureList.

Errors
1429 Database selection missing.
1430 Document export failed.

CreateSchemaDiagram
See also

Method: CreateSchemaDiagram (nKind as SPYSchemaDefKind, strName as String, strFile as String)

Return Value
None.

Description
The method creates a diagram of the schema type strName of kind nKind and saves the output file into strFile. Note that this functionality is limited to Schema View only. See Document.CurrentViewMode and SPYViewModes.

Errors
1400 The object is no longer valid.
1414 Failed to save diagram.
1415 Invalid schema definition type specified.

CurrentViewMode
See also

Method: CurrentViewMode as SPYViewModes

Description
The property holds the current view mode of the document. See also Document.SwitchViewMode.

Errors
1400 The object is no longer valid.
1407 Invalid address for the return parameter was specified.

DataRoot
See also

Property: DataRoot as XMLData (read-only)

Description
This property provides access to the document's first XMLData object of type
spyXMLDataElement. This is typically the root element for all document content data. See
XMLSpyDocument.RootElement to get the root element of the whole document including XML
prolog data. If the CurrentViewMode is not spyViewGrid or spyViewAuthentic an UpdateXMLData
may be necessary to get access to the latest XMLData.

Errors
1400 The document object is no longer valid.
1407 Invalid address for the return parameter was specified.

DocEditView

See also

Method: DocEditView as DocEditView

Description
Holds a reference to the current Authentic View object.

Errors
1400 The object is no longer valid.
1407 Invalid address for the return parameter was specified.
1417 Document needs to be open in authentic view mode.

Encoding

See also

Property: Encoding as String

Description
This property provides access to the document's encoding value. However, this property can only
be accessed when the document is opened in spyViewGrid, spyViewText or spyViewAuthentic. See
CurrentViewMode on how to detect that a document's actual view mode.

This property makes the method SetEncoding obsolete.

Possible values are, for example:

8859-1,
8859-2,
ASCII, ISO-646,
850,
1252,
1255,
SHIFT-JIS, MS-KANJI,
BIG5, FIVE,
UTF-7,
UTF-8,
UTF-16

Errors
1400 The document object is no longer valid.
Invalid address for the return parameter was specified.
Operation not supported in current view mode.

**EndChanges**

See also

*Method:* `EndChanges()`

**Description**

Use the method `EndChanges` to display all changes since the call to `Document.StartChanges`.

**Errors**

- 1400  The object is no longer valid.

**ExecuteXQuery**

See also

*Method:* `ExecuteXQuery (strXMLFileName as String)`

**Description**

Execute the XQuery statements contained in the document of the document object. Either an XQuery execution or an XQuery Update is performed depending on the file extension of the document. Use the XML file specified in the argument as the XML target document that the XQuery document processes.

- If the document has an XQuery file extension as defined in the Options dialog of XMLSpy, then an XQuery execution is performed. By default: `.xq`, `.xql`, and `.xquery` are set as XQuery file extensions in XMLSpy.
- If the document has an XQuery Update file extension as defined in the Options dialog of XMLSpy, then an XQuery Update action is performed. By default: `.xqu` is set as an XQuery Update file extension in XMLSpy.

If your XQuery script does not use an XML source, set the parameter `strXMLFileName` to an empty string.

**Errors**

- 1400  The document object is no longer valid.
- 1423  XQuery transformation error.
- 1424  Not all files required for operation could be loaded. Most likely, the file specified in `strXMLFileName` does not exist or is not valid.

**ExportToDatabase**

See also

*Method:* `ExportToDatabase (pFromChild as XMLData, pExportSettings as ExportSettings, pDatabase as DatabaseConnection)`
Description

ExportToDatabase exports the XML document starting with the element pFromChild. The parameter pExportSettings defines the behaviour of the export (see Application.GetExportSettings). The parameter pDatabase specifies the destination of the export (see Application.GetDatabaseSettings). UpdateXMLData() might be indirectly needed as you have to pass the XMLData as parameter to this function.

Errors

1400  The object is no longer valid.
1407  Invalid parameter or invalid address for the return parameter was specified.
1416  Error during export.
1429  Database selection missing.
1430  Document export failed.

Example

Dim objDoc As Document
Set objDoc = objSpy.ActiveDocument

'set the behaviour of the export with ExportSettings
Dim objExpSettings As ExportSettings
Set objExpSettings = objSpy.GetExportSettings

'set the destination with DatabaseConnection
Dim objDB As DatabaseConnection
Set objDB = objSpy.GetDatabaseSettings

objDB.CreateMissingTables = True
objDB.CreateNew = True
objDB.File = "C:\Export.mdb"

objDoc.ExportToDatabase objDoc.RootElement, objExpSettings, objDB
If Err.Number <> 0 Then
a = MsgBox("Error: " & (Err.Number - vbObjectError) & Chr(13) & "Description: " & Err.Description)
End If

ExportToText

See also

Method: ExportToText (pFromChild as XMLData, pExportSettings as ExportSettings, pTextSettings as TextImportExportSettings)

Description

ExportToText exports tabular information from the document starting at pFromChild into one or many text files. Columns of the resulting tables are generated in alphabetical order of the column header names. Use GetExportElementList to learn about the data that will be exported. The parameter pExpor t Settings defines the specifics for the export. Set the property ExportSettings.ElementList to the - possibly modified - list returned by GetExportElementList to avoid exporting all contained tables. The parameter pText Settings defines the options specific to text export and import. You need to set the
property `TextImportExportSettings.DestinationFolder` before you call `ExportToText`. `UpdateXMLData()` might be indirectly needed as you have to pass the `XMLData` as parameter to this function.

See also [Import and export of data](#).

### Errors

- **1400** The object is no longer valid.
- **1407** Invalid parameter or invalid address for the return parameter was specified.
- **1416** Error during export.
- **1430** Document export failed.

### Example

```
' ---------------------------------------------------------
' VBA client code fragment - export document to text files
' ---------------------------------------------------------
Dim objDoc As Document
Set objDoc = objSpy.ActiveDocument

Dim objExpSettings As ExportSettings
Set objExpSettings = objSpy.GetExportSettings
objExpSettings.ElementList = objDoc.GetExportElementList(
    objDoc.RootElement,
    objExpSettings)

Dim objTextExp As TextImportExportSettings
Set objTextExp = objSpy.GetTextExportSettings
objTextExp.HeaderRow = True
objTextExp.DestinationFolder = "C:\Exports"

On Error Resume Next
objDoc.ExportToText objDoc.RootElement, objExpSettings, objTextExp
If Err.Number <> 0 Then
    a = MsgBox("Error: " & (Err.Number - vbObjectError) & Chr(13) & "Description: " & Err.Description)
End If

FlattenDTDOrSchema
```

**Method**: `FlattenDTDOrSchema` *(sOutputPath as String, nOutputPathDialogAction as SPYDialogAction)*

**Parameters**

- **sOutputPath**
  The file path for the newly generated file.

- **nOutputPathDialogAction**
  Defines the dialog interaction for this call.

**Description**

`FlattenDTDOrSchema` takes an existing DTD or schema, generates a flattened file, and saves the
generated file at the specified location. In the case of DTDs, flattening removes parameter entities and produces a single DTD from a collection of modules; sections marked IGNORE are suppressed and unused parameter entities are deleted. When an XML Schema is flattened, (i) the components of all included schemas are added as global components of the active schema, and (ii) included schemas are deleted.

**Errors**
- 1400 The object is no longer valid.
- 1412 Error during conversion.

**FullName**

**See also**

**Property:** `FullName` as String

**Description**
This property can be used to get or set the full file name - including the path - to where the document gets saved. The validity of the name is not verified before the next save operation.

This property makes the methods `GetPathName` and `SetPathName` obsolete.

**Errors**
- 1400 The document object is no longer valid.
- 1402 Empty string has been specified as full file name.

**GenerateDTDOrSchema**

**See also**

**Method:** `GenerateDTDOrSchema (nFormat as SPYDTDSchemaFormat, nValuesList as integer, nDetection as SPYTypeDetection, nFrequentElements as SPYFrequentElements)`

**Parameters**
- **nFormat**
  Sets the schema output format to DTD, or W3C.

- **nValuesList**
  Generate not more than this amount of enumeration-facets per type. Set to -1 for unlimited.

- **nDetection**
  Specifies granularity of simple type detection.

- **nFrequentElements**
  Shall the types for all elements be defined as global? Use that value `spyGlobalComplexType` to define them on global scope. Otherwise, use the value `spyGlobalElements`.

**Description**
Use this method to automatically generate a DTD or schema for the current XML document. For a finer tuning of DTD / schema generation, use `GenerateDTDOrSchemaEx`. 
Note that this functionality is not available in ZIP View only. See `Document.CurrentViewMode` and `SPYViewModes`.

**Errors**
- 1400 The object is no longer valid.
- 1407 Invalid parameter or invalid address for the return parameter was specified.

**GenerateDTDOrSchemaEx**

**See also**

**Method:** `GenerateDTDOrSchemaEx (objDlg as DTDSchemaGeneratorDlg) as Document`

**Description**
Use this method to automatically generate a DTD or schema for the current XML document. A `DTDSchemaGeneratorDlg` object is used to pass information to the schema/DTD generator. The generation process can be configured to allow user interaction or run without further user input. Note that this functionality is not available in ZIP View only. See `Document.CurrentViewMode` and `SPYViewModes`.

**Errors**
- 1400 The object is no longer valid.
- 1407 Invalid parameter or invalid address for the return parameter was specified.

**GenerateProgramCode**

**Method:** `GenerateProgramCode (objDlg as CodeGeneratorDlg)`

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

**Description**
Generate Java, C++ or C# class files from the XML Schema definitions in your document. A `CodeGeneratorDlg` object is used to pass information to the code generator. The generation process can be configured to allow user interaction or run without further user input.

**Errors**
- 1400 The document object is no longer valid.
- 1407 An empty file name has been specified.
- 1421 Feature not available in this edition

**GenerateSampleXML**

**Method:** `GenerateSampleXML (objDlg as GenerateSampleXMLDlg) as Document`

**Description**
Generates a sample XML if the document is a schema or DTD. Use `Dialogs.GenerateSampleXMLDlg` to get an initialized set of options.

Available with TypeLibrary version 1.5

**Errors**
- 1400 The document object is no longer valid.
**GenerateSchemaDocumentation**

*Method:* `GenerateSchemaDocumentation(objDlg as SchemaDocumentationDlg)`

*Description*
Generate documentation for a schema definition file in HTML, MS-Word, or RTF format. The parameter `objDlg` is used to parameterize the generation process. Use `Dialogs.SchemaDocumentationDlg` to get an initialized set of options. As a minimum, you will need to set the property `SchemaDocumentationDlg.OutputFile` before starting the generation process. Note that this functionality is limited to Schema View only. See `Document.CurrentViewMode` and `SPYViewModes`.

*Errors*
- 1400: The document object is no longer valid.
- 1407: Invalid parameters have been passed or an empty file name has been specified as output target.
- 1417: The document is not opened in schema view, maybe it is not an `.xsd` file.
- 1421: Feature is not available in this edition.
- 1422: Error during generation

**GetDBStructureList**

*See also*

*Method:* `GetDBStructureList(pDatabase as DatabaseConnection) as ElementList`

*Description*
`GetDBStructureList` creates a collection of elements from the Schema document for which tables in the specified database are created. The function returns a collection of `ElementListItem` where the properties `ElementListItem.Name` contain the names of the tables.

See also `CreateDBStructureFromXMLSchema`.

*Errors*
- 1400: The object is no longer valid.
- 1427: Failed creating parser for the specified XML.
- 1428: Export of element list failed.
- 1429: Database selection missing.

**GetExportElementList**

*See also*

*Method:* `GetExportElementList(pFromChild as XMLData, pExportSettings as ExportSettings) as ElementList`

*Description*
GetExportElementList creates a collection of elements to export from the document, depending on the settings in pExportSettings and starting from the element pFromChild. The function returns a collection of ElementListItems where the properties ElementListItem.Name contain the names of the tables that can be exported from the document. The property ElementListItem.FieldCount contains the number of columns in the table. The property ElementListItem.RecordCount contains the number of records in the table. The property ElementListItem.ElementKind is unused. UpdateXMLData() might be indirectly needed as you have to pass the XMLData as parameter to this function.

See also Import and export of data.

Errors
- 1400 The object is no longer valid.
- 1407 Invalid parameter or invalid address for the return parameter was specified.
- 1427 Failed creating parser for the specified XML.
- 1428 Export of element list failed.

GetPathName (obsolete)

<table>
<thead>
<tr>
<th>Superseded by</th>
<th>Document.FullName</th>
</tr>
</thead>
<tbody>
<tr>
<td>// ----- javascript sample -----</td>
<td></td>
</tr>
<tr>
<td>// instead of:</td>
<td></td>
</tr>
<tr>
<td>// strPathName = Application.ActiveDocument.GetPathName();</td>
<td></td>
</tr>
<tr>
<td>// use now:</td>
<td></td>
</tr>
<tr>
<td>strPathName = Application.ActiveDocument.FullName;</td>
<td></td>
</tr>
</tbody>
</table>

See also

Method: GetPathName() as String

Description
The method GetPathName gets the path of the active document.

See also Document.SetPathName (obsolete).

GridView

See also

Property: GridView as GridView

Description
This property provides access to the grid view functionality of the document.

Errors
- 1400 The object is no longer valid.
- 1407 Invalid address for the return parameter was specified.
- 1417 Document needs to be open in enhanced grid view mode.
**IsModified**

See also

*Property:* IsModified as Boolean

**Description**

True if the document is modified.

**Errors**

- 1400 The object is no longer valid.
- 1407 Invalid address for the return parameter was specified.

**IsValid**

See also

*Method:* IsValid *(strError as Variant)* as Boolean

**Return Value**

True if the document is valid, false if not. To call IsValid(), the application GUI must be visible. (If you wish to validate without the GUI being visible, please use Altova RaptorXML Server.)

**Description**

IsValid validates the document against its associated schema or DTD. strError gives you the same error message as when you validate the file within the GUI.

**Errors**

- 1400 The object is no longer valid.
- 1407 Invalid parameter or invalid address for the return parameter was specified.
- 1408 Unable to validate file.

**IsValidEx**

*Method:* IsValidEx *(nXSDVersion as SPYValidateXSDVersion, nErrorLimit as int, nErrorFormat as SPYValidateErrorFormat, out strError as Variant)* as Boolean

**Return Value**

True if the document is valid, false if not.

**Description**

IsValidEx validates the document against its associated schema or DTD.

**In parameters:**

- nXSDVersion which is an enumeration value of SPYValidateXSDVersion that selects the XSD version to validate against.
- nErrorLimit which is an integer. Values must be 1 to 999.
- nErrorFormat which is an enumeration value of SPYValidateErrorFormat that selects the XSD version to validate against.
Out parameter:
strError is the error message, and is the same as that received when validating the file within the GUI.

Errors
1400 The object is no longer valid.
1407 Invalid parameter or invalid address for the return parameter was specified.
1408 Unable to validate file.

IsWellFormed
See also

Method: IsWellFormed (pData as XMLData, bWithChildren as Boolean, strError as Variant, nErrorPos as Variant, pBadXMLData as Variant) as Boolean

Return Value
True if the document is well formed.

Description
IsWellFormed checks the document for well-formedness starting at the element pData.

If the document is not well formed, strError contains an error message, nErrorPos the position in the file and pBadXMLData holds a reference to the element which breaks the well-formedness. These out-parameters are defined as VARIANTs to support scripting languages like VBScript.

Errors
1400 The object is no longer valid.
1407 Invalid parameter or invalid address for the return parameter was specified.

Example
See IsValid.

Name
See also

Property: Name as String (read-only)

Description
Use this property to retrieve the name - not including the path - of the document file. To change the file name for a document use the property FullName.

Errors
1400 The document object is no longer valid.
1407 Invalid address for the return parameter was specified.
Parent
See also

*Property:* Parent as Documents (read-only)

**Description**
Access the parent of the document object.

**Errors**

- 1400 The document object is no longer valid.
- 1407 Invalid address for the return parameter was specified.

*Property:* Parent as Application (read-only)

Path
See also

*Property:* Path as String (read-only)

**Description**
Use this property to retrieve the path - not including the file name - of the document file. To change the file name and path for a document use the property `FullName`.

**Errors**

- 1400 The document object is no longer valid.
- 1407 Invalid address for the return parameter was specified.

RootElement
See also

*Property:* RootElement as XMLData (read-only)

**Description**
The property `RootElement` provides access to the root element of the XML structure of the document including the XML prolog data. To access the first element of a document's content navigate to the first child of kind `spyXMLDataElement` or use the `Document.DataRoot` property. If the `CurrentViewMode` is not `spyViewGrid` or `spyViewAuthentic` an UpdateXMLData may be necessary to get access to the latest XMLData.

**Errors**

- 1400 The document object is no longer valid.
- 1407 Invalid address for the return parameter was specified.

Save
See also

*Method:* Save()

**Description**
The method writes any modifications of the document to the associated file. See also Document.FullName.

**Errors**

- 1400  The document object is no longer valid.
- 1407  An empty file name has been specified.
- 1403  Error when saving file, probably the file name is invalid.

**SaveAs**

**See also**

**Method:** SaveAs (strFileName as String)

**Description**

Save the document to the file specified. If saving was successful, the FullName property gets set to the specified file name.

**Errors**

- 1400  The document object is no longer valid.
- 1407  An empty file name has been specified.
- 1403  Error when saving file, probably the file name is invalid.

**Saved**

**See also**

**Property:** Saved as Boolean (read-only)

**Description**

This property can be used to check if the document has been saved after the last modifications. It returns the negation of IsModified.

**Errors**

- 1400  The document object is no longer valid.
- 1407  Invalid address for the return parameter was specified.

**SaveInString**

**See also**

**Method:** SaveInString (pData as XMLData, bMarked as Boolean) as String

**Parameters**

- **pData**
  XMLData element to start. Set pData to Document.RootElement if you want to copy the complete file.

- **bMarked**
  If bMarked is true, only the elements selected in the grid view are copied.

**Return Value**
Returns a string with the XML data.

**Description**

SaveInString starts at the element pData and converts the XMLData objects to a string representation. UpdateXMLData() might be indirectly needed as you have to pass the XMLData as parameter to this function.

**Errors**

- 1400  The object is no longer valid.
- 1407  Invalid parameter or invalid address for the return parameter was specified.

**SaveToURL**

**See also**

**Method:** SaveToURL (strURL as String, strUser as String, strPassword as String)

**Return Value**

**Description**

SaveToURL() writes the document to the URL strURL. This method does not set the permanent file path of the document.

**Errors**

- 1400  The object is no longer valid.
- 1402  Invalid URL specified.
- 1403  Error while saving to URL.

**SetActiveDocument**

**See also**

**Method:** SetActiveDocument()

**Description**

The method sets the document as the active and brings it to the front.

**Errors**

- 1400  The object is no longer valid.

**SetEncoding (obsolete)**

**Superseded by** Document.Encoding

```javascript
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.SetEncoding("UTF-16");  
// use now:  
Application.ActiveDocument.Encoding = "UTF-16";
```

**See also**
Method: `SetEncoding (strEncoding as String)`

Description

`SetEncoding` sets the encoding of the document like the menu item "File/Encoding..." in XMLSpy. Possible values for `strEncoding` are, for example:

- 8859-1,
- 8859-2,
- ASCII, ISO-646,
- 850,
- 1252,
- 1255,
- SHIFT-JIS, MS-KANJI,
- BIG5, FIVE,
- UTF-7,
- UTF-8,
- UTF-16
SetExternallsIsValid

See also

Method: SetExternallsIsValid (bValid as Boolean)

Parameters

bValid
Sets the result of an external validation process.

Description
The internal information set by this method is only queried on cancelling the default validation in any OnBeforeValidate handler.

Available with TypeLibrary version 1.5

Errors

1400 The object is no longer valid.

SetPathName (obsolete)

Superseded by Document.FullName

// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.SetPathName("C:\myXMLFiles\test.xml");  
// use now:  
Application.ActiveDocument.FullName = "C:\myXMLFiles\test.xml";

See also

Method: SetPathName (strPath as String)

Description
The method SetPathName sets the path of the active document. SetPathName only copies the string and does not check if the path is valid. All succeeding save operations are done into this file.

StartChanges
See also

Method: StartChanges()

Description
After StartChanges is executed XMLSpy will not update its editor windows until Document.EndChanges is called. This increases performance of complex tasks to the XML structure.
Errors
1400 The object is no longer valid.

Suggestions
Property: Suggestions as Array

Description
This property contains the last valid user suggestions for this document. The XMLSpy generated suggestions can be modified before they are shown to the user in the OnBeforeShowSuggestions event.

Errors
1400 The object is no longer valid.
1407 Invalid parameter or invalid address for the return parameter was specified.

SwitchViewMode
See also

Method: SwitchViewMode (nMode as SPYViewModes) as Boolean

Return value
Returns true if view mode is switched.

Description
The method sets the current view mode of the document in XMLSpy. See also Document.CurrentViewMode.

Errors
1400 The object is no longer valid.
1407 Invalid address for the return parameter was specified.
1417 Invalid view mode specified.

TextView
See also

Property: TextView as TextView

Description
This property provides access to the text view functionality of the document.

Errors
1400 The object is no longer valid.
1407 Invalid address for the return parameter was specified.

Title
See also

Property: Title as String (read-only)
Description
Title contains the file name of the document. To get the path and filename of the file use
FullName.

Errors
1400 The document object is no longer valid.
1407 Invalid address for the return parameter was specified.

TransformXSL
See also

Method: TransformXSL()

Description
TransformXSL processes the XML document via the associated XSL file. See
Document.AssignXSL on how to place a reference to a XSL file into the document.

Errors
1400 The document object is no longer valid.
1411 Error during transformation process.

TransformXSLEx
See also

Method: TransformXSLEx(nAction as SPYDialogAction)

Description
TransformXSLEx processes the XML document via the associated XSL file. The parameter
specifies whether a dialog asking for the result document name should pop up or not. See
Document.AssignXSL on how to place a reference to a XSL file into the document.

Errors
1400 The document object is no longer valid.
1411 Error during transformation process.

TransformXSLFO
See also

Method: TransformXSLFO()

Description
TransformXSLFO processes the XML document via the associated XSLFO file. See
AssignXSLFO on how to place a reference to a XSLFO file into the document. You need to
assign a FOP processor to XMLSpy before you can use this method.

Errors
1400 The document object is no longer valid.
1411 Error during transformation process.
UpdateViews
See also

Method: UpdateViews()

Description
To redraw the Enhanced Grid View and the Tree View call UpdateViews. This can be important after you changed the XMLData structure of a document. This method does not redraw the text view of XMLSpy.

Errors
1400 The document object is no longer valid.

UpdateXMLData
See also

Method: UpdateXMLData() as Boolean

Description
The XMLData tree is updated from the current view. Please note that this can fail in case of the TextView if the current XML text is not well-formed. This is not necessary if CurrentViewMode is spyViewGrid or spyViewAuthentic because these views keep the XMLData updated.

Available with TypeLibrary version 1.5

Errors
1400 The document object is no longer valid.

3.2.11 Documents
See also

Properties
Count
Item

Methods
NewAuthenticFile
NewFile
NewFileFromText
OpenAuthenticFile
OpenFile
OpenURL
OpenURLDialog

Description
This object represents the set of documents currently open in XMLSpy. Use this object to open further documents or iterate through already opened documents.

**Examples**
```vbscript
' ---------------------------------------
' XMLSpy scripting environment - VBScript
' iterate through open documents
' ---------------------------------------
Dim objDocuments
Set objDocuments = Application.Documents

For Each objDoc In objDocuments
    'do something useful with your document
    objDoc.SetActiveDocument()
Next

// ---------------------------------------
// XMLSpy scripting environment - JScript
// close all open documents
// ---------------------------------------
for (var iter = new Enumerator (Application.Documents);
    ! iter.atEnd();
    iter.moveNext())
{
    // MsgBox ("Closing file " + iter.item().Name);
    iter.item().Close (true);
}
```

**Count**

**See also**

*Property:* Count as long

**Description**
Count of open documents.

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1600</td>
<td>Invalid Documents object</td>
</tr>
<tr>
<td>1601</td>
<td>Invalid input parameter</td>
</tr>
</tbody>
</table>

**Item**

**See also**

*Method:* Item (n as long) as Document

**Description**
Gets the document with the index n in this collection. Index is 1-based.

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1600</td>
<td>Invalid Documents object</td>
</tr>
<tr>
<td>1601</td>
<td>Invalid input parameter</td>
</tr>
</tbody>
</table>
**NewAuthenticFile**

See also

**Method:** NewAuthenticFile (strSPSPath as String, strXMLPath as String) as Document

**Parameters**

strSPSPath
The path to the SPS document.

strXMLPath
The new XML document name.

**Return Value**

The method returns the new document.

**Description**

NewAuthenticFile creates a new XML file and opens it in Authentic View using SPS design strSPSPath.

**NewFile**

See also

**Method:** NewFile (strFile as String, strType as String) as Document

**Parameters**

strFile
Full path of new file.

strType
Type of new file as string (i.e. "xml", "xsd", ...)

**Return Value**

Returns the new file.

**Description**

NewFile creates a new file of type strType (i.e. "xml"). The newly created file is also the ActiveDocument.

**NewFileFromText**

See also

**Method:** NewFileFromText (strText as String, strType as String) as Document

**Parameters**

strText
The content of the new document in plain text.

strType
Type of the document to create (i.e. "xml").

**Return Value**
The method returns the new document.

**Description**
NewFileFromText creates a new document with strText as its content.

### OpenAuthenticFile

**See also**

**Method:** `OpenAuthenticFile (strSPSPath as String, strXMLPath as String) as Document`

**Parameters**

- `strSPSPath`  
The path to the SPS document.

- `strXMLPath`  
The path to the XML document (can be empty).

**Return Value**
The method returns the new document.

**Description**
OpenAuthenticFile opens an XML file or database in Authentic View using SPS design strSPSPath.

### OpenFile

**See also**

**Method:** `OpenFile (strPath as String, bDialog as Boolean) as Document`

**Parameters**

- `strPath`  
The path and file name of file to open.

- `bDialog`  
Show dialogs for user input.

**Return Value**
The method returns the opened file on success.

**Description**
OpenFile opens the file strPath. If bDialog is TRUE, a file-dialog will be displayed.

**Example**

```vba
Dim objDoc As Document
Set objDoc = objSpy.Documents.OpenFile(strFile, False)
```
OpenURL

See also

*Method:* `OpenURL (strURL as String, nURLType as SPYURLTypes, nLoading as SPYLoading, strUser as String, strPassword as String) as Document`

**Parameters**

- `strURL`
  URL to open as document.

- `nURLType`
  Type of document to open. Set to -1 for auto detection.

- `nLoading`
  Set `nLoading` to 0 (zero) if you want to load it from cache or proxy. Otherwise set `nLoading` to 1.

- `strUser`
  Name of the user if required. Can be empty.

- `strPassword`
  Password for authentication. Can be empty.

**Return Value**

The method returns the opened document.

**Description**

`OpenURL` opens the URL `strURL`.

OpenURLDialog

See also

*Method:* `OpenURLDialog (strURL as String, nURLType as SPYURLTypes, nLoading as SPYLoading, strUser as String, strPassword as String) as Document`

**Parameters**

- `strURL`
  URL to open as document.

- `nURLType`
  Type of document to open. Set to -1 for auto detection.

- `nLoading`
  Set `nLoading` to 0 (zero) if you want to load it from cache or proxy. Otherwise set `nLoading` to 1.

- `strUser`
  Name of the user if required. Can be empty.

- `strPassword`
  Password for authentication. Can be empty.
Password for authentication. Can be empty.

**Return Value**
The method returns the opened document.

**Description**
*OpenURLDialog* displays the "open URL" dialog to the user and presets the input fields with the given parameters.

### 3.2.12 DTDSchemaGeneratorDlg

**See also**

**Properties and Methods**

Standard automation properties
- Application
- Parent

- DTDSchemaFormat
- ValueList
- TypeDetection
- FrequentElements
- MergeAllEqualNamed
- ResolveEntities
- AttributeTypeDefinition
- GlobalAttributes
- OnlyStringEnums
- MaxEnumLength
- OutputPath
- OutputPathDialogAction

**Description**
Use this object to configure the generation of a schema or DTD. The method *GenerateDTDDOrSchemaEx* expects a *DTDSchemaGeneratorDlg* as parameter to configure the generation as well as the associated user interactions.

**Application**

*Property*: Application as Application (read-only)

**Description**
Access the XMLSpy application object.

**Errors**

- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

**AttributeTypeDefinition**

*Property*: AttributeTypeDefinition as SPYAttributeTypeDefinition

**Description**
Specifies how attribute definitions get merged.

**Errors**
- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

**DTDSchemaFormat**

Property: DTDSchemaFormat as SPYDTDSchemaFormat

Description
Sets the schema output format to DTD, or W3C.

**Errors**
- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

**FrequentElements**

Property: FrequentElements as SPYFrequentElements

Description
Shall the types for all elements be defined as global? Use that value spyGlobalComplexType to define them on global scope. Otherwise, use the value spyGlobalElements.

**Errors**
- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

**Global Attributes**

Property: GlobalAttributes as Boolean

Description
Shall attributes with same name and type be resolved globally?

**Errors**
- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

**MaxEnumLength**

Property: MaxEnumLength as Integer

Description
Specifies the maximum number of characters allowed for enumeration names. If one value is longer than this, no enumeration will be generated.

**Errors**
- 3000 The object is no longer valid.
3001  Invalid address for the return parameter was specified.

**MergeAllEqualNamed**

*Property:* `MergeAllEqualNamed` as Boolean

*Description*
Shall types of all elements with the same name be merged into one type?

*Errors*
- 3000  The object is no longer valid.
- 3001  Invalid address for the return parameter was specified.

**OnlyStringEnums**

*Property:* `OnlyStringEnums` as Boolean

*Description*
Specifies if enumerations will be created only for plain strings or all types of values.

*Errors*
- 3000  The object is no longer valid.
- 3001  Invalid address for the return parameter was specified.

**OutputPath**

*Property:* `OutputPath` as String

*Description*
Selects the file name for the generated schema/DTD.

*Errors*
- 3000  The object is no longer valid.
- 3001  Invalid address for the return parameter was specified.

**OutputPathDialogAction**

*Property:* `OutputPathDialogAction` as `SPYDialogAction`

*Description*
 Defines how the sub-dialog for selecting the schema/DTD output path gets handled. Set this value to `spyDialogUserInput(2)` to show the dialog with the current value of the `OutputPath` property as default. Use `spyDialogOK(0)` to hide the dialog from the user.

*Errors*
- 3000  The object is no longer valid.
- 3001  Invalid address for the return parameter was specified.
Parent

**Property:** Parent as Dialogs (read-only)

**Description**
Access the parent of the object.

**Errors**
- 3000  The object is no longer valid.
- 3001  Invalid address for the return parameter was specified.

ResolveEntities

**Property:** ResolveEntities as Boolean

**Description**
Shall all entities be resolved before generation starts? If yes, an info-set will be built.

**Errors**
- 3000  The object is no longer valid.
- 3001  Invalid address for the return parameter was specified.

TypeDetection

**Property:** TypeDetection as SPYTypeDetection

**Description**
Specifies granularity of simple type detection.

**Errors**
- 3000  The object is no longer valid.
- 3001  Invalid address for the return parameter was specified.

ValueList

**Property:** ValueList as Integer

**Description**
Generate not more than this amount of enumeration-facets per type. Set to -1 for unlimited.

**Errors**
- 3000  The object is no longer valid.
- 3001  Invalid address for the return parameter was specified.

3.2.13 ElementList

See also

**Properties**
- Count
**Item**

**Methods**
*RemoveElement*

**Description**
Element lists are used for different purposes during export and import of data. Depending on this purpose, different properties of *ElementListItem* are used.

It can hold
- a list of table names returned by a call to `Application.GetDatabaseTables`,
- a list of field names returned by a call to `Application.GetDatabaseImportElementList` or `Application.GetTextImportElementList`,
- a field name filter list used in `Application.ImportFromDatabase` and `Application.ImportFromText`,
- a list of table names and counts for their rows and columns as returned by calls to `GetExportElementList` or
- a field name filter list used in `Document.ExportToDatabase` and `Document.ExportToText`.

**Count**

See also

*Property:* `Count` as long (read-only)

**Description**
Count of elements in this collection.

**Item**

See also

*Method:* `Item(n as long)` as *ElementListItem*

**Description**
Gets the element with the index `n` from this collection. The first item has index 1.

**RemoveElement**

See also

*Method:* `RemoveElement(Index as long)`

**Description**
`RemoveElement` removes the element `Index` from the collection. The first `Item` has index 1.

**3.2.14 ElementListItem**

See also

**Properties**
Name
ElementKind
FieldCount
RecordCount

Description
An element in an ElementList. Usage of its properties depends on the purpose of the element list. For details see ElementList.

ElementKind
See also

Property: ElementKind as SPYXMLDataKind

Description
Specifies if a field should be imported as XML element (data value of spyXMLDataElement) or attribute (data value of spyXMLDataAttr).

FieldCount
See also

Property: FieldCount as long (read-only)

Description
Count of fields (i.e. columns) in the table described by this element. This property is only valid after a call to Document.GetExportElementList.

Name
See also

Property: Name as String (read-only)

Description
Name of the element. This is either the name of a table or a field, depending on the purpose of the element list.

RecordCount
See also

Property: RecordCount as long (read-only)

Description
Count of records (i.e. rows) in the table described by this element. This property is only valid after a call to Document.GetExportElementList.
3.2.15 ExportSettings

See also

Properties

ElementList

EntitiesToText

ExportAllElements

SubLevelLimit

FromAttributes

FromSingleSubElements

FromTextValues

CreateKeys

IndependentPrimaryKey

Namespace

ExportCompleteXML

StartFromElement

Description

ExportSettings contains options used during export of XML data to a database or text file. See Import and export of data for a general overview.

CreateKeys

See also

Property: CreateKeys as Boolean

Description

This property turns creation of keys (i.e. primary key and foreign key) on or off. Default is True.

ElementList

See also

Property: ElementList as ElementList

Description

Default is empty list. This list of elements defines which fields will be exported. To get the list of available fields use Document.GetExportElementList. It is possible to prevent exporting columns by removing elements from this list with ElementList.RemoveElement before passing it to Document.ExportToDatabase or Document.ExportToText.
**EntitiesToText**

**See also**

**Property:** `EntitiesToText` as Boolean

**Description**
Defines if XML entities should be converted to text or left as they are during export. Default is True.

**ExportAllElements**

**See also**

**Property:** `ExportAllElements` as Boolean

**Description**
If set to true, all elements in the document will be exported. If set to false, then `ExportSettings.SubLevelLimit` is used to restrict the number of sub levels to export. Default is true.

**ExportCompleteXML**

**See also**

**Property:** `ExportCompleteXML` as Boolean

**Description**
Defines whether the complete XML is exported or only the element specified by `StartFromElement` and its children. Default is True.

**FromAttributes**

**See also**

**Property:** `FromAttributes` as Boolean

**Description**
Set `FromAttributes` to false if no export data should be created from attributes. Default is True.

**FromSingleSubElements**

**See also**

**Property:** `FromSingleSubElements` as Boolean

**Description**
Set `FromSingleSubElements` to false if no export data should be created from elements. Default is True.
FromTextValues
See also

Property: FromTextValues as Boolean

Description
Set FromTextValues to false if no export data should be created from text values. Default is True.

IndependentPrimaryKey
See also

Property: IndependentPrimaryKey as Boolean

Description
Turns creation of independent primary key counter for every element on or off. If ExportSettings.CreateKeys is False, this property will be ignored. Default is True.

Namespace
See also

Property: Namespace as SPYExportNamespace

Description
The default setting removes all namespace prefixes from the element names. In some database formats the colon is not a legal character. Default is spyNoNamespace.

StartFromElement
See also

Property: StartFromElement as String

Description
Specifies the start element for the export. This property is only considered when ExportCompleteXML is false.

SubLevelLimit
See also

Property: SubLevelLimit as Integer

Description
Defines the number of sub levels to include for the export. Default is 0. This property is ignored if ExportSettings.ExportAllElements is true.
### 3.2.16 FileSelectionDlg

#### See also

**Properties and Methods**

- **Standard automation properties**
  - `Application`
  - `Parent`

- **Dialog properties**
  - `FullName`

Acceptance or cancellation of action that caused event

**DialogAction**

#### Description

The dialog object allows you to receive information about an event and pass back information to the event handler in the same way as with a user dialog. Use the `FileSelectionDlg.FullName` to select or modify the file path and set the `FileSelectionDlg.DialogAction` property to cancel or agree with the action that caused the event.

#### Application

**See also**

- **Property:** `Application` as `Application` (read-only)

**Description**

Access the XMLSpy application object.

**Errors**

- 2400 The object is no longer valid.
- 2401 Invalid address for the return parameter was specified.

#### DialogAction

**Property:** `DialogAction` as `SPYDialogAction`

**Description**

If you want your script to perform the file selection operation without any user interaction necessary, simulate user interaction by either setting the property to `spyDialogOK(0)` or `spyDialogCancel(1)`.

To allow your script to fill in the default values but let the user see and react on the dialog, use the value `spyDialogUserInput(2)`. If you receive a FileSelectionDlg object in an event handler, `spyDialogUserInput(2)` is not supported and will be interpreted as `spyDialogOK(0)`.

**Errors**

- 2400 The object is no longer valid.
- 2401 Invalid value for dialog action or invalid address for the return parameter was specified.
**FullName**

*Property:* `FullName` as `String`

**Description**
Access the full path of the file the gets selected by the dialog. Most events that pass a `FileSelectionDlg` object to you allow you modify this value and thus influence the action that caused the event (e.g. load or save to a different location).

**Errors**
- 2400: The object is no longer valid.
- 2401: Invalid address for the return parameter was specified.

**Parent**

*See also*

*Property:* `Parent` as `Dialogs` (read-only)

**Description**
Access the parent of the object.

**Errors**
- 2400: The object is no longer valid.
- 2401: Invalid address for the return parameter was specified.

### 3.2.17 FindInFilesDlg

*See also*

**Properties and Methods**

Standard automation properties
- `Application`
- `Parent`
- `Find`
- `RegularExpression`
- `Replace`
- `DoReplace`
- `ReplaceOnDisk`
- `MatchWholeWord`
- `MatchCase`
- `SearchLocation`
- `StartFolder`
- `IncludeSubfolders`
- `SearchInProjectFiles`
- `DoExternal`
- `FileExtension`
- `AdvancedXMLSearch`
- `XMLElementNames`
- `XMLElementContents`
- `XMLAttributeNames`
- `XMLAttributeContents`
**XMLComments**
**XMLCData**
**XMLPI**
**XMLRest**
**ShowResult**

**Description**
Use this object to configure the search (or replacement) for strings in files. The method **FindInFiles** expects a **FindInFilesDlg** as parameter.

**AdvancedXMLSearch**

**Property:** AdvancedXMLSearch as Boolean

**Description**
Specifies if the XML search properties (XMLElementNames, XMLElementContents, XMLAttributeNames, XMLAttributeContents, XMLComments, XMLData, XMLPI and XMLRest) are considered. The default is false.

**Errors**
- 3500  The object is no longer valid.
- 3501  Invalid address for the return parameter was specified.

**Application**

**Property:** Application as Application (read-only)

**Description**
Access the XMLSpy application object.

**Errors**
- 3500  The object is no longer valid.
- 3501  Invalid address for the return parameter was specified.

**DoReplace**

**Property:** DoReplace as Boolean

**Description**
Specifies if the matched string is replaced by the string defined in **Replace**. The default is false.

**Errors**
- 3500  The object is no longer valid.
- 3501  Invalid address for the return parameter was specified.

**FileExtension**

**Property:** FileExtension as String

**Description**
Specifies the file filter of the files that should be considered during the search. Multiple file filters must be delimited with a semicolon (eg: ".xml;*.dtd;*.xsd"). Use the wildcards * and ? to define the file filter.
Errors
3500  The object is no longer valid.
3501  Invalid address for the return parameter was specified.

Find
Property: Find as String

Description
Specifies the string to search for.

Errors
3500  The object is no longer valid.
3501  Invalid address for the return parameter was specified.

IncludeSubfolders
Property: IncludeSubfolders as Boolean

Description
Specifies if subfolders are searched too. The default is true.

Errors
3500  The object is no longer valid.
3501  Invalid address for the return parameter was specified.

MatchCase
Property: MatchCase as Boolean

Description
Specifies if the search is case sensitive. The default is true.

Errors
3500  The object is no longer valid.
3501  Invalid address for the return parameter was specified.

MatchWholeWord
Property: MatchWholeWord as Boolean

Description
Specifies whether the whole word or just a part of it must match. The default is false.

Errors
3500  The object is no longer valid.
3501  Invalid address for the return parameter was specified.

Parent
Property: Parent as Dialogs (read-only)

Description
Access the parent of the object.

Errors
3500 The object is no longer valid.
3501 Invalid address for the return parameter was specified.

RegularExpression
Property: RegularExpression as Boolean

Description
Specifies if Find contains a regular expression. The default is false.

Errors
3500 The object is no longer valid.
3501 Invalid address for the return parameter was specified.

Replace
Property: Replace as String

Description
Specifies the replacement string. The matched string is only replaced if DoReplace is set true.

Errors
3500 The object is no longer valid.
3501 Invalid address for the return parameter was specified.

ReplaceOnDisk
Property: ReplaceOnDisk as Boolean

Description
Specifies if the replacement is done directly on disk. The modified file is not opened. The default is false.

Errors
3500 The object is no longer valid.
3501 Invalid address for the return parameter was specified.

SearchInProjectFilesDoExternal
Property: SearchInProjectFilesDoExternal as Boolean

Description
Specifies if the external folders in the open project are searched, when a project search is performed. The default is false.

Errors
3500 The object is no longer valid.
3501 Invalid address for the return parameter was specified.
SearchLocation

Property: SearchLocation as SPYFindInFilesSearchLocation

Description
Specifies the location of the search. The default is spyFindInFiles_Documents.

Errors
3500 The object is no longer valid.
3501 Invalid address for the return parameter was specified.

ShowResult

Property: ShowResult as Boolean

Description
Specifies if the result is displayed in the Find in Files output window. The default is false.

Errors
3500 The object is no longer valid.
3501 Invalid address for the return parameter was specified.

StartFolder

Property: StartFolder as String

Description
Specifies the folder where the disk search starts.

Errors
3500 The object is no longer valid.
3501 Invalid address for the return parameter was specified.

XMLAttributeContents

Property: XMLAttributeContents as Boolean

Description
Specifies if attribute contents are searched when AdvancedXMLSearch is true. The default is true.

Errors
3500 The object is no longer valid.
3501 Invalid address for the return parameter was specified.

XMLAttributeNames

Property: XMLAttributeNames as Boolean

Description
Specifies if attribute names are searched when AdvancedXMLSearch is true. The default is true.

Errors
3500 The object is no longer valid.
3501 Invalid address for the return parameter was specified.
**XMLCData**

*Property:* `XMLCData` as Boolean

*Description*
Specifies if CData tags are searched when `AdvancedXMLSearch` is true. The default is true.

*Errors*
- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

**XMLComments**

*Property:* `XMLComments` as Boolean

*Description*
Specifies if comments are searched when `AdvancedXMLSearch` is true. The default is true.

*Errors*
- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

**XMLElementContents**

*Property:* `XMLElementContents` as Boolean

*Description*
Specifies if element contents are searched when `AdvancedXMLSearch` is true. The default is true.

*Errors*
- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

**XMLElementNames**

*Property:* `XMLElementNames` as Boolean

*Description*
Specifies if element names are searched when `AdvancedXMLSearch` is true. The default is true.

*Errors*
- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

**XMLPI**

*Property:* `XMLPI` as Boolean

*Description*
Specifies if XML processing instructions are searched when `AdvancedXMLSearch` is true. The default is true.

*Errors*
- 3500 The object is no longer valid.
3501 Invalid address for the return parameter was specified.

XMLRest

Property: `XMLRest` as Boolean

Description
Specifies if the rest of the XML (which is not covered by the other XML search properties) is searched when `AdvancedXMLSearch` is true. The default is true.

Errors
- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

3.2.18 FindInFilesResult

See also

Properties and Methods

Standard automation properties
- `Application`
- `Parent`
- `Count`
- `Item`
- `Path`
- `Document`

Description
This object represents a file that matched the search criteria. It contains a list of `FindInFilesResultMatch` objects that describe the matching position.

Application

Property: `Application` as `Application` (read-only)

Description
Access the XMLSpy application object.

Errors
- 3700 The object is no longer valid.
- 3701 Invalid address for the return parameter was specified.

Count

Property: `Count` as long (read-only)

Description
Count of elements in this collection.
Document

Property: Path as Document (read-only)

Description
This property returns the Document object if the matched file is already open in XMLSpy.

Errors
3700 The object is no longer valid.
3701 Invalid address for the return parameter was specified.

Item

Method: Item(n as long) as FindInFilesResultMatch

Description
Gets the element with the index n from this collection. The first item has index 1.

Parent

Property: Parent as FindInFilesResults (read-only)

Description
Access the parent of the object.

Errors
3700 The object is no longer valid.
3701 Invalid address for the return parameter was specified.

Path

Property: Path as String (read-only)

Description
Returns the path of the file that matched the search criteria.

Errors
3700 The object is no longer valid.
3701 Invalid address for the return parameter was specified.

3.2.19 FindInFilesResultMatch

See also

Properties and Methods

Standard automation properties
Application
Parent
Description
Contains the exact position in the file of the matched string.

Application
Property: Application as Application (read-only)
Description
Access the XMLSpy application object.

Errors
3800 The object is no longer valid.
3801 Invalid address for the return parameter was specified.

Length
Property: Length as Long (read-only)
Description
Returns the length of the matched string.

Errors
3800 The object is no longer valid.
3801 Invalid address for the return parameter was specified.

Line
Property: Line as Long (read-only)
Description
Returns the line number of the match. The line numbering starts with 0.

Errors
3800 The object is no longer valid.
3801 Invalid address for the return parameter was specified.

LineText
Property: LineText as String (read-only)
Description
Returns the text of the line.

Errors
3800 The object is no longer valid.
Invalid address for the return parameter was specified.

**Parent**

*Property:* `Parent` as `FindInFilesResult` (read-only)

**Description**
Access the parent of the object.

**Errors**
- 3800: The object is no longer valid.
- 3801: Invalid address for the return parameter was specified.

**Position**

*Property:* `Position` as `Long` (read-only)

**Description**
Returns the start position of the match in the line. The position numbering starts with 0.

**Errors**
- 3800: The object is no longer valid.
- 3801: Invalid address for the return parameter was specified.

**Replaced**

*Property:* `Replaced` as `Boolean` (read-only)

**Description**
True if the matched string was replaced.

**Errors**
- 3800: The object is no longer valid.
- 3801: Invalid address for the return parameter was specified.

### 3.2.20 FindInFilesResults

**See also**

**Properties and Methods**

Standard automation properties

*Application*

*Parent*

*Count*

*Item*

**Description**
This is the result of the `FindInFiles` method. It is a list of `FindInFilesResult` objects.
Application

Property: Application as Application (read-only)

Description
Access the XMLSpy application object.

Errors
3600  The object is no longer valid.
3601  Invalid address for the return parameter was specified.

Count

Property: Count as long (read-only)

Description
Count of elements in this collection.

Item

Method: Item(n as long) as FindInFilesResult

Description
Gets the element with the index \( n \) from this collection. The first item has index 1.

Parent

Property: Parent as Application (read-only)

Description
Access the parent of the object.

Errors
3600  The object is no longer valid.
3601  Invalid address for the return parameter was specified.

3.2.21  GenerateSampleXMLDlg

See also

Properties and Methods

Standard automation properties
Application
Parent

NonMandatoryAttributes
NonMandatoryElements
RepeatCount
FillAttributesWithSampleData
FillElementsWithSampleData
Properties that are no longer supported
TakeFirstChoice - obsolete
FillWithSampleData - obsolete
Optimization - obsolete

Description
Used to set the parameters for the generation of sample XML instances based on a W3C schema or DTD.

Application
Property: Application as Application (read-only)
Description
Access the XMLSpy application object.

Errors
2200 The object is no longer valid.
2201 Invalid address for the return parameter was specified.

ChoiceMode
Property: ChoiceMode as SPYSampleXMLGenerationChoiceMode

Description
Specifies which elements will be generated.

Errors
2200 The object is no longer valid.
2201 Invalid address for the return parameter was specified.

ConsiderSampleValueHints
Property: ConsiderSampleValueHints as Boolean

Description
Selects whether to use SampleValueHints or not.

Errors
2200 The object is no longer valid.
2201 Invalid address for the return parameter was specified.

**ContentOfNillableElementsIsNonMandatory**

*Property:* `ContentOfNillableElementsIsNonMandatory` as Boolean

*Description*
If true, the contents of elements that are nillable will not be treated as mandatory.

*Errors*
2200 The object is no longer valid.
2201 Invalid address for the return parameter was specified.

**FillAttributesWithSampleData**

*Property:* `FillAttributesWithSampleData` as Boolean

*Description*
If true, attributes will have sample content.

*Errors*
2200 The object is no longer valid.
2201 Invalid address for the return parameter was specified.

**FillElementsWithSampleData**

*Property:* `FillElementsWithSampleData` as Boolean

*Description*
If true, elements will have sample content.

*Errors*
2200 The object is no longer valid.
2201 Invalid address for the return parameter was specified.

**FillWithSampleData - obsolete**

*Property:* `FillWithSampleData` as Boolean

*Description*
Do no longer access this property. Use `FillAttributesWithSampleData` and `FillElementsWithSampleData`, instead.

*Errors*
0001 The property is no longer accessible.

**LocalNameOfRootElement**

*Property:* `LocalNameOfRootElement` as String

*Description*
Specifies the local name of the root element for the generated sample XML.

Errors
2200 The object is no longer valid.
2201 Invalid address for the return parameter was specified.

NamespacesOfRootElement

Property: NamespacesOfRootElement as String

Description
Specifies the namespace URI of the root element for the generated sample XML.

Errors
2200 The object is no longer valid.
2201 Invalid address for the return parameter was specified.

NonMandatoryAttributes

Property: NonMandatoryAttributes as Boolean

Description
If true attributes which are not mandatory are created in the sample XML instance file.

Errors
2200 The object is no longer valid.
2201 Invalid address for the return parameter was specified.

NonMandatoryElements

Property: NonMandatoryElements as Boolean

Description
If true, elements which are not mandatory are created in the sample XML instance file.

Errors
2200 The object is no longer valid.
2201 Invalid address was specified for the return parameter.

Optimization - obsolete

Property: Optimization as SPYSampleXMLGenerationOptimization

Description
Do not use this property any longer. Use ChoiceMode and NonMandatoryElements.

Errors
0001 The property is no longer accessible.

OptionsDialogAction

Property: OptionsDialogAction as SPYDialogAction
Description
To allow your script to fill in the default values and let the user see and react on the dialog, set this property to the value `spyDialogUserInput(2)`. If you want your script to define all the options in the schema documentation dialog without any user interaction necessary, use `spyDialogOK(0)`. Default is `spyDialogOK`.

Errors
- 2200 The object is no longer valid.
- 2201 Invalid value has been used to set the property.
  Invalid address for the return parameter was specified.

Parent
Property: Parent as Dialogs (read-only)

Description
Access the parent of the object.

Errors
- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

RepeatCount
Property: RepeatCount as long

Description
Number of elements to create for repeated types.

Errors
- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

SampleValueHints
Property: SampleValueHints as SPYSampleXMLGenerationSampleValueHints

Description
Specifies how to select data for the generated sample file.

Errors
- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

SchemaOrDTDAssignment
Property: SchemaOrDTDAssignment as SPYSampleXMLGenerationSchemaOrDTDAssignment

Description
Specifies in which way a reference to the related schema or DTD - which is this document - will
be generated into the sample XML.

Errors
   2200   The object is no longer valid.
   2201   Invalid address for the return parameter was specified.

**TakeFirstChoice - obsolete**
*Property:* TakesFirstChoice as Boolean

Description
Do no longer use this property.

Errors
   0001   The property is no longer accessible.

**TryToUseNonAbstractTypes**
*Property:* TryToUseNonAbstractTypes as Boolean

Description
If true, tries to use a non-abstract type for xsi:type, if element has an abstract type.

Errors
   2200   The object is no longer valid.
   2201   Invalid address for the return parameter was specified.

### 3.2.22 GridView

See also

**Methods**
- Deselect
- Select
- SetFocus

**Properties**
- CurrentFocus
- IsVisible

Description
GridView Class

**Events**

**OnBeforeDrag**

See also

*Event:* OnBeforeDrag() as Boolean
**XMLSpy scripting environment - VBScript:**
Function On_BeforeDrag()
    ' On_BeforeStartEditing = False ' to prohibit dragging
End Function

**XMLSpy scripting environment - JScript:**
function On_BeforeDrag()
{
    // return false; /* to prohibit dragging */
}

**XMLSpy IDE Plugin:**
IXMLSpyPlugIn.OnEvent(4, ...) // nEventId = 4

**Description**
This event gets fired on an attempt to drag an XMLData element on the grid view. Return false to prevent dragging the data element to a different position.

**OnBeforeDrop**

**See also**

**Event:** OnBeforeDrop(objXMLData as XMLData) as Boolean

**XMLSpy scripting environment - VBScript:**
Function On_BeforeDrop(objXMLData)
    ' On_BeforeStartEditing = False ' to prohibit dropping
End Function

**XMLSpy scripting environment - JScript:**
function On_BeforeDrop(objXMLData)
{
    // return false; /* to prohibit dropping */
}

**XMLSpy IDE Plugin:**
IXMLSpyPlugIn.OnEvent(5, ...) // nEventId = 5

**Description**
This event gets fired on an attempt to drop a previously dragged XMLData element on the grid view. Return false to prevent the data element to be moved from its original position to the drop destination position.

**OnBeforeStartEditing**

**See also**

**Event:** OnBeforeStartEditing(objXMLData as XMLData, bEditingName as Boolean)as Boolean
**XMLSpy scripting environment - VBScript:**

Function `On_BeforeStartEditing(objXMLData, bEditingName)`

' On_BeforeStartEditing = False ' to prohibit editing the field
End Function

**XMLSpy scripting environment - JScript:**

function `On_BeforeStartEditing(objXMLData, bEditingName)`
{
    // return false; /* to prohibit editing the field */
}

**XMLSpy IDE Plugin:**

`IXMLSpyPlugIn.OnEvent(1,...) // nEventId = 1`

**Description**

This event gets fired before the editing mode for a grid cell gets entered. If the parameter `bEditingName` is true, the name part of the element will be edited, if its value is false, the value part will be edited.

**OnEditingFinished**

**See also**

**Event:** `OnEditingFinished(objXMLData as XMLData, bEditingName as Boolean)`

**XMLSpy scripting environment - VBScript:**

Function `On_EditingFinished(objXMLData, bEditingName)`
End Function

**XMLSpy scripting environment - JScript:**

function `On_EditingFinished(objXMLData, bEditingName)`
{
}

**XMLSpy IDE Plugin:**

`IXMLSpyPlugIn.OnEvent(2,...) // nEventId = 2`

**Description**

This event gets fired when the editing mode of a grid cell gets left. The parameter `bEditingName` specifies if the name part of the element has been edited.

**OnFocusChanged**

**See also**

**Event:** `OnFocusChanged(objXMLData as XMLData, bSetFocus as Boolean, bEditingName as Boolean)`
**XMLSpy scripting environment - VBScript:**

```vbscript
Function On_FocusChanged(objXMLData, bSetFocus, bEditingName)
End Function
```

**XMLSpy scripting environment - JScript:**

```javascript
function On_FocusChanged(objXMLData, bSetFocus, bEditingName)
{
}
```

**XMLSpy IDE Plugin:**

```csharp
IXMLSpyPlugIn.OnEvent (3, ...) // nEventId = 3
```

**Description**

This event gets fired whenever a grid cell receives or loses the cursor focus. If the parameter `bEditingName` is `true`, focus of the name part of the grid element has changed. Otherwise, focus of the value part has changed.

### CurrentFocus

**See also**

**Property:** `CurrentFocus` as `XMLData`

**Description**

Holds the XML element with the current focus. This property is read-only.

### Deselect

**See also**

**Method:** `Deselect(pData as XMLData)`

**Description**

Deselects the element `pData` in the grid view.

### IsVisible

**See also**

**Property:** `IsVisible` as `Boolean`

**Description**

True if the grid view is the active view of the document. This property is read-only.

### Select

**See also**

**Method:** `Select(pData as XMLData)`

**Description**
Selects the XML element `pData` in the grid view.

**SetFocus**

**See also**

**Method:** `SetFocus (pFocusData as XMLData)`

**Description**

Sets the focus to the element `pFocusData` in the grid view.

### 3.2.23 SchemaDocumentationDlg

**See also**

**Properties and Methods**

- Standard automation properties
  - `Application`
  - `Parent`
- Interaction and visibility properties
  - `OutputFile`
  - `OutputFileDialogAction`
  - `OptionsDialogAction`
  - `ShowProgressBar`
  - `ShowResult`
- Document generation options and methods
  - `OutputFormat`
  - `UseFixedDesign`
  - `SPSFile`
  - `EmbedDiagrams`
  - `DiagramFormat`
  - `MultipleOutputFiles`
  - `EmbedCSSInHTML`
  - `CreateDiagramsFolder`
  - `GenerateRelativeLinks`
  - `IncludeAll`
  - `IncludeIndex`
  - `IncludeGlobalAttributes`
  - `IncludeGlobalElements`
  - `IncludeLocalAttributes`
  - `IncludeLocalElements`
  - `IncludeGroups`
  - `IncludeComplexTypes`
  - `IncludeSimpleTypes`
  - `IncludeAttributeGroups`
  - `IncludeRedefines`
  - `IncludeReferencedSchemas`
  - `AllDetails`
ShowDiagram
ShowNamespace
ShowType
ShowChildren
ShowUsedBy
ShowProperties
ShowSingleFacets
ShowPatterns
ShowEnumerations
ShowAttributes
ShowIdentityConstraints
ShowAnnotations
ShowSourceCode

**Description**
This object combines all options for schema document generation as they are available through user interface dialog boxes in XMLSpy. The document generation options are initialized with the values used during the last generation of schema documentation. However, before using the object you have to set the `SetOutputFile` property to a valid file path. Use `OptionsDialogAction`, `OutputFileDialogAction` and `ShowProgressBar` to specify the level of user interaction desired. You can use `IncludeAll` and `AllDetails` to set whole option groups at once or the individual properties to operate on a finer granularity.

**AllDetails**

**See also**

**Method:** `AllDetails` (`i_bDetailsOn` as Boolean)

**Description**
Use this method to turn all details options on or off.

**Errors**
2900 The object is no longer valid.

**Application**

**See also**

**Property:** `Application` as `Application` (read-only)

**Description**
Access the XMLSpy application object.

**Errors**
2900 The object is no longer valid.
2901 Invalid address for the return parameter was specified.
CreateDiagramsFolder

See also

*Property:* CreateDiagramsFolder as Boolean

**Description**
Set this property to `true`, to create a directory for the created images. Otherwise the diagrams will be created next to the documentation. This property is only available when the diagrams are not embedded. The default for the first run is false.

**Errors**
- 2900  The object is no longer valid.
- 2901  Invalid address for the return parameter was specified.

DiagramFormat

See also

*Property:* DiagramFormat as SPYImageKind

**Description**
This property specifies the generated diagram image type. This property is not available for HTML documentation. The property is initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`. The default for the first run is PNG.

**Errors**
- 2900  The object is no longer valid.
- 2901  Invalid address for the return parameter was specified.

EmbedCSSInHTML

See also

*Property:* EmbedCSSInHTML as Boolean

**Description**
Set this property to `true`, to embed the CSS data in the generated HTML document. Otherwise a separate file will be created and linked. This property is only available for HTML documentation. The default for the first run is true.

**Errors**
- 2900  The object is no longer valid.
- 2901  Invalid address for the return parameter was specified.

EmbedDiagrams

See also

*Property:* EmbedDiagrams as Boolean
Description
Set this property to `true`, to embed the diagrams in the generated document. This property is not available for HTML documentation. The property is initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`. The default for the first run is true.

Errors
- 2900: The object is no longer valid.
- 2901: Invalid address for the return parameter was specified.

GenerateRelativeLinks

See also

Property: `GenerateRelativeLinks` as Boolean

Description
Set this property to `true`, to create relative paths to local files. This property is not available for HTML documentation. The property is initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`. The default for the first run is false.

Errors
- 2900: The object is no longer valid.
- 2901: Invalid address for the return parameter was specified.

IncludeAll

See also

Method: `IncludeAll` (i_bInclude as Boolean)

Description
Use this method to mark or unmark all include options.

Errors
- 2900: The object is no longer valid.

IncludeAttributeGroups

See also

Property: `IncludeAttributeGroups` as Boolean

Description
Set this property to `true`, to include attribute groups in the schema documentation. The property is initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`. The default for the first run is true.

Errors
IncludeComplexTypes

See also

Property: IncludeComplexTypes as Boolean

Description
Set this property to true, to include complex types in the schema documentation. The property is initialized with the value used during the last call to Document.GenerateSchemaDocumentation. The default for the first run is true.

Errors
2900 The object is no longer valid.
2901 Invalid address for the return parameter was specified.

IncludeGlobalAttributes

See also

Property: IncludeGlobalAttributes as Boolean

Description
Set this property to true, to include global attributes in the schema documentation. The property is initialized with the value used during the last call to Document.GenerateSchemaDocumentation. The default for the first run is true.

Errors
2900 The object is no longer valid.
2901 Invalid address for the return parameter was specified.

IncludeGlobalElements

See also

Property: IncludeGlobalElements as Boolean

Description
Set this property to true, to include global elements in the schema documentation. The property is initialized with the value used during the last call to Document.GenerateSchemaDocumentation. The default for the first run is true.

Errors
2900 The object is no longer valid.
2901 Invalid address for the return parameter was specified.

IncludeGroups

See also
Property: IncludeGroups as Boolean

Description
Set this property to true, to include groups in the schema documentation. The property is initialized with the value used during the last call to Document.GenerateSchemaDocumentation. The default for the first run is true.

Errors
2900 The object is no longer valid.
2901 Invalid address for the return parameter was specified.

IncludeIndex
See also

Property: IncludeIndex as Boolean

Description
Set this property to true, to include an index in the schema documentation. The property is initialized with the value used during the last call to Document.GenerateSchemaDocumentation. The default for the first run is true.

Errors
2900 The object is no longer valid.
2901 Invalid address for the return parameter was specified.

IncludeLocalAttributes
See also

Property: IncludeLocalAttributes as Boolean

Description
Set this property to true, to include local attributes in the schema documentation. The property is initialized with the value used during the last call to Document.GenerateSchemaDocumentation. The default for the first run is true.

Errors
2900 The object is no longer valid.
2901 Invalid address for the return parameter was specified.

IncludeLocalElements
See also

Property: IncludeLocalElements as Boolean

Description
Set this property to true, to include local elements in the schema documentation. The property is initialized with the value used during the last call to Document.GenerateSchemaDocumentation. The default for the first run is true.
IncludeRedefines

See also

**Property:** IncludeRedefines as Boolean

**Description**
Set this property to `true`, to include redefines in the schema documentation. The property is initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`. The default for the first run is true.

Errors
2900 The object is no longer valid.
2901 Invalid address for the return parameter was specified.

IncludeReferencedSchemas

See also

**Property:** IncludeReferencedSchemas as Boolean

**Description**
Set this property to `true`, to include referenced schemas in the schema documentation. The property is initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`. The default for the first run is true.

Errors
2900 The object is no longer valid.
2901 Invalid address for the return parameter was specified.

IncludeSimpleTypes

See also

**Property:** IncludeSimpleTypes as Boolean

**Description**
Set this property to `true`, to include simple types in the schema documentation. The property is initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`. The default for the first run is true.

Errors
2900 The object is no longer valid.
2901 Invalid address for the return parameter was specified.
MultipleOutputFiles

See also

Property: MultipleOutputFiles as Boolean

Description
Set this property to true, to split the documentation files. The property is initialized with the value used during the last call to Document.GenerateSchemaDocumentation. The default for the first run is false.

Errors
2900  The object is no longer valid.
2901  Invalid value has been used to set the property.
                   Invalid address for the return parameter was specified.

OptionsDialogAction

See also

Property: OptionsDialogAction as SPYDialogAction

Description
To allow your script to fill in the default values and let the user see and react on the dialog, set this property to the value spyDialogUserInput(2). If you want your script to define all the options in the schema documentation dialog without any user interaction necessary, use spyDialogOK(0). Default is spyDialogOK.

Errors
2900  The object is no longer valid.
2901  Invalid value has been used to set the property.
                   Invalid address for the return parameter was specified.

OutputFile

See also

Property: OutputFile as String

Description
Full path and name of the file that will contain the generated documentation. In case of HTML output, additional '.png' files will be generated based on this filename. The default value for this property is an empty string and needs to be replaced before using this object in a call to Document.GenerateSchemaDocumentation.

Errors
2900  The object is no longer valid.
2901  Invalid address for the return parameter was specified.
**OutputFileDialogAction**

See also

*Property:* `OutputFileDialogAction` as `SPYDialogAction`

**Description**
To allow the user to select the output file with a file selection dialog, set this property to `spyDialogUserInput(2)`. If the value stored in `OutputFile` should be taken and no user interaction should occur, use `spyDialogOK(0)`. Default is `spyDialogOK`.

**Errors**
- 2900 The object is no longer valid.
- 2901 Invalid value has been used to set the property. Invalid address for the return parameter was specified.

**OutputFormat**

See also

*Property:* `OutputFormat` as `SPYSchemaDocumentationFormat`

**Description**
Defines the kind of documentation that will be generated: HTML (value=0), MS-Word (value=1), or RTF (value=2). The property gets initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`. The default for the first run is HTML.

**Errors**
- 2900 The object is no longer valid.
- 2901 Invalid value has been used to set the property. Invalid address for the return parameter was specified.

**Parent**

See also

*Property:* `Parent` as `Dialogs` (read-only)

**Description**
Access the parent of the object.

**Errors**
- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

**ShowAnnotations**

See also

*Property:* `ShowAnnotations` as `Boolean`

**Description**
Set this property to `true`, to show the annotations to a type definition in the schema.
documentation. The property is initialized with the value used during the last call to
\texttt{Document.GenerateSchemaDocumentation}. The default for the first run is true.

**Errors**

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

### ShowAttributes

See also

**Property:** \texttt{ShowAttributes} \texttt{as Boolean}

**Description**
Set this property to \texttt{true}, to show the type definitions attributes in the schema documentation.
The property is initialized with the value used during the last call to
\texttt{Document.GenerateSchemaDocumentation}. The default for the first run is true.

**Errors**

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

### ShowChildren

See also

**Property:** \texttt{ShowChildren} \texttt{as Boolean}

**Description**
Set this property to \texttt{true}, to show the children of a type definition as links in the schema
documentation. The property is initialized with the value used during the last call to
\texttt{Document.GenerateSchemaDocumentation}. The default for the first run is true.

**Errors**

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

### ShowDiagram

See also

**Property:** \texttt{ShowDiagram} \texttt{as Boolean}

**Description**
Set this property to \texttt{true}, to show type definitions as diagrams in the schema documentation.
The property is initialized with the value used during the last call to
\texttt{Document.GenerateSchemaDocumentation}. The default for the first run is true.

**Errors**

- 2900 The object is no longer valid.
2901  Invalid address for the return parameter was specified.

ShowEnumerations
See also

*Property:*  *ShowEnumerations*  as Boolean

**Description**
Set this property to *true*, to show the enumerations contained in a type definition in the schema documentation. The property is initialized with the value used during the last call to *Document.GenerateSchemaDocumentation*. The default for the first run is true.

**Errors**
- 2900  The object is no longer valid.
- 2901  Invalid address for the return parameter was specified.

ShowIdentityConstraints
See also

*Property:*  *ShowIdentityConstraints*  as Boolean

**Description**
Set this property to *true*, to show a type definitions identity constraints in the schema documentation. The property is initialized with the value used during the last call to *Document.GenerateSchemaDocumentation*. The default for the first run is true.

**Errors**
- 2900  The object is no longer valid.
- 2901  Invalid address for the return parameter was specified.

ShowNamespace
See also

*Property:*  *ShowNamespace*  as Boolean

**Description**
Set this property to *true*, to show the namespace of type definitions in the schema documentation. The property is initialized with the value used during the last call to *Document.GenerateSchemaDocumentation*. The default for the first run is true.

**Errors**
- 2900  The object is no longer valid.
- 2901  Invalid address for the return parameter was specified.
ShowPatterns

See also

Property: ShowPatterns as Boolean

Description
Set this property to true, to show the patterns of a type definition in the schema documentation. The property is initialized with the value used during the last call to Document.GenerateSchemaDocumentation. The default for the first run is true.

Errors
- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowProgressBar

See also

Property: ShowProgressBar as Boolean

Description
Set this property to true, to make the window showing the document generation progress visible. Use false, to hide it. Default is false.

Errors
- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowProperties

See also

Property: ShowProperties as Boolean

Description
Set this property to true, to show the type definition properties in the schema documentation. The property is initialized with the value used during the last call to Document.GenerateSchemaDocumentation. The default for the first run is true.

Errors
- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowResult

See also

Property: ShowResult as Boolean

Description
Set this property to `true`, to automatically open the resulting document when generation was successful. HTML documentation will be opened in XMLSpy. To show Word documentation, MS-Word will be started. The property gets initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`. The default for the first run is true.

**Errors**
- 2900  The object is no longer valid.
- 2901  Invalid address for the return parameter was specified.

**ShowSingleFacets**

**See also**

**Property:** `ShowSingleFacets` as `Boolean`

**Description**
Set this property to `true`, to show the facets of a type definition in the schema documentation. The property is initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`. The default for the first run is true.

**Errors**
- 2900  The object is no longer valid.
- 2901  Invalid address for the return parameter was specified.

**ShowSourceCode**

**See also**

**Property:** `ShowSourceCode` as `Boolean`

**Description**
Set this property to `true`, to show the XML source code for type definitions in the schema documentation. The property is initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`. The default for the first run is true.

**Errors**
- 2900  The object is no longer valid.
- 2901  Invalid address for the return parameter was specified.

**ShowType**

**See also**

**Property:** `ShowType` as `Boolean`

**Description**
Set this property to `true`, to show the type of type definitions in the schema documentation. The property is initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`. The default for the first run is true.

**Errors**
ShowUsedBy

See also

Property: ShowUsedBy as Boolean

Description
Set this property to true, to show the used-by relation for type definitions in the schema documentation. The property is initialized with the value used during the last call to Document.GenerateSchemaDocumentation. The default for the first run is true.

Errors
2900 The object is no longer valid.
2901 Invalid address for the return parameter was specified.

SPSFile

See also

Property: SPSFile as String

Description
Full path and name of the SPS file that will be used to generate the documentation.

Errors
2900 The object is no longer valid.
2901 Invalid address for the return parameter was specified.

UseFixedDesign

See also

Property: UseFixedDesign as Boolean

Description
Specifies whether the documentation should be created with a fixed design or with a design specified by a SPS file (which requires StyleVision).

Errors
2900 The object is no longer valid.
2901 Invalid address for the return parameter was specified.

3.2.24 SpyProject

See also

Methods
CloseProject
SaveProject
SaveProjectAs

Properties
RootItems
ProjectFile

Description
SpyProject Class

CloseProject
See also

Declaration: `CloseProject(bDiscardChanges as Boolean, bCloseFiles as Boolean, bDialog as Boolean)`

Parameters
bDiscardChanges
Set `bDiscardChanges` to FALSE if you want to save the changes of the open project files and the project.

bCloseFiles
Set `bCloseFiles` to TRUE to close all open project files.

bDialog
Show dialogs for user input.

Description
CloseProject closes the current project.

ProjectFile
See also

Declaration: `ProjectFile` as String

Description
Path and filename of the project.

RootItems
See also

Declaration: `RootItems` as `SpyProjectItems`

Description
Root level of collection of project items.
SaveProject
See also

Declaration: SaveProject

Description
SaveProject saves the current project.

SaveProjectAs
See also

Declaration: SaveProjectAs (strPath as String, bDialog as Boolean)

Parameters
strPath
Full path with file name of new project file.

bDialog
If bDialog is TRUE, a file-dialog will be displayed.

Description
SaveProjectAs stores the project data into a new location.

3.2.25 SpyProjectItem
See also

Methods
Open

Properties
ChildItems
ParentItem
FileExtensions
ItemType
Name
Path
ValidateWith
XMLForXSLTransformation
XSLForXMLTransformation
XSLTransformationFileExtension
XSLTransformationFolder

Description
SpyProjectItem Class

ChildItems
See also

Declaration: ChildItems as SpyProjectItems
Description
If the item is a folder, ChildItems is the collection of the folder content.

FileExtensions
See also

Declaration: FileExtensions as String

Description
Used to set the file extensions if the project item is a folder.

ItemType
See also

Declaration: ItemType as SPYProjectItemTypes

Description
This property is read-only.

Name
See also

Declaration: Name as String

Description
Name of the project item. This property is read-only.

Open
See also

Declaration: Open as Document

Return Value
The project item opened as document.

Description
Opens the project item.

ParentItem
See also

Declaration: ParentItem as SpyProjectItem

Description
Parent item of the current project item. Can be NULL (Nothing) if the project item is a top-level item.
Path
See also

*Declaration:* Path as String

**Description**
Path of project item. This property is read-only.

ValidateWith
See also

*Declaration:* ValidateWith as String

**Description**
Used to set the schema/DTD for validation.

XMLForXSLTransformation
See also

*Declaration:* XMLForXSLTransformation as String

**Description**
Used to set the XML for XSL transformation.

XSLForXMLTransformation
See also

*Declaration:* XSLForXMLTransformation as String

**Description**
Used to set the XSL for XML transformation.

XSLTransformationFileExtension
See also

*Declaration:* XSLTransformationFileExtension as String

**Description**
Used to set the file extension for XSL transformation output files.

XSLTransformationFolder
See also

*Declaration:* XSLTransformationFolder as String

**Description**
Used to set the destination folder for XSL transformation output files.
3.2.26 SpyProjectItems

See also

Methods
AddFile
AddFolder
AddURL
RemoveItem

Properties
Count
Item

Description
SpyProjectItems Class

AddFile
See also

Declaration: AddFile (strPath as String)

Parameters
strPath
Full path with file name of new project item

Description
The method adds a new file to the collection of project items.

AddFolder
See also

Declaration: AddFolder (strName as String)

Parameters
strName
Name of the new folder.

Description
The method AddFolder adds a folder with the name strName to the collection of project items.

AddURL
See also

Declaration: AddURL (strURL as String, nURLType as SPYURLTypes, strUser as String, strPassword as String, bSave as Boolean)

Description
strURL
URL to open as document.

nURLType
Type of document to open. Set to -1 for auto detection.

strUser
Name of the user if required. Can be empty.

strPassword
Password for authentication. Can be empty.

bSave
Save user and password information.

Description
The method adds an URL item to the project collection.

Count
See also

Declaration: Count as long

Description
This property gets the count of project items in the collection. The property is read-only.

Item
See also

Declaration: Item (n as long) as SpyProjectItem

Description
Retrieves the n-th element of the collection of project items. The first item has index 1.

RemoveItem
See also

Declaration: RemoveItem (pItem as SpyProjectItem)

Description
RemoveItem deletes the item pItem from the collection of project items.

3.2.27 TextImportExportSettings
See also

Properties for import only
ImportFile

Properties for export only
Properties for import and export

DestinationFolder
FieldExtension
CommentIncluded
RemoveDelimiter
RemoveNewline

Description
TextImportExportSettings contains options common to text import and export functions.

CommentIncluded
See also

Property: CommentIncluded as Boolean

Description
This property tells whether additional comments are added to the generated text file. Default is true. This property is used only when exporting to text files.

DestinationFolder
See also

Property: DestinationFolder as String

Description
The property DestinationFolder sets the folder where the created files are saved during text export.

EnclosingCharacter
See also

Property: EnclosingCharacter as SPYTextEnclosing

Description
This property defines the character that encloses all field values for import and export. Default is spyNoEnclosing.

Encoding
See also

Property: Encoding as String

Description
The property `Encoding` sets the character encoding for the text files for importing and exporting.

**EncodingByteOrder**

See also

*Property:* `EncodingByteOrder` as `SPYEncodingByteOrder`

**Description**
The property `EncodingByteOrder` sets the byte order for Unicode characters. Default is `spyNONE`.

**FieldDelimiter**

See also

*Property:* `FieldDelimiter` as `SPYTextDelimiters`

**Description**
The property `FieldDelimiter` defines the delimiter between the fields during import and export. Default is `spyTabulator`.

**FileExtension**

See also

*Property:* `FileExtension` as `String`

**Description**
This property sets the file extension for files created on text export.

**HeaderRow**

See also

*Property:* `HeaderRow` as `Boolean`

**Description**
The property `HeaderRow` is used during import and export. Set `HeaderRow` true on import, if the first line of the text file contains the names of the columns. Set `HeaderRow` true on export, if the first line in the created text files should contain the name of the columns. Default value is true.

**ImportFile**

See also

*Property:* `ImportFile` as `String`

**Description**
This property is used to set the text file for import. The string has to be a full qualified path. See also `Import and Export`. 
RemoveDelimiter

See also

Property: RemoveDelimiter as Boolean

Description
The property RemoveDelimiter defines whether characters in the text that are equal to the delimiter character are removed. Default is false. This property is used only when exporting to text files.

RemoveNewline

See also

Property: RemoveNewline as Boolean

Description
The property RemoveNewline defines whether newline characters in the text are removed. Default is false. This property is used only when exporting to text files.

3.2.28 TextView

See also

Properties and Methods

Application
Parent

LineFromPosition
PositionFromLine
LineLength
SelText
GetRangeText
ReplaceText
MoveCaret
GoToLineChar
SelectText
SelectionStart
SelectionEnd
Text
LineCount
Length

Description
Events

**OnBeforeShowSuggestions**

See also

**Event: OnBeforeShowSuggestions**() as Boolean

Description
This event gets fired before a suggestion window is shown. The Document property Suggestions contains a string array that is recommended to the user. It is possible to modify the displayed recommendations during this event. Before doing so you have to assign an empty array to the Suggestions property. The best location for this is the OnDocumentOpened event. To prevent the suggestion window to show up return false and true to continue its display.

Examples
Given below are examples of how this event can be scripted.

XMLSpy scripting environment - VBScript:
Function On_BeforeShowSuggestions ()
End Function

XMLSpy scripting environment - JScript:
function On_BeforeShowSuggestions ()
{
}

XMLSpy IDE Plugin:
IXMLSpyPlugIn.OnEvent (33, ... ) // nEventId = 33

**OnChar**

See also

**Event: OnChar**(nChar as Long, bExistSuggestion as Boolean) as Boolean

Description
This event gets fired on each key stroke. The parameter nChar is the key that was pressed and bExistSuggestion tells whether a XMLSpy generated suggestions window is displayed after this key. The Document property Suggestions contains a string array that is recommended to the user. It is possible to modify the displayed recommendations during this event. Before doing so you have to assign an empty array to the Suggestions property. The best location for this is the OnDocumentOpened event. To prevent the suggestion window to show up return false and true to continue its display.

It is also possible to create a new suggestions window when none is provided by XMLSpy. Set the Document property Suggestions to a string array with your recommendations and return true. This event is fired before the OnBeforeShowSuggestions event. If you prevent to show the suggestion window by returning false then OnBeforeShowSuggestions is not fired.

Examples
Given below are examples of how this event can be scripted.

**XMLSpy scripting environment - VBScript:**

```vbnet
Function On_Char(nChar, bExistSuggestions)  
End Function
```

**XMLSpy scripting environment - JScript:**

```javascript
function On_Char(nChar, bExistSuggestions)  
{
}
```

**XMLSpy IDE Plugin:**

```csharp
IXMLSpyPlugIn.OnEvent (35, ...) // nEventId = 35
```

### Application

**Property:** *Application as Application* (read-only)

**Description**

Access the XMLSpy application object.

**Errors**

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

### GetRangeText

**Method:** *GetRangeText(nStart as Long, nEnd as Long) as String*

**Description**

Returns the text in the specified range.

**Errors**

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

### GoToLineChar

**Method:** *GoToLineChar(nLine as Long, nChar as Long)*

**Description**

Moves the caret to the specified line and character position.

**Errors**

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

### Length

**Property:** *Length as Long*
**Description**
Returns the character count of the document.

**Errors**
- 3900  The object is no longer valid.
- 3901  Invalid address for the return parameter was specified.

**LineCount**
*Property:* LineCount as Long

**Description**
Returns the number of lines in the document.

**Errors**
- 3900  The object is no longer valid.
- 3901  Invalid address for the return parameter was specified.

**LineFromPosition**
*Method:* LineFromPosition(nCharPos as Long) as Long

**Description**
Returns the line number of the character position.

**Errors**
- 3900  The object is no longer valid.
- 3901  Invalid address for the return parameter was specified.

**LineLength**
*Method:* LineLength(nLine as Long) as Long

**Description**
Returns the length of the line.

**Errors**
- 3900  The object is no longer valid.
- 3901  Invalid address for the return parameter was specified.

**MoveCaret**
*Method:* MoveCaret(nDiff as Long)

**Description**
Moves the caret nDiff characters.

**Errors**
- 3900  The object is no longer valid.
- 3901  Invalid address for the return parameter was specified.
Parent

**Property:** Parent as Document (read-only)

**Description**
Access the parent of the object.

**Errors**
- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

PositionFromLine

**Method:** PositionFromLine(nLine as Long) as Long

**Description**
Returns the start position of the line.

**Errors**
- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

ReplaceText

**Method:** ReplaceText(nPosFrom as Long, nPosTill as Long, sText as String)

**Description**
Replaces the text in the specified range.

**Errors**
- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

SelectionEnd

**Property:** SelectionEnd as Long

**Description**
Returns/sets the text selection end position.

**Errors**
- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

SelectionStart

**Property:** SelectionStart as Long

**Description**
Returns/sets the text selection start position.

**Errors**
- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.
SelectText

**Method:** SelectText(nPosFrom as Long, nPosTill as Long)

**Description**
Selects the text in the specified range.

**Errors**
- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

SelText

**Property:** SelText as String

**Description**
Returns/sets the selected text.

**Errors**
- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

Text

**Property:** Text as String

**Description**
Returns/sets the document text.

**Errors**
- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

3.2.29 XMLData

**See also**

**Properties**
- Kind
- Name
- TextValue
- HasChildren
- MayHaveChildren
- Parent

**Methods**
- GetFirstChild
- GetNextChild
- GetCurrentChild
- InsertChild
- InsertChildAfter
InsertChildBefore
AppendChild
EraseAllChildren
EraseChild
EraseCurrentChild
IsSameNode
CountChildren
CountChildrenKind
GetChild
GetChildAttribute
GetChildElement
GetChildKind
GetNamespacePrefixForURI
HasChildrenKind
SetTextValueXMLEncoded

Description
The XMLData interface provides direct XML-level access to a document. You can read and directly modify the XML representation of the document. However, please, note the following restrictions:

- The XMLData representation is only valid when the document is shown in grid view or authentic view.
- When in authentic view, additional XMLData elements are automatically inserted as parents of each visible document element. Typically this is an XMLData of kind spyXMLDataElement with the Name property set to 'Text'.
- When you use the XMLData interface while in a different view mode you will not receive errors, but changes are not reflected to the view and might get lost during the next view switch.

Note also:

- Setting a new text value for an XML element is possible if the element does not have non-text children. A text value can be set even if the element has attributes.
- When setting a new text value for an XML element which has more than one text child, the latter will be deleted and replaced by one new text child.
- When reading the text value of an XML element which has more than one text child, only the value of the first text child will be returned.

Objects of this class represent the different atomic parts of an XML document. See the enumeration type SPYXMLDataKind for the available part types. Each part knows its children, thus forming a XMLData tree with Document.RootElement at its top. To get the top element of the document content - ignoring the XML header - use Document.DataRoot. For an examples on how to traverse the XMLData tree, see GetNextChild.
AppendChild

See also

**Declaration:** `AppendChild (pNewData as XMLData)`

**Description**
`AppendChild` appends `pNewData` as last child to the `XMLData` object.

**Errors**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The XMLData object is no longer valid.</td>
</tr>
<tr>
<td>1505</td>
<td>Invalid XMLData kind was specified.</td>
</tr>
<tr>
<td>1506</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
<tr>
<td>1507</td>
<td>Element cannot have Children</td>
</tr>
<tr>
<td>1512</td>
<td>Cyclic insertion - new data element is already part of document</td>
</tr>
<tr>
<td>1514</td>
<td>Invalid XMLData kind was specified for this position.</td>
</tr>
<tr>
<td>1900</td>
<td>Document must not be modified</td>
</tr>
</tbody>
</table>

**Example**

```
Dim objCurrentParent As XMLData
Dim objNewChild As XMLData

Set objNewChild = objSpy.ActiveDocument.CreateChild(spyXMLDataElement)

objCurrentParent.AppendChild objNewChild

Set objNewChild = Nothing
```

CountChildren

See also

**Declaration:** `CountChildren` as long

**Description**
`CountChildren` gets the number of children.

Available with TypeLibrary version 1.5

**Errors**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>The XMLData object is no longer valid.</td>
</tr>
</tbody>
</table>

CountChildrenKind

See also

**Declaration:** `CountChildrenKind (nKind as SPYXMLDataKind)` as long

**Description**
`CountChildrenKind` gets the number of children of the specific kind.
Available with TypeLibrary version 1.5

Errors

1500  The XMLData object is no longer valid.

EraseAllChildren

See also

Declaration: EraseAllChildren

Description

EraseAllChildren deletes all associated children of the XMLData object.

Errors

1500  The XMLData object is no longer valid.
1900  Document must not be modified

Example

The sample erases all elements of the active document.

Dim objCurrentParent As XMLData
objCurrentParent.EraseAllChildren

EraseChild

Method: EraseChild (Child as XMLData)

Description

Deletes the given child node.

Errors

1500  Invalid object.
1506  Invalid input xml
1510  Invalid parameter.

EraseCurrentChild

See also

Declaration: EraseCurrentChild

Description

EraseCurrentChild deletes the current XMLData child object. Before you call EraseCurrentChild you must initialize an internal iterator with XMLData.GetFirstChild. After deleting the current child, EraseCurrentChild increments the internal iterator of the XMLData element. No error is returned when the last child gets erased and the iterator is moved
past the end of the child list. The next call to EraseCurrentChild however, will return error 1503.

Errors
- **1500** The XMLData object is no longer valid.
- **1503** No iterator is initialized for this XMLData object, or the iterator points past the last child.
- **1900** Document must not be modified

Examples
// ---------------------------------------
// XMLSpy scripting environment - JScript
// erase all children of XMLData
// ---------------------------------------
// let's get an XMLData element, we assume that the cursor selects the parent of a list in grid view

// the following line would be shorter, of course
// objList.EraseAllChildren();

// but we want to demonstrate the usage of EraseCurrentChild
if ((objList != null) && (objList.HasChildren))
{
    try
    {
        objEle = objList.GetFirstChild(-1);
        while (objEle != null)
        {
            objList.EraseCurrentChild();
            // no need to call GetNextChild
        }
    }
    catch (err)
    // 1503 - we reached end of child list
    {
        if ((err.number & 0xffff) != 1503) throw (err); } }

GetChild

See also

**Declaration:** GetChild(position as long) as XMLData

Return Value
Returns an XML element as XMLData object.

Description
GetChild() returns a reference to the child at the given index (zero-based).

Available with TypeLibrary version 1.5

Errors
- **1500** The XMLData object is no longer valid.
- **1510** Invalid address for the return parameter was specified.
**GetChildAttribute**

*Method:* `GetChildAttribute` (strName as string) child as XMLData object *(NULL on error)*

*Description*
Retrieves the attribute having the given name.

*Errors*
- 1500  Invalid object.
- 1510  Invalid parameter.

**GetChildElement**

*Method:* `GetChildElement` (strName as string, nIndex as long) child as XMLData object *(NULL on error)*

*Description*
Retrieves the Nth child element with the given name.

*Errors*
- 1500  Invalid object.
- 1510  Invalid parameter.

**GetChildKind**

*See also*

*Declaration:* `GetChildKind` *(position as long, nKind as `SPYXMLDataKind`) as XMLData*

*Return Value*
Returns an XML element as XMLData object.

*Description*
`GetChildKind()` returns a reference to a child of this kind at the given index (zero-based). The position parameter is relative to the number of children of the specified kind and not to all children of the object.

Available with TypeLibrary version 1.5

*Errors*
- 1500  The XMLData object is no longer valid.
- 1510  Invalid address for the return parameter was specified.
GetCurrentChild

See also

Declaration: GetCurrentChild as XMLData

Return Value
Returns an XML element as XMLData object.

Description
GetCurrentChild gets the current child. Before you call GetCurrentChild you must initialize an internal iterator with XMLData.GetFirstChild.

Errors
1500 The XMLData object is no longer valid.
1503 No iterator is initialized for this XMLData object.
1510 Invalid address for the return parameter was specified.

GetFirstChild

See also

Declaration: GetFirstChild (nKind as SPYXMLDataKind) as XMLData

Return Value
Returns an XML element as XMLData object.

Description
GetFirstChild initializes a new iterator and returns the first child. Set nKind = -1 to get an iterator for all kinds of children.
REMARK: The iterator is stored inside the XMLData object and gets destroyed when the XMLData object gets destroyed. Be sure to keep a reference to this object as long as you want to use GetCurrentChild, GetNextChild or EraseCurrentChild.

Errors
1500 The XMLData object is no longer valid.
1501 Invalid XMLData kind was specified.
1504 Element has no children of specified kind.
1510 Invalid address for the return parameter was specified.

Example
See the example at XMLData.GetNextChild.

GetNamespacePrefixForURI

Method: GetNamespacePrefixForURI (strURI as string) strNS as string

Description
Returns the namespace prefix of the supplied URI.
Errors
1500 Invalid object.
1510 Invalid parameter.

GetNextChild
See also

Declaration: GetNextChild as XMLData

Return Value
Returns an XML element as XMLData object.

Description
GetNextChild steps to the next child of this element. Before you call GetNextChild you must initialize an internal iterator with XMLData.GetFirstChild.

Check for the last child of the element as shown in the sample below.

Errors
1500 The XMLData object is no longer valid.
1503 No iterator is initialized for this XMLData object.
1510 Invalid address for the return parameter was specified.

Examples
' ----------------------------------------------
' VBA code snippet - iterate XMLData children
' ----------------------------------------------
On Error Resume Next
Set objParent = objSpy.ActiveDocument.RootElement

'get elements of all kinds
Set objCurrentChild = objParent.GetFirstChild(-1)

Do
' do something useful with the child

' step to next child
Set objCurrentChild = objParent.GetNextChild
Loop Until (Err.Number - vbObjectError = 1503)

// ---------------------------------------
// XMLSpy scripting environment - JScript
// iterate through children of XMLData
// ---------------------------------------
try
{
  var objXMLData = ... // initialize somehow
  var objChild = objXMLData.GetFirstChild(-1);
while (true) {
    // do something useful with objChild
    objChild = objXMLData.GetNextChild();
}
} catch (err) {
    if ((err.number & 0xffff) == 1504) {
        // element has no children
    } else if ((err.number & 0xffff) == 1503) {
        // last child reached
    } else {
        throw (err);
    }
}

GetTextValueXMLDecoded

Method: GetTextValueXMLDecoded () as string

Description
Gets the decoded text value of the XML.

Errors
- 1500 Invalid object.
- 1510 Invalid parameter.

HasChildren

See also

Declaration: HasChildren as Boolean

Description
The property is true if the object is the parent of other XMLData objects. This property is read-only.

Errors
- 1500 The XMLData object is no longer valid.
- 1510 Invalid address for the return parameter was specified.

HasChildrenKind

See also

Declaration: HasChildrenKind (nKind as SPYXMLDataKind) as Boolean

Description
The method returns true if the object is the parent of other XMLData objects of the specific kind.
Available with TypeLibrary version 1.5

**Errors**

- 1500  The XMLData object is no longer valid.
- 1510  Invalid address for the return parameter was specified.

**InsertChild**

**See also**

*Declaration:* `InsertChild(pNewData as XMLData)`

**Description**

`InsertChild` inserts the new child before the current child (see also `XMLData.GetFirstChild`, `XMLData.GetNextChild` to set the current child).

**Errors**

- 1500  The XMLData object is no longer valid.
- 1503  No iterator is initialized for this XMLData object.
- 1505  Invalid XMLData kind was specified.
- 1506  Invalid address for the return parameter was specified.
- 1507  Element cannot have Children
- 1512  Cyclic insertion - new data element is already part of document
- 1514  Invalid XMLData kind was specified for this position.
- 1900  Document must not be modified

**InsertChildAfter**

*Method:* `InsertChildBefore(Node as XMLData, NewData as XMLData)`

**Description**

Inserts a new XML node (supplied with the second parameter) after the specified node (first parameter).

**Errors**

- 1500  Invalid object.
- 1506  Invalid input xml
- 1507  No children allowed
- 1510  Invalid parameter.
- 1512  Child is already added
- 1514  Invalid kind at position

**InsertChildBefore**

*Method:* `InsertChildBefore(Node as XMLData, NewData as XMLData)`

**Description**

Inserts a new XML node (supplied with the second parameter) before the specified node (first parameter).
Errors
1500 Invalid object.
1506 Invalid input xml
1507 No children allowed
1510 Invalid parameter.
1512 Child is already added
1514 Invalid kind at position

IsSameNode
See also

Declaration: IsSameNode (pNodeToCompare as XMLData) as Boolean

Description
Returns true if pNodeToCompare references the same node as the object itself.

Errors
1500 The XMLData object is no longer valid.
1506 Invalid address for the return parameter was specified.

Kind
See also

Declaration: Kind as SPYXMLDataKind

Description
Kind of this XMLData object. This property is read-only.

Errors
1500 The XMLData object is no longer valid.
1510 Invalid address for the return parameter was specified.

MayHaveChildren
See also

Declaration: MayHaveChildren as Boolean

Description
Indicates whether it is allowed to add children to this XMLData object. This property is read-only.

Errors
1500 The XMLData object is no longer valid.
1510 Invalid address for the return parameter was specified.
Name
See also

*Declaration:* Name as String

Description
Used to modify and to get the name of the XMLData object.

Errors
1500  The XMLData object is no longer valid.
1510  Invalid address for the return parameter was specified.

Parent
See also

*Declaration:* Parent as XMLData

Return value
Parent as XMLData object. Nothing (or NULL) if there is no parent element.

Description
Parent of this element. This property is read-only.

Errors
1500  The XMLData object is no longer valid.
1510  Invalid address for the return parameter was specified.

SetTextValueXMLEncoded

*Method:* SetTextValueXMLEncoded (strVal as String)

Description
Sets the encoded text value of the XML.

Errors
1500  Invalid object.
1513  Modification not allowed.

TextValue
See also

*Declaration:* TextValue as String

Description
Used to modify and to get the text value of this XMLData object.

Errors
1500  The XMLData object is no longer valid.
1510   Invalid address for the return parameter was specified.
3.3 Interfaces (obsolete)

Interfaces contained in this book are obsolete. It is recommended to migrate your applications to the new interfaces. See the different properties and methods in this book for migration hints.

3.3.1 AuthenticEvent (obsolete)

<table>
<thead>
<tr>
<th>Superseded by</th>
<th>AuthenticView and AuthenticRange</th>
</tr>
</thead>
</table>

The DocEditView object is renamed to OldAuthenticView.
DocEditSelection is renamed to AuthenticSelection.
DocEditEvent is renamed to AuthenticEvent.
DocEditDataTransfer is renamed to AuthenticDataTransfer.

Their usage - except for AuthenticDataTransfer - is no longer recommended. We will continue to support existing functionality for a yet undefined period of time but no new features will be added to these interface. All functionality available up to now in DocEditView, DocEditSelection, DocEditEvent and DocEditDataTransfer is now available via AuthenticView, AuthenticRange and AuthenticDataTransfer. Many new features have been added.

For examples on migrating from DocEdit to Authentic see the description of the different methods and properties of the different DocEdit objects.

See also

Properties

- altKey
- altLeft
- ctrlKey
- ctrlLeft
- shiftKey
- shiftLeft
- keyCode
- repeat
- button
- clientX
- clientY
- dataTransfer
- srcElement
- fromElement
- propertyName
- cancelBubble
- returnValue
type

Description
DocEditEvent interface.

altKey (obsolete)

Superseded by parameters to
  AuthenticView.OnKeyboardEvent (On_AuthenticView_KeyPressed)
  AuthenticView.OnMouseEvent (On_AuthenticView_MouseEvent)
  AuthenticView.OnDragOver (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to
the different event handler functions to simplify data access. The event object will be supported
for a not yet defined period of time for compatibility reasons. No improvements are planned. It is
highly recommended to migrate to the new event handler functions.

// ----- XMLSpy scripting environment - javascript sample -----
// instead of:
// function On_DocEditKeyPressed ()
// {
//     if (Application.ActiveDocument.DocEditView.event.altKey ||
//        MsgBox ("alt key is down");
// }
// use now:
function On_AuthenticView_KeyPressed (SPYKeyEvent i_eKeyEvent,
            long i_nKeyCode,
            SPYVirtualKeyMask i_nVirtualKeyStatus)
{
    if (i_nVirtualKeyStatus & spyAltKeyMask)
        MsgBox ("alt key is down");
}

See also

Declaration: altKey as Boolean

Description
True if the right ALT key is pressed.
altLeft (obsolete)

Superseded by parameters to

- AuthenticView.OnKeyboardEvent (On_AuthenticView_KeyboardEvent)
- AuthenticView.OnMouseEvent (On_AuthenticView_MouseEvent)
- AuthenticView.OnDragOver (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

```javascript
// ----- XMLSpy scripting environment - javascript sample -----
// instead of:
// function On_DocEditKeyDown ()
// {
//   if (Application.ActiveDocument.DocEditView.event.altKey ||
//      MsgBox ("alt key is down");
// }
// use now:
function On_AuthenticView_KeyDown (SPYKeyEvent i_eKeyEvent, long i_nKeyCode, SPYVirtualKeyMask i_nVirtualKeyStatus)
{
   if (i_nVirtualKeyStatus & spyAltKeyMask)
      MsgBox ("alt key is down");
}
```

See also

**Declaration:** altLeft as Boolean

**Description**

True if the left ALT key is pressed.
button (obsolete)

Superseded by parameters to

AuthenticView.OnMouseEvent (On_AuthenticView_MouseEvent)
AuthenticView.OnDragOver (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

```javascript
// ----- XMLSpy scripting environment - javascript sample ----- 
// instead of:
// function On_DocEditButtonDown ()
// {
//     if (Application.ActiveDocument.DocEditView.event.button == 1)
//         MsgBox ("left mouse button down detected");
// }
// use now:
function On_AuthenticView_MouseEvent (long i_nXPos, long i_nYPos, SPYMouseEvent i_eMouseEvent, IAuthenticRange *i_ipRange)
{
    if (i_eMouseEvent & spyLeftButtonDownMask)
        MsgBox ("left mouse button down detected");
}
```

See also

Declaration: button as long

Description

Specifies which mouse button is pressed:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No button is pressed.</td>
</tr>
<tr>
<td>1</td>
<td>Left button is pressed.</td>
</tr>
<tr>
<td>2</td>
<td>Right button is pressed.</td>
</tr>
<tr>
<td>3</td>
<td>Left and right buttons are both pressed.</td>
</tr>
<tr>
<td>4</td>
<td>Middle button is pressed.</td>
</tr>
<tr>
<td>5</td>
<td>Left and middle buttons both are pressed.</td>
</tr>
<tr>
<td>6</td>
<td>Right and middle buttons are both pressed.</td>
</tr>
<tr>
<td>7</td>
<td>All three buttons are pressed.</td>
</tr>
</tbody>
</table>
cancelBubble (obsolete)

Superseded by the boolean return value of following event handler functions

- **AuthenticView.OnKeyboardEvent** (On_AuthenticView_KeyPressed)
- **AuthenticView.OnMouseEvent** (On_AuthenticView_MouseEvent)
- **AuthenticView.OnDragOver** (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

Returning *true* from an event handler function signals that the event has been handled and normal event handling should be aborted.

```javascript
// ----- XMLSpy scripting environment - javascript sample -----  
// instead of:  
// function On_DocEditKeyPressed ()  
// {  
//     if (Application.ActiveDocument.DocEditView.event.keyCode == 0x20)  
//     {  
//         // cancel key processing, swallow spaces :-)  
//         Application.ActiveDocument.DocEditView.event.cancelBubble = true;  
//     }  
// }  
// use now:  
function On_AuthenticView_KeyPressed (SPYKeyEvent i_eKeyEvent, long i_nKeyCode, SPYVirtualKeyMask i_nVirtualKeyStatus)  
{  
    if (i_nKeyCode == 0x20)  
        return true; // cancel key processing, swallow spaces :-)  
}
```

See also

*Declaration:* cancelBubble as Boolean

*Description*

Set cancelBubble to TRUE if the default event handler should not be called.
clientX (obsolete)

**Superseded by parameters to**

- `AuthenticView.OnMouseEvent` (On_AuthenticView_MouseEvent)
- `AuthenticView.OnBeforeDrop` (On_AuthenticView_BeforeDrop)
- `AuthenticView.OnDragOver` (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

```javascript
// ----- XMLSpy scripting environment - javascript sample ----- 
// instead of: 
// function On_DocEditMouseMove ()
// {
//             "/" + Application.ActiveDocument.DocEditView.event.clientY);
// }
// use now: 
function On_AuthenticView_MouseEvent (long i_nXPos, long i_nYPos, SPYMouseEvent i_eMouseEvent, IAuthenticRange *i_ipRange) {
    if (i_eMouseEvent & spyMouseMoveMask)
        MsgBox ("moving over " + i_nXPos + "/" + i_nYPos);
}
```

**See also**

*Declaration:* `clientX` as long

**Description**

X value of the current mouse position in client coordinates.
clientY (obsolete)

Superseded by parameters to

- `AuthenticView.OnMouseEvent` (On_AuthenticView_MouseEvent)
- `AuthenticView.OnBeforeDrop` (On_AuthenticView_BeforeDrop)
- `AuthenticView.OnDragOver` (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

// ----- XMLSpy scripting environment - javascript sample -----  
// instead of: 
// function On_DocEditMouseMove () 
// { 
//             "/" + Application.ActiveDocument.DocEditView.event.clientY); 
// } 
// use now:  
function On_AuthenticView_MouseEvent (long i_nXPos, long i_nYPos, SPYMouseEvent i_eMouseEvent, IAuthenticRange *i_ipRange)  
{ 
    if (i_eMouseEvent & spyMouseMoveMask)  
        MsgBox ("moving over " + i_nXPos + "/" + i_nYPos); 
}

See also

Declaration: `clientY` as long

Description

Y value of the current mouse position in client coordinates.
ctrlKey (obsolete)

Superseded by parameters to
- **AuthenticView.OnKeyboardEvent** (On_AuthenticView_KeyPressed)
- **AuthenticView.OnMouseEvent** (On_AuthenticView_MouseEvent)
- **AuthenticView.OnDragOver** (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

```javascript
// ----- XMLSpy scripting environment - javascript sample -----
// instead of:
// function On_DocEditMouseMove ()
// {
//     if (Application.ActiveDocument.DocEditView.event.ctrlKey ||
//        MsgBox ("control key is down");
// }
// use now:
function On_AuthenticView_MouseMove (long i_nXPos, long i_nYPos, SPYMouseEvent i_eMouseEvent, IAuthenticRange *i_ipRange)
{
    if (i_eMouseEvent & spyCtrlKeyMask)
        MsgBox ("control key is down");
}
```

See also

*Declaration:* **ctrlKey** as Boolean

*Description*

True if the right CTRL key is pressed.
ctrlLeft (obsolescent)

Superseded by parameters to

- AuthenticView.OnKeyboardEvent (On_AuthenticView_KeyPressed)
- AuthenticView.OnMouseEvent (On_AuthenticView_MouseEvent)
- AuthenticView.OnDragOver (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

// ----- XMLSpy scripting environment - javascript sample -----  
// instead of:  
// function On_DocEditMouseMove ()  
// {  
//     if (Application.ActiveDocument.DocEditView.event.ctrlKey ||  
//        MsgBox ("control key is down");  
// }  
// use now:  
function On_AuthenticView_MouseMove (long i_nXPos, long i_nYPos, SPYMouseEvent i_eMouseEvent, IAuthenticRange *i_ipRange)  
{  
    if (i_eMouseEvent & spyCtrlKeyMask)  
        MsgBox ("control key is down");  
}

See also

Declaration: ctrlLeft as Boolean

Description
True if the left CTRL key is pressed.
dataTransfer (obsolete)

Superseded by parameters to

**AuthenticView.OnBeforeDrop** (On_AuthenticView_BeforeDrop)
**AuthenticView.OnDragOver** (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to
the different event handler functions to simplify data access. The event object will be supported
for a not yet defined period of time for compatibility reasons. No improvements are planned. It is
highly recommended to migrate to the new event handler functions.

```javascript
// ----- XMLSpy scripting environment - javascript sample ----- 
// instead of:
// function On_DocEditDrop ()
// {
//     if (Application.ActiveDocument.DocEditView.event.dataTransfer != null)
//         if (! Application.ActiveDocument.DocEditView.event.dataTransfer.ownDrag)
//             // cancel key processing, don't drop foreign objects :-)
//             Application.ActiveDocument.DocEditView.event.cancelBubble = true;
//     }
// use now:
function On_AuthenticView_BeforeDrop (long i_nXPos, long i_nYPos, IAuthenticRange *i_ipRange, IAuthenticDataTransfer *i_ipData)
{
    if (i_ipRange != null)
        if (! i_ipRange.ownDrag)
            return true; // cancel key processing, don't drop foreign objects :-)

    return false;
}
```

See also

*Declaration:* `dataTransfer` as Variant

*Description*

Property `dataTransfer`. 
fromElement (obsolete)

Not supported

See also

Declaration: fromElement as Variant (not supported)

Description
Currently no event sets this property.

keyCode (obsolete)

Superseded by a parameter to AuthenticView.OnKeyboardEvent
(On_AuthenticView_KeyPressed)

The event object that holds the information of the last event is now replaced by parameters to
the different event handler functions to simplify data access. The event object will be supported
for a not yet defined period of time for compatibility reasons. No improvements are planned. It is
highly recommended to migrate to the new event handler functions.

// ----- XMLSpy scripting environment - javascript sample -----
// instead of:
// function On_DocEditKeyPressed ()
// {
//     if (Application.ActiveDocument.DocEditView.event.keyCode == 0x20)
//     {
//         // cancel key processing, swallow spaces :-)
//         Application.ActiveDocument.DocEditView.event.cancelBubble = true;
//     }
// }
// use now:
function On_AuthenticView_KeyPressed (SPYKeyEvent i_eKeyEvent, long i_nKeyCode,
SPYVirtualKeyMask i_nVirtualKeyStatus)
{
    if (i_nKeyCode == 0x20)
        return true; // cancel key processing, swallow spaces :-)
}

See also

Declaration: keyCode as long

Description
Keycode of the currently pressed key. This property is read-write.
**propertyName (obsolete)**

| Not supported |

See also

Declaration: `propertyName` as String (not supported)

Description
Currently no event sets this property.

**repeat (obsolete)**

| Not supported |

See also

Declaration: `repeat` as Boolean (not supported)

Description
True if the `onkeydown` event is repeated.

**returnValue (obsolete)**

| No longer supported |

See also

Declaration: `returnValue` as Variant

Description
Use `returnValue` to set a return value for your event handler.
shiftKey (obsolete)

Superseded by parameters to

**AuthenticView.OnKeyboardEvent** (On_AuthenticView_KeyPressed)

**AuthenticView.OnMouseEvent** (On_AuthenticView_MouseEvent)

**AuthenticView.OnDragOver** (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

```javascript
// ----- XMLSpy scripting environment - javascript sample -----
// instead of:
// function On_DocEditDragOver ()
// {
//     if (Application.ActiveDocument.DocEditView.event.shiftKey ||
//        MsgBox ("shift key is down");
// }
// use now:
function On_AuthenticView_DragOver (long i_nXPos, long i_nYPos,
SPYMouseEvent i_eMouseEvent,
IAuthenticRange *i_ipRange,
IAuthenticDataTransfer *i_ipData)
{
    if (i_eMouseEvent & spyShiftKeyMask)
        MsgBox ("shift key is down");
}
```

See also

*Declaration:* shiftKey as Boolean

*Description*

True if the right SHIFT key is pressed.
shiftLeft (obsolete)

Superseded by parameters to

- `AuthenticView.OnKeyboardEvent` (On_AuthenticView_KeyDown)
- `AuthenticView.OnMouseEvent` (On_AuthenticView_MouseEvent)
- `AuthenticView.OnDragOver` (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

```javascript
// ----- XMLSpy scripting environment - javascript sample ----- 
// instead of:
// function On_DocEditDragOver ()
// {
//     if (Application.ActiveDocument.DocEditView.event.shiftKey ||
//        MsgBox ("shift key is down");
// }
// use now:
function On_AuthenticView_DragOver (long i_nXPos, long i_nYPos,
                                    SPYMouseEvent i_eMouseEvent,
                                    IAuthenticRange *i_ipRange,
                                    IAuthenticDataTransfer *i_ipData)
{
    if (i_eMouseEvent & spyShiftKeyMask)
        MsgBox ("shift key is down");
}
```

See also

*Declaration:* `shiftLeft` as Boolean

*Description*
True if the left SHIFT key is pressed.
srcElement (obsolete)

Superseded by parameters to

- AuthenticView.OnMouseEvent (On_AuthenticView_MouseEvent)
- AuthenticView.OnBeforeDrop (On_AuthenticView_BeforeDrop)
- AuthenticView.OnDragOver (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

With the new event handler function, a range object selecting this element is provided instead of the XMLData element currently below the mouse cursor.

```javascript
// ----- XMLSpy scripting environment - javascript sample ----- 
// instead of:
// function On_DocEditMouseMove ()
// {
//     var objEvent = Application.ActiveDocument.DocEditView.event;
//     if (objEvent.srcElement != null)
//         MsgBox ("moving over " + objEvent.srcElement.Parent.Name);
// }
// use now:
function On_AuthenticView_MouseEvent (long i_nXPos, long i_nYPos, SPYMouseEvent i_eMouseEvent, IAuthenticRange *i_ipRange)
{
    if ((i_eMouseEvent & spyMouseMoveMask) && (i_ipRange != null))
        MsgBox ("moving over " + i_ipRange.FirstXMLData.Parent.Name);
}
```

See also

Declaration: **srcElement** as Variant

Description

Element which fires the current event. This is usually an **XMLData** object.
type (obsolete)

Not supported

See also

Declaration: type as String (not supported)

Description
Currently no event sets this property.
### 3.3.2 AuthenticSelection (obsolete)

**Superseded by AuthenticRange**

The DocEditView object is renamed to OldAuthenticView. DocEditSelection is renamed to AuthenticSelection. DocEditEvent is renamed to AuthenticEvent. DocEditDataTransfer is renamed to AuthenticDataTransfer.

Their usage - except for AuthenticDataTransfer - is no longer recommended. We will continue to support existing functionality for a yet undefined period of time but no new features will be added to these interface. All functionality available up to now in DocEditView, DocEditSelection, DocEditEvent and DocEditDataTransfer is now available via AuthenticView, AuthenticRange and AuthenticDataTransfer. Many new features have been added.

For examples on migrating from DocEdit to Authentic see the description of the different methods and properties of the different DocEdit objects.

**See also**

**Properties**  
*Start*  
*StartTextPosition*  
*End*  
*EndTextPosition*
End (obsolete)

Superseded by AuthenticRange.LastXMLData

// ----- javascript sample -----  
// instead of:  
// var objXMLData =  
// use now:  
var objXMLData =  

See also

Declaration: End as XMLData

Description
XML element where the current selection ends.

EndTextPosition (obsolete)

Superseded by AuthenticRange.LastXMLDataOffset

// ----- javascript sample -----  
// instead of:  
// var nOffset =  
// use now:  
var nOffset =  
Application.ActiveDocument.AuthenticView.Selection.LastXMLDataOffset;

See also

Declaration: EndTextPosition as long

Description
Start (obsolete)

Superseded by **AuthenticRange.FirstXMLData**

```javascript
// ----- javascript sample -----  
// instead of:
// var objXMLData =  
// use now:
var objXMLData =  
```

See also

**Declaration:** `Start as XMLData`

**Description**
XML element where the current selection starts.

StartTextPosition (obsolete)

Superseded by **AuthenticRange.FirstXMLDataOffset**

```javascript
// ----- javascript sample -----  
// instead of:
// var nOffset =  
// use now:
var nOffset =  
```

See also

**Declaration:** `StartTextPosition as long`

**Description**
3.3.3 OldAuthenticView (obsolete)

<table>
<thead>
<tr>
<th>Superseded by</th>
<th>AuthenticView and AuthenticRange</th>
</tr>
</thead>
<tbody>
<tr>
<td>The DocEditView object is renamed to OldAuthenticView.</td>
<td></td>
</tr>
<tr>
<td>DocEditSelection is renamed to AuthenticSelection.</td>
<td></td>
</tr>
<tr>
<td>DocEditEvent is renamed to AuthenticEvent.</td>
<td></td>
</tr>
<tr>
<td>DocEditDataTransfer is renamed to AuthenticDataTransfer.</td>
<td></td>
</tr>
</tbody>
</table>

Their usage - except for AuthenticDataTransfer - is no longer recommended. We will continue to support existing functionality for a yet undefined period of time but no new features will be added to these interfaces. All functionality available up to now in DocEditView, DocEditSelection, DocEditEvent and DocEditDataTransfer is now available via AuthenticView, AuthenticRange and AuthenticDataTransfer. Many new features have been added.

For examples on migrating from DocEdit to Authentic see the description of the different methods and properties of the different DocEdit objects.

See also

Methods

LoadXML
SaveXML

EditClear
EditCopy
EditCut
EditPaste
EditRedo
EditSelectAll
EditUndo

RowAppend
RowDelete
RowDuplicate
RowInsert
RowMoveDown
RowMoveUp

ApplyTextState
IsTextStateApplied
IsTextStateEnabled

MarkUpView

SelectionSet
SelectionMoveTabOrder
Interfaces (obsolete)

GetNextVisible
GetPreviousVisible
GetAllowedElements

Properties
CurrentSelection

event

XMLRoot

IsEditClearEnabled
IsEditCopyEnabled
IsEditCutEnabled
IsEditPasteEnabled
IsEditRedoEnabled
IsEditUndoEnabled

IsRowAppendEnabled
IsRowDeleteEnabled
IsRowDuplicateEnabled
IsRowInsertEnabled
IsRowMoveDownEnabled
IsRowMoveUpEnabled

Description
Interface for Authentic View.
ApplyTextState (obsolete)

Superseded by AuthenticRange.PerformAction

Use spyAuthenticApply for the eAction parameter. The PerformAction method allows to apply text state attributes to any range of the document, not only the current UI selection.

// ----- javascript sample -----  
// instead of:  
// use now:  
    MsgBox ("Error: can't set current selection to bold");

See also

Declaration: ApplyTextState (elementName as String)

Description
Applies or removes the text state defined by the parameter elementName. Common examples for the parameter elementName would be strong and italic.

In an XML document there are segments of data, which may contain sub-elements. For example consider the following HTML:

<bf>fragment</bf>

The HTML tag <b> will cause the word fragment to be bold. However, this only happens because the HTML parser knows that the tag <b> is bold. With XML there is much more flexibility. It is possible to define any XML tag to do anything you desire. The point is that it is possible to apply a Text state using XML. But the Text state that is applied must be part of the schema. For example in the OrgChart.xml, OrgChart.sps, OrgChart.xsd example the tag <strong> is the same as bold. And to apply bold the method ApplyTextState() is called. But like the row and edit operations it is necessary to test if it is possible to apply the text state.

See also IsTextStateEnabled and IsTextStateApplied.
CurrentSelection (obsolete)

Superseded by AuthenticView.Selection

The returned AuthenticRange object supports navigation via XMLData elements as well as navigation by document elements (e.g. characters, words, tags) or text cursor positions.

// ----- javascript sample -----  
// instead of:  
// use now:  
var objRange = Application.ActiveDocument.AuthenticView.Selection;

See also

Declaration: CurrentSelection as DocEditSelection

Description
The property provides access to the current selection in the Authentic View.

EditClear (obsolete)

Superseded by AuthenticRange.Delete

The Delete method of AuthenticRange allows to delete any range of the document, not only the current UI selection.

// ----- javascript sample -----  
// instead of:  
// use now:  
    MsgBox ("Error: can't delete current selection");

See also

Declaration: EditClear

Description
Deletes the current selection.
EditCopy (obsolete)

**Superseded by AuthenticRange.Copy**

The Copy method of AuthenticRange allows to delete any range of the document, not only the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// use now:  
    MsgBox ("Error: can't copy current selection");
```

See also

*Declaration: EditCopy*

**Description**
Copies the current selection to the clipboard.

EditCut (obsolete)

**Superseded by AuthenticRange.Cut**

The Cut method of AuthenticRange allows to delete any range of the document, not only the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// use now:  
    MsgBox ("Error: can't cut out current selection");
```

See also

*Declaration: EditCut*

**Description**
Cuts the current selection from the document and copies it to the clipboard.
EditPaste (obsolete)

**Superseded by AuthenticRange.Paste**

The Paste method of AuthenticRange allows to delete any range of the document, not only the current UI selection.

```javascript
// ----- javascript sample ----- 
// instead of: 
// use now: 
    MsgBox ("Error: can't paste to current selection");
```

See also

*Declaration:* EditPaste

*Description*

Pastes the content from the clipboard into the document.

EditRedo (obsolete)

**Superseded by AuthenticView.Redo**

```javascript
// ----- javascript sample ----- 
// instead of: 
// use now: 
if (! Application.ActiveDocument.AuthenticView.Redo()) 
    MsgBox ("Error: no redo step available");
```

See also

*Declaration:* EditRedo

*Description*

Redo the last undo step.
EditSelectAll (obsolete)

Superseded by AuthenticView.WholeDocument and AuthenticRange.Select

// ----- javascript sample ----- 
// instead of: 
// Application.ActiveDocument.DocEditView.EditSelectAll(); 
// use now: 

See also

Declaration: EditSelectAll

Description
The method selects the complete document.

EditUndo (obsolete)

Superseded by AuthenticView.Undo

// ----- javascript sample ----- 
// instead of: 
// use now: 
if (! Application.ActiveDocument.AuthenticView.Undo())
    MsgBox ("Error: no undo step available");

See also

Declaration: EditUndo

Description
Undo the last action.
event (obsolete)

Superseded by parameters to **AuthenticView events**.

See also

*Declaration:* event as **DocEditEvent**

*Description*
The event property holds a **DocEditEvent** object which contains information about the current event.

GetAllowedElements (obsolete)

Superseded by **AuthenticRange.CanPerformActionWith**

AuthenticRange now supports all functionality of the 'elements' entry helper. Besides querying the elements that can be inserted, appended, etc., you can invoke the action as well. See **AuthenticRange.PerformAction** for more information.

```javascript
// ----- javascript sample ----- 
// instead of: 
// var arrElements = New Array(); 
// var objEndElement = objDocEditView.CurrentSelection.End; 
// objDocEditView.GetAllowedElements(k_ActionInsertBefore, objStartElement, objEndElement, arrElements); 
// use now: 
var arrElements = New Array(); 
Application.ActiveDocument.AuthenticView.Selection.CanPerformActionWith (spyAuthenticInsertBefore, arrElements);
```

See also

*Declaration:* GetAllowedElements (**nAction** as **SpyAuthenticElementActions**, **pStartElement** as **XMLData**, **pEndElement** as **XMLData**, **pElements** as **Variant**)

*Description*
GetAllowedElements() returns the allowed elements for the various actions specified by **nAction**.

*JavaScript example:*

```javascript
Function GetAllowed()
```
```javascript
{
    var objView = Application.ActiveDocument.DocEditView;

    var arrElements = New Array(1);

    var objStart = objView.CurrentSelection.Start;
    var objEnd = objView.CurrentSelection.End;

    var strText;
    strText = "valid elements at current selection:\n\n";

    For(var i = 1;i <= 4;i++) {
        objPlugIn.GetAllowedElements(i, objStart, objEnd, arrElements);
        strText = strText + ListArray(arrElements) + "------------------";
    }

    Return strText;
}

Function ListArray(arrIn)
{
    var strText = "";

    If(TypeOf(arrIn) == "object"){
        For(var i = 0;i <= (arrIn.length - 1);i++)
            strText = strText + arrIn[i] + "\n";
    }

    Return strText;
}

VBScript example:

Sub DisplayAllowed
    Dim objView
    Set objView = Application.ActiveDocument.DocEditView

    Dim arrElements()

    Dim objStart
    Dim objEnd
    Set objStart = objView.CurrentSelection.Start
    Set objEnd = objView.CurrentSelection.End

    Dim strText
    strText = "valid elements at current selection:" & chr(13) & chr(13)

    Dim i

    For i = 1 To 4
        objView.GetAllowedElements i, objStart, objEnd, arrElements
        strText = strText & ListArray(arrElements) & "------------------" & chr(13)
    Next
```
```vbnet
msgbox strText
End Sub

Function ListArray(arrIn)
    Dim strText

    If IsArray(arrIn) Then
        Dim i

        For i = 0 To UBound(arrIn)
            strText = strText & arrIn(i) & chr(13)
        Next
    End If

    ListArray = strText
End Function
```
GetNextVisible (obsolete)

Superseded by AuthenticRange.SelectNext

AuthenticRange now supports a wide range of element navigation methods based on document elements like characters, words, tags and many more. Selecting the text passage that represents the content of the next XML element is just one of them.

```javascript
// ----- javascript sample ----- 
// instead of:
// var objCurrXMLData = ... 
// var objXMLData = Application.ActiveDocument.DocEditView.GetNextVisible(objCurrXMLData); 
// Application.ActiveDocument.DocEditView.SelectionSet (objXMLData, 0, objXMLData, -1); 
// use now:
var objRange = ... 
try 
  { objRange.SelectNext (spyAuthenticTag).Select(); } 
catch (err) 
  { 
    if ((err.number & 0xffff) == 2003) 
      MsgBox ("end of document reached"); 
    else 
      throw (err); 
  }
```

See also

**Declaration:** `GetNextVisible (pElement as XMLData) as XMLData`

**Description**
The method gets the next visible XML element in the document.
GetPreviousVisible (obsolete)

Superseded by **AuthenticRange.SelectPrevious**

AuthenticRange now supports a wide range of element navigation methods based on document elements like characters, words, tags and many more. Selecting the text passage that represents the content of the previous XML element is just one of them.

```javascript
// ----- javascript sample ----- 
// instead of: 
// var objCurrXMLData = ... 
// var objXMLData = Application.ActiveDocument.DocEditView.GetPreviousVisible(objCurrXMLData);
// Application.ActiveDocument.DocEditView.SelectionSet (objXMLData, 0, objXMLData, -1);
// use now: 
var objRange = ...
try {
    objRange.SelectPrevious (spyAuthenticTag).Select();
} catch (err) {
    if ((err.number & 0xffff) == 2004) 
        MsgBox ("begin of document reached");
    else 
        throw (err);
}
```

See also

**Declaration:** GetPreviousVisible (pElement as XMLData) as XMLData

**Description**
The method gets the previous visible XML element in the document.
IsEditClearEnabled (obsolete)

**Superseded by** **AuthenticRange.IsDeleteEnabled**

The IsDeleteEnabled property is now supported for any range of the document, not only the current UI selection.

```javascript
// ----- javascript sample ----- 
// instead of: 
// use now:
if (objCurrSelection.IsDeleteEnabled) 
    objCurrSelection.Delete();
```

See also

**Declaration:** IsEditClearEnabled as Boolean

**Description**
True if EditClear is possible. See also Editing operations.

IsEditCopyEnabled (obsolete)

**Superseded by** **AuthenticRange.IsCopyEnabled**

The IsCopyEnabled property is now supported for any range of the document, not only the current UI selection.

```javascript
// ----- javascript sample ----- 
// instead of: 
// use now:
if (objCurrSelection.IsCopyEnabled) 
    objCurrSelection.Copy();
```

See also

**Declaration:** IsEditCopyEnabled as Boolean

**Description**
True if copy to clipboard is possible. See also EditCopy and Editing operations.
IsEditCutEnabled (obsolete)

**Superseded by AuthenticRange.IsCutEnabled**

The IsCutEnabled property is now supported for any range of the document, not only the current UI selection.

```javascript
// ----- javascript sample -----  
// instead of:  
// use now:  
if (objCurrSelection.IsCutEnabled)  
   objCurrSelection.Cut();
```

See also

**Declaration:** IsEditCutEnabled as Boolean

**Description**

True if EditCut is currently possible. See also Editing operations.

IsEditPasteEnabled (obsolete)

**Superseded by AuthenticRange.IsPasteEnabled**

The IsPasteEnabled property is now supported for any range of the document, not only the current UI selection.

```javascript
// ----- javascript sample -----  
// instead of:  
// use now:  
if (objCurrSelection.IsPasteEnabled)  
   objCurrSelection.Paste();
```

See also

**Declaration:** IsEditPasteEnabled as Boolean

**Description**

True if EditPaste is possible. See also Editing operations.
**IsEditRedoEnabled (obsolete)**

*Superseded by [AuthenticView.IsRedoEnabled](https://www.altova.com/)

```javascript
// ----- javascript sample -----  
// instead of:  
// use now:  
```

See also

*Declaration: [IsEditRedoEnabled](https://www.altova.com/) as Boolean*

**Description**

True if [EditRedo](https://www.altova.com/) is currently possible. See also [Editing operations](https://www.altova.com/).

**IsEditUndoEnabled (obsolete)**

*Superseded by [AuthenticView.IsUndoEnabled](https://www.altova.com/)

```javascript
// ----- javascript sample -----  
// instead of:  
// use now:  
if (Application.ActiveDocument.AuthenticView.IsUndoEnabled)  
    Application.ActiveDocument.AuthenticView.Undo();
```

See also

*Declaration: [IsEditUndoEnabled](https://www.altova.com/) as Boolean*

**Description**

True if [EditUndo](https://www.altova.com/) is possible. See also [Editing operations](https://www.altova.com/).
IsRowAppendEnabled (obsolete)

Superseded by AuthenticRange.IsInDynamicTable

The operations 'insert', 'append', 'delete' and 'duplicate' row are available whenever the selection is inside a dynamic table.

```javascript
// ----- javascript sample ----- 
// instead of:
// use now:
if (Application.ActiveDocument.AuthenticView.Selection.IsInDynamicTable())
```

See also

Declaration: IsRowAppendEnabled as Boolean

Description
True if RowAppend is possible. See also Row operations.

IsRowDeleteEnabled (obsolete)

Superseded by AuthenticRange.IsInDynamicTable

The operations 'insert', 'append', 'delete' and 'duplicate' row are available whenever the selection is inside a dynamic table.

```javascript
// ----- javascript sample ----- 
// instead of:
// use now:
if (Application.ActiveDocument.AuthenticView.Selection.IsInDynamicTable())
```

See also

Declaration: IsRowDeleteEnabled as Boolean

Description
True if RowDelete is possible. See also Row operations.
IsRowDuplicateEnabled (obsolete)

**Superseded by** [AuthenticRange.IsInDynamicTable](#)

The operations 'insert', 'append', 'delete' and 'duplicate' row are available whenever the selection is inside a dynamic table.

```javascript
// ----- javascript sample ----- 
// instead of: 
// use now: 
if (Application.ActiveDocument.AuthenticView.Selection.IsInDynamicTable())
```

**See also**

**Declaration:** IsRowDuplicateEnabled as Boolean

**Description**

True if RowDuplicate is currently possible. See also Row operations.

IsRowInsertEnabled (obsolete)

**Superseded by** [AuthenticRange.IsInDynamicTable](#)

The operations 'insert', 'append', 'delete' and 'duplicate' row are available whenever the selection is inside a dynamic table.

```javascript
// ----- javascript sample ----- 
// instead of: 
// use now: 
if (Application.ActiveDocument.AuthenticView.Selection.IsInDynamicTable())
```

**See also**

**Declaration:** IsRowInsertEnabled as Boolean

**Description**

True if RowInsert is possible. See also Row operations.
IsRowMoveDownEnabled (obsolete)

Superseded by **AuthenticRange.IsLastRow**

```javascript
// ----- javascript sample ----- 
// instead of: 
// use now: 
```

See also

**Declaration:** IsRowMoveDownEnabled as Boolean

**Description**

True if **RowMoveDown** is currently possible. See also **Row operations.**

IsRowMoveUpEnabled (obsolete)

Superseded by **AuthenticRange.IsFirstRow**

```javascript
// ----- javascript sample ----- 
// instead of: 
// use now: 
```

See also

**Declaration:** IsRowMoveUpEnabled as Boolean

**Description**

True if **RowMoveUp** is possible. See also **Row operations.**
IsTextStateApplied (obsolete)

**Superseded by** [AuthenticRange.IsTextStateApplied](#)

```javascript
// ----- javascript sample -----  
// instead of:  
// use now:  
  MsgBox("bold on");  
else  
  MsgBox("bold off");
```

**See also**

**Declaration:** `IsTextStateApplied(elementName as String) as Boolean`

**Description**
Checks to see if the text state has already been applied. Common examples for the parameter `elementName` would be strong and italic.

IsTextStateEnabled (obsolete)

**Superseded by** [AuthenticRange.CanPerformAction](#)

Use `spyAuthenticApply` for the `eAction` parameter. The `CanPerformAction` method allows to operate on any range of the document, not only the current UI selection.

```javascript
// ----- javascript sample -----  
// instead of:  
// use now:  
  ... // e.g. enable 'bold' button
```

**See also**

**Declaration:** `IsTextStateEnabled(i_strElementName as String) as Boolean`

**Description**
Checks to see if it is possible to apply a text state. Common examples for the parameter `elementName` would be strong and italic.
LoadXML (obsolete)

**Superseded by AuthenticView.AsXMLString**

AuthenticView now supports the property AsXMLString that can be used to directly access and replace the document content as an XMLString.

```javascript
// ----- javascript sample -----  
// instead of: 
// Application.ActiveDocument.DocEditView.LoadXML (strDocAsXMLString);  
// use now: 
try 
  { Application.ActiveDocument.AuthenticView.AsXMLString = strDocAsXMLString; 
  } 
catch (err) 
  { MsgBox ("Error: invalid XML string"); } 
```

See also

**Declaration:** `LoadXML(xmlString as String)`

**Description**

Loads the current XML document with the XML string applied. The new content is displayed immediately.

The `xmlString` parameter must begin with the XML declaration, e.g.,

```javascript
objPlugIn.LoadXML("<?xml version='1.0' encoding='UTF-8'?><root></root>";
```

MarkUpView (obsolete)

**Superseded by AuthenticView.MarkupVisibility**

```javascript
// ----- javascript sample -----  
// instead of: 
// use now: 
spyAuthenticMarkupLarge;
```

See also

**Declaration:** `MarkUpView(kind as long)`

**Description**

By default the document displayed is using HTML techniques. But sometimes it is desirable to show the editing tags. Using this method it is possible to display three different types of markup tags:
<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>hide the markup tags</td>
</tr>
<tr>
<td>2</td>
<td>show the large markup tags</td>
</tr>
<tr>
<td>3</td>
<td>show the mixed markup tags</td>
</tr>
</tbody>
</table>
RowAppend (obsolete)

**Superseded by AuthenticRange.AppendRow**

The table operations of AuthenticRange now allow to manipulate any table in the current document independent of the current UI selection.

```javascript
// ----- javascript sample ----- 
// instead of: 
// use now: 
  MsgBox ("Error: can't append row");
```

See also

*Declaration:* RowAppend

**Description**

Appends a row at the current position.

See also [Row operations](#).

RowDelete (obsolete)

**Superseded by AuthenticRange.DeleteRow**

The table operations of AuthenticRange now allow to manipulate any table in the current document independent of the current UI selection.

```javascript
// ----- javascript sample ----- 
// instead of: 
// use now: 
  MsgBox ("Error: can't delete row");
```

See also

*Declaration:* RowDelete

**Description**

Deletes the currently selected row(s).

See also [Row operations](#).
RowDuplicate (obsolete)

**Superseded by AuthenticRange.DuplicateRow**

The table operations of AuthenticRange now allow to manipulate any table in the current document independent of the current UI selection.

// ----- javascript sample -----  
// instead of:  
// use now:  
    MsgBox ("Error: can't duplicate row");

See also

*Declaration:* RowDuplicate

*Description*

The method duplicates the currently selected rows.

See also Row operations.

RowInsert (obsolete)

**Superseded by AuthenticRange.InsertRow**

The table operations of AuthenticRange now allow to manipulate any table in the current document independent of the current UI selection.

// ----- javascript sample -----  
// instead of:  
// use now:  
    MsgBox ("Error: can't insert row");

See also

*Declaration:* RowInsert

*Description*

Inserts a new row immediately above the current selection.

See also Row operations.
RowMoveDown (obsolete)

Superseded by **AuthenticRange.MoveRowDown**

The table operations of AuthenticRange now allow to manipulate any table in the current document independent of the current UI selection.

```javascript
// ----- javascript sample ----- 
// instead of: 
// use now: 
  MsgBox ("Error: can't move row down");
```

See also

**Declaration:** RowMoveDown

**Description**
Moves the current row one position down.

See also **Row operations.**

RowMoveUp (obsolete)

Superseded by **AuthenticRange.MoveRowUp**

The table operations of AuthenticRange now allow to manipulate any table in the current document independent of the current UI selection.

```javascript
// ----- javascript sample ----- 
// instead of: 
// use now: 
  MsgBox ("Error: can't move row up");
```

See also

**Declaration:** RowMoveUp

**Description**
Moves the current row one position up.

See also **Row operations.**
SaveXML (obsolete)

**Superseded by AuthenticView.AsXMLString**

AuthenticView now supports the property XMLString that can be used to directly access and replace the document content as an XMLString.

```javascript
// ----- javascript sample -----  
// instead of: 
// use now:  
try 
    { 
        var strDocAsXMLString = Application.ActiveDocument.AuthenticView.AsXMLString;  
        ... // do something here  
    } 
catch (err) 
    { MsgBox ("Error: invalid XML string"); } 
```

**See also**

*Declaration:* SaveXML as String

**Return Value**

XML structure as string

**Description**

Saves the current XML data to a string that is returned to the caller.
SelectionMoveTabOrder (obsolete)

Superseded by **AuthenticRange.SelectNext**

AuthenticRange now supports a wide range of element navigation methods based on document elements like characters, words, tags and many more. Selecting the next paragraph is just one of them, and navigation is not necessarily bound to the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.SelectionMoveTabOrder(true, true);  
// use now:  
(spyAuthenticParagraph).Select();  
// to append a row to a table use AuthenticRange.AppendRow
```

See also

**Declaration:** SelectionMoveTabOrder (*bForward* as Boolean, *bTag* as Boolean)

**Description**

SelectionMoveTabOrder() moves the current selection forwards or backwards.

If *bTag* is false and the current selection is at the last cell of a table a new line will be added.
SelectionSet (obsolete)

Superseded by AuthenticRange.FirstXMLData and related properties

AuthenticRange supports navigation via XMLData elements as well as navigation by document elements (e.g. characters, words, tags) or text cursor positions.

// ----- javascript sample -----  
// instead of:  
// if (! Application.ActiveDocument.DocEditView.SelectionSet(varXMLData1, 0,  
// varXMLData2, -1))  
//   MsgBox ("Error: invalid data position");  
// use now:  
try  
{  
  var objSelection = Application.ActiveDocument.AuthenticView.Selection;  
  objSelection.FirstXMLData = varXMLData1;  
  objSelection.FirstXMLdataOffset = 0;  
  objSelection.LastXMLData = varXMLData2;  
  objSelection.LastXMLDataOffset = -1;  
  objSelection.Select();  
}  
catch (err)  
{  
  MsgBox ("Error: invalid data position");  
}  
// to select all text between varXMLData1 and varXMLData2, inclusive

See also

Declaration: SelectionSet (pStartElement as XMLData, nStartPos as long,  
pEndElement as XMLData, nEndPos as long) as Boolean

Description
Use SelectionSet() to set a new selection in the Authentic View. Its possible to set  
pEndElement to null (nothing) if the selection should be just over one (pStartElement) XML  
element.
XMLRoot (obsolete)

**Superseded by** [AuthenticView/XMLDataRoot](#)

```javascript
// ----- javascript sample -----  
// instead of:
// var objXMLData = Application.ActiveDocument.DocEditView.XMLRoot;
// use now:
var objXMLData = Application.ActiveDocument.AuthenticView.XMLDataRoot;
```

**See also**

*Declaration*: `XMLRoot` as `XMLData`

**Description**

`XMLRoot` is the parent element of the currently displayed XML structure. Using the `XMLData` interface you have full access to the complete content of the file.
3.4 Enumerations

This is a list of all enumerations used by the XMLSpy API. If your scripting environment does not support enumerations use the number-values instead.

3.4.1 ENUMApplicationStatus

Description
Enumeration to specify the current Application status.

Possible values:

- eApplicationRunning = 0
- eApplicationAfterLicenseCheck = 1
- eApplicationBeforeLicenseCheck = 2
- eApplicationConcurrentLicenseCheckFailed = 3
- eApplicationProcessingCommandLine = 4

3.4.2 SPYAttributeTypeDefinition

Description
Attribute type definition that can be selected for generation of Sample XML. This type is used with the method `GenerateDTDOrSchema` and `GenerateDTDOrSchemaEx`.

Possible values:

- spyMergedGlobal = 0
- spyDistinctGlobal = 1
- spyLocal = 2

3.4.3 SPYAuthenticActions

Description
Actions that can be performed on `AuthenticRange` objects.

Possible values:

- spyAuthenticInsertAt = 0
- spyAuthenticApply = 1
- spyAuthenticClearSurr = 2
- spyAuthenticAppend = 3
- spyAuthenticInsertBefore = 4
- spyAuthenticRemove = 5

3.4.4 SPYAuthenticDocumentPosition

Description
Relative and absolute positions used for navigating with `AuthenticRange` objects.

Possible values:

- spyAuthenticDocumentBegin = 0
- spyAuthenticDocumentEnd = 1
3.4.5 **SPYAuthenticElementActions**

**Description**
Actions that can be used with the obsolete object GetAllowedElements (superseded by AuthenticRange.CanPerformActionWith).

**Possible values:**
- k_ActionInsertAt = 0
- k_ActionApply    = 1
- k_ActionClearSurr = 2
- k_ActionAppend   = 3
- k_ActionInsertBefore = 4
- k_ActionRemove  = 5

3.4.6 **SPYAuthenticElementKind**

**Description**
Enumeration of the different kinds of elements used for navigation and selection within the AuthenticRange and AuthenticView objects.

**Possible values:**
- spyAuthenticChar   = 0
- spyAuthenticWord   = 1
- spyAuthenticLine   = 3
- spyAuthenticParagraph   = 4
- spyAuthenticTag    = 6
- spyAuthenticDocument = 8
- spyAuthenticTable  = 9
- spyAuthenticTableRow = 10
- spyAuthenticTableColumn = 11

3.4.7 **SPYAuthenticMarkupVisibility**

**Description**
Enumeration values to customize the visibility of markup with MarkupVisibility.

**Possible values:**
- spyAuthenticMarkupHidden    = 0
- spyAuthenticMarkupSmall     = 1
- spyAuthenticMarkupLarge     = 2
- spyAuthenticMarkupMixed     = 3

3.4.8 **SPYAuthenticToolbarButtonState**

**Description**
Authentic toolbar button states are given by the following enumeration:

**Possible values:**
authenticToolbarButtonDefault = 0
authenticToolbarButtonEnabled = 1
authenticToolbarButtonDisabled = 2

3.4.9 SPYDatabaseKind

Description
Values to select different kinds of databases for import. See DatabaseConnection.DatabaseKind for its use.

Possible values:
spyDB_Access = 0
spyDB_SQLServer = 1
spyDB_Oracle = 2
spyDB_Sybase = 3
spyDB_MySQL = 4
spyDB_DB2 = 5
spyDB_Other = 6
spyDB_Unspecified = 7
spyDB_PostgreSQL = 8
spyDB_iSeries = 9

3.4.10 SPYDialogAction

Description
Values to simulate different interactions on dialogs. See Dialogs for all dialogs available.

Possible values:
spyDialogOK = 0 // simulate click on OK button
spyDialogCancel = 1 // simulate click on Cancel button
spyDialogUserInput = 2 // show dialog and allow user interaction

3.4.11 SPYDOMType

Description
Enumeration values to parameterize generation of C++ code from schema definitions.

Possible values:
spyDOMType_msxml4 = 0 // Obsolete
spyDOMType_xerces = 1
spyDOMType_xerces3 = 2
spyDOMType_msxml6 = 3

spyDOMType_xerces indicates Xerces 2.x usage; spyDOMType_xerces3 indicates Xerces 3.x usage.
3.4.12  **SPYDTDSchemaFormat**

Description
Enumeration to identify the different schema formats.

Possible values:

spyDTD = 0
spyW3C = 1

3.4.13  **SPYEncodingByteOrder**

Description
Enumeration values to specify encoding byte ordering for text import and export.

Possible values:

spyNONE = 0
spyLITTLE_ENDIAN = 1
spyBIG_ENDIAN = 2

3.4.14  **SPYExportNamespace**

Description
Enumeration type to configure handling of namespace identifiers during export.

Possible values:

spyNoNamespace = 0
spyReplaceColonWithUnderscore = 1

3.4.15  **SPYFindInFilesSearchLocation**

Description
The different locations where a search can be performed. This type is used with the `FindInFilesDlg` dialog.

Possible values:

spyFindInFiles_Documents = 0
spyFindInFiles_Project = 1
spyFindInFiles_Folder = 2

3.4.16  **SPYFrequentElements**

Description
Enumeration value to parameterize schema generation.

Possible values:

spyGlobalElements = 0
spyGlobalComplexType = 1

3.4.17  **SPYImageKind**

Description
Enumeration values to parameterize image type of the generated documentation. These values are used in `SchemaDocumentationDialog.DiagramFormat`. 
Possible values:
spyImageType_PNG = 0
spyImageType_EMF = 1

3.4.18 SPYImportColumnsType
Description
Enumeration to specify different Import columns types.

Possible values:
spyImportColumns_Element = 0
spyImportColumns_Attribute = 1

3.4.19 SPYKeyEvent
Description
Enumeration type to identify the different key events. These events correspond with the equally named windows messages.

Possible values:
spyKeyDown = 0
spyKeyUp = 1
spyKeyPressed = 2

3.4.20 SPYKeyStatus
Description
Enumeration type to identify the key status.

Possible values:
spyLeftShiftKeyMask = 1
spyRightShiftKeyMask = 2
spyLeftCtrlKeyMask = 4
spyRightCtrlKeyMask = 8
spyLeftAltKeyMask = 16
spyRightAltKeyMask = 32

3.4.21 SPYLibType
Description
Enumeration values to parameterize generation of C++ code from schema definitions.

Possible values:
spyLibType_static = 0
spyLibType_dll = 1

3.4.22 SPYLoading
Description
Enumeration values to define loading behaviour of URL files.
Possible values:
spyUseCacheProxy = 0
spyReload = 1

3.4.23  SPYMouseEvent

Description
Enumeration type that defines the mouse status during a mouse event. Use the enumeration values as bitmasks rather than directly comparing with them.

Examples
'
 to check for ctrl-leftbutton-down in VB
If (i_eMouseEvent = (XMLSpyLib.spyLeftButtonDownMask Or XMLSpyLib.spyCtrlKeyDownMask)) Then
   ' react on ctrl-leftbutton-down
End If
'
 to check for double-click with any button in VBScript
If (((i_eMouseEvent And spyDoubleClickMask) <> 0) Then
   ' react on double-click
End If

Possible values:
spyNoButtonMask = 0
spyMouseMoveMask = 1
spyLeftButtonMask = 2
spyMiddleButtonMask = 4
spyRightButtonMask = 8
spyButtonDownMask = 16
spyButtonUpMask = 32
spyDoubleClickMask = 64
spyShiftKeyDownMask = 128
spyCtrlKeyDownMask = 256
spyLeftButtonDownMask = 34 // spyLeftButtonMask | spyButtonDownMask
spyMiddleButtonDownMask = 36 // spyMiddleButtonMask | spyButtonDownMask
spyRightButtonDownMask = 40 // spyRightButtonMask | spyButtonDownMask
spyMiddleButtonUpMask = 20 // spyMiddleButtonMask | spyButtonUpMask
spyRightButtonUpMask = 24 // spyRightButtonMask | spyButtonUpMask
spyLeftDoubleClickMask = 66 // spyRightButtonMask | spyDoubleClickMask
spyMiddleDoubleClickMask = 68 // spyMiddleButtonMask | spyDoubleClickMask
spyRightDoubleClickMask = 72 // spyRightButtonMask | spyDoubleClickMask

3.4.24  SPYNumberDateTimeFormat

Description
Enumeration value to configure database connections.

Possible values:
spySystemLocale = 0
spySchemaCompatible = 1
3.4.25  **SPYProgrammingLanguage**

**Description**
Enumeration values to select the programming language for code generation from schema definitions.

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

**Possible values:**
- `spyUndefinedLanguage` = -1
- `spyJava` = 0
- `spyCpp` = 1
- `spyCSharp` = 2

3.4.26  **SPYProjectItemTypes**

**Description**
Enumeration values to identify the different elements in project item lists. See `SpyProjectItem.ItemType`.

**Possible values:**
- `spyUnknownItem` = 0
- `spyFileItem` = 1
- `spyFolderItem` = 2
- `spyURLItem` = 3

3.4.27  **SPYProjectType**

**Description**
Enumeration values to parameterize generation of C# from schema definitions.

**Possible values:**
- `spyVisualStudioProject` = 0  Obsolete
- `spyVisualStudio2003Project` = 1  Obsolete
- `spyBorlandProject` = 2  Obsolete
- `spyMonoMakefile` = 3  Obsolete
- `spyVisualStudio2005Project` = 4  For C++ code also
- `spyVisualStudio2008Project` = 5  For C++ code also
- `spyVisualStudio2010Project` = 6  For C++ code also

3.4.28  **SpySampleXMLGenerationChoiceMode**

**Description**
This enumeration is used in `GenerateSampleXMLDlg.ChoiceMode`.

**Possible values:**
- `spySampleXMLGen_FirstBranch` = 0
- `spySampleXMLGen_AllBranches` = 1
- `spySampleXMLGen_ShortestBranch` = 2
3.4.29  SPYSampleXMLGenerationOptimization (Obsolete)

This enumeration is OBSOLETE since v2014.

Description
Specify the elements that will be generated in the Sample XML.
This enumeration is used in `GenerateSampleXMLDlg`.

Possible values:
- `spySampleXMLGen_Optimized` = 0
- `spySampleXMLGen_NonMandatoryElements` = 1
- `spySampleXMLGen_Everything` = 2

3.4.30  SpySampleXMLGenerationSampleValueHints

Description
This enumeration is used in `GenerateSampleXMLDlg`, `SampleValueHints`

```
spySampleXMLGen_FirstFit       = 0
spySampleXMLGen_RandomFit      = 1
spySampleXMLGen_CycleThrough   = 2
```

3.4.31  SPYSampleXMLGenerationSchemaOrDTDAssignment

Description
Specifies what kind of reference to the schema/DTD should be added to the generated Sample XML.
This enumeration is used in `GenerateSampleXMLDlg`.

Possible values:
- `spySampleXMLGen_AssignRelatively` = 0
- `spySampleXMLGen_AssignAbsolutely` = 1
- `spySampleXMLGen_DoNotAssign` = 2

3.4.32  SPYSchemaDefKind

Description
Enumeration type to select schema diagram types.

Possible values:
3.4.33 **SPYSchemaDocumentationFormat**

**Description**
Enumeration values to parameterize generation of schema documentation. These values are used in `SchemaDocumentationDialog.OutputFormat`.

**Possible values:**
- `spySchemaDoc_HTML` = 0
- `spySchemaDoc_MSWord` = 1
- `spySchemaDoc_RTF` = 2
- `spySchemaDoc_PDF` = 3

3.4.34 **SPYSchemaExtensionType**

**Description**
Enumeration to specify different Schema Extension types.

**Possible values:**
- `spySchemaExtension_None` = 0
- `spySchemaExtension_SQL_XML` = 1
- `spySchemaExtension_MS_SQL_Server` = 2
- `spySchemaExtension_Oracle` = 3

3.4.35 **SPYSchemaFormat**

**Description**
Enumeration to specify different Schema Format types.

**Possible values:**
- `spySchemaFormat_Hierarchical` = 0
3.4.36 SPYTextDelimiters

**Description**
Enumeration values to specify text delimiters for text export.

**Possible values:**
- spyTabulator = 0
- spySemicolon = 1
- spyComma = 2
- spySpace = 3

3.4.37 SPYTextEnclosing

**Description**
Enumeration value to specify text enclosing characters for text import and export.

**Possible values:**
- spyNoEnclosing = 0
- spySingleQuote = 1
- spyDoubleQuote = 2

3.4.38 SPYTypeDetection

**Description**
Enumeration to select how type detection works during `GenerateDTDOrSchema` and `GenerateDTDOrSchemaEx`.

**Possible values:**
- spyBestPossible = 0
- spyNumbersOnly = 1
- spyNoDetection = 2

3.4.39 SPYURLTypes

**Description**
Enumeration to specify different URL types.

**Possible values:**
-.spyURLTypeAuto = -1
- spyURLTypeXML = 0
- spyURLTypeDTD = 1

3.4.40 SPYValidateXSDVersion

**Description**
Enumeration values that select what XSD version to use. The XSD version that is selected depends on both (i) the presence/absence—and, if present, the value—of the `/xs:schema/@vc:minVersion` attribute of the XSD document, and (ii) the value of this enumeration.

- `spyValidateXSDVersion_1_0` selects XSD 1.0 if `vc:minVersion` is absent, or is present with any value.
- `spyValidateXSDVersion_1_1` selects XSD 1.1 if `vc:minVersion` is absent, or is present with
any value.
spyValidateXSDVersion_AutoDetect selects XSD 1.1 if vc:minVersion=1.1. If the
vc:minVersion attribute is absent, or is present with a value other than 1.1, then XSD 1.0 is
selected.

Possible values
spyValidateXSDVersion_AutoDetect = 0
spyValidateXSDVersion_1_1 = 1
spyValidateXSDVersion_1_0 = 2

3.4.41 SPYValidateErrorFormat

Description
Enumeration values that select the format of the error message.

Possible values
spyValidateErrorFormat_Text = 0
spyValidateErrorFormat_ShortXML = 1
spyValidateErrorFormat_LongXML = 2

3.4.42 SPYViewModes

Description
Enumeration values that define the different view modes for XML documents. The mode
spyViewAuthentic(4) identifies the mode that was intermediately called DocEdit mode and is now
called Authentic mode. The mode spyViewJsonSchema identifies a mode which is mapped to the
Schema Design View on the GUI but is distinguished internally.

Possible values:
spyViewGrid = 0
spyViewText = 1
spyViewBrowser = 2
spyViewSchema = 3
spyViewContent = 4 // obsolete
spyViewAuthentic = 4
spyViewWSDL = 5
spyViewZIP = 6
spyViewEditionInfo = 7
spyViewXBRL = 8
spyViewJsonSchema = 9

3.4.43 SPYVirtualKeyMask

Description
Enumeration type for the most frequently used key masks that identify the status of the virtual
keys. Use these values as bitmasks rather then directly comparing with them. When necessary,
you can create further masks by using the logical or operator.
Examples

' VBScript sample: check if ctrl-key is pressed
If ((i_nVirtualKeyStatus And spyCtrlKeyMask) <> 0)) Then
' ctrl-key is pressed
End If

' VBScript sample: check if ONLY ctrl-key is pressed
If (i_nVirtualKeyStatus == spyCtrlKeyMask) Then
' exactly ctrl-key is pressed
End If

// JScript sample: check if any of the right virtual keys is pressed
if ((i_nVirtualKeyStatus & (spyRightShiftKeyMask | spyRightCtrlKeyMask | spyRightAltKeyMask)) != 0)
{
    ; ' right virtual key is pressed
}

Possible values:
spyNoVirtualKeyMask       = 0
spyLeftShiftKeyMask       = 1
spyRightShiftKeyMask      = 2
spyLeftCtrlKeyMask        = 4
spyRightCtrlKeyMask       = 8
spyLeftAltKeyMask         = 16
spyRightAltKeyMask        = 32
spyShiftKeyMask           = 3    // spyLeftShiftKeyMask | spyRightShiftKeyMask
spyCtrlKeyMask            = 12   // spyLeftCtrlKeyMask | spyRightCtrlKeyMask
spyAltKeyMask             = 48   // spyLeftAltKeyMask | spyRightAltKeyMask

3.4.44 SPYXMLDataKind

Description
The different types of XMLData elements available for XML documents.

Possible values:
spyXMLDataXMLDocStruct    = 0
spyXMLDataXMLEntityDocStruct = 1
spyXMLDataDTDDocStruct    = 2
spyXMLDataXML            = 3
spyXMLDataElement        = 4
spyXMLDataAttr           = 5
spyXMLDataText           = 6
spyXMLDataCData          = 7
spyXMLDataComment        = 8
spyXMLDataPI             = 9
spyXMLDataDefDoctype     = 10
spyXMLDataDefExternalID  = 11
spyXMLDataDefElement     = 12
spyXMLDataDefAttlist     = 13
spyXMLDataDefEntity      = 14
spyXMLDataDefNotation    = 15
spyXMLDataKindsCount     = 16
3.5 Application API for Java (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programmers' Reference | Java.

The Application API in Java has an interface built up of Java classes, each of which corresponds to an object in the Application API. Developers can use these Java classes to interact with the COM API. These classes are listed below and described in subsequent sections. For a description of the Application API objects themselves, see the Application API documentation. Bear in mind that some API features are only available in scripting environments; these have therefore not been ported to Java.

Java classes

SpyApplication
  SpyProject
    SpyProjectItems
      SpyProjectItem
  SpyDocuments
    SpyDoc
      SpyAuthenticView
        SpyAuthenticRange
    SpyDocEditView
      SpyDocEditSelection
    SpyGridView
    SpyTextView
    SpyXMLData
  SpyDialogs
    SpyCodeGeneratorDlg
    SpyDTDSchemaGeneratorDlg
    SpyFileSelectionDlg
    SpyFindInFilesDlg
    SpyGenerateSampleXMLDlg
    SpySchemaDocumentationDlg
  SpyDatabaseConnection
  SpyElementList
  SpyElementListItem
  SpyExportSettings
  SpyFindInFilesResults
    SpyFindInFilesResult
      SpyFindInFilesMatch
  SpyTextImportExportSettings

Implementation of COM properties in Java
Properties in Java have been defined to include both a set and get method (set if it is allowed by the COM implementation). For example, the COM class Document contains the GridView property. In Java the method is called SpyDoc and the property is defined as a GetGridView method.

If you encounter compiling problems, please check the following points:

- The xmlspylib.dll must be available in ..\windows\system32.
- The XMLSpyInterface.jar file must be inserted in the ClassPath environment variable.

### Setting the ClassPath variable in Windows XP

1. Click Start | Settings | Control panel | System | Advanced | Environment Variables. This opens the Environment Variables dialog box.
2. If a ClassPath entry already exists in the System variables group, select the ClassPath entry, and click the Edit button. Edit the path to: "C:\Program Files Altova\xmlspy\XMLSpyInterface.jar".
3. If a ClassPath entry does not exist in the System variables group, click the New button. The New System Variable dialog pops up. Enter CLASSPATH as the variable name, and "C:\Program Files\Altova\xmlspy\XMLSpyInterface.jar" as the ClassPath variable (alter the path to match your installation, if necessary).

### 3.5.1 Sample source code (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

The "SpyDoc doc = app.GetDocuments().OpenFile(...)" command parameter must be altered to suit your environment.

What the sample does:

- Starts a new XMLSpy instance
- Opens the Datasheet.xml file (alter the path here...)
- Switches to the Enhanced Grid view
- Appends a new child element called "NewChild" with the text value "NewValue" element to the root element
- Checks if the document is valid and outputs a message to the Java console
- Quits and releases the XMLSpy application

```java
import XMLSpyInterface.*;

public class TestSpyInterface
{
    public TestSpyInterface() {}

    public static void main(String[] args)
    {
        SpyApplication app = null;
        SpyDoc oDoc = null;
        SpyXMLData oData = null;
        SpyXMLData oNewChild = null;

        try
        {
            app = new SpyApplication();
            app.ShowApplication( true );

            oDoc = app.GetDocuments().OpenFile("C:\FilePath\OrgChart.xml", true );

            // OrgChart.xml is in the folder C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2019. The filepath should be in the form: C:\Documents and Settings\<Username>\Folder\Filename.xml

            if ( oDoc != null )
            {
                oDoc.SwitchViewMode(SPYViewModes.spyViewGrid);
                oData = oDoc.GetRootElement();
                oNewChild = oDoc.CreateChild(SPYXMLDataKind.spyXMLDataElement);

                oNewChild.SetName( "NewChild" );
                oNewChild.SetTextValue("newValue");
                oData.AppendChild(oNewChild);

                if ( oDoc.IsValid() == false )
                {
                    // is to be expected after above insertion
                    System.out.println( "!!!!!!validation error: " + oDoc.GetErrorString() );
                    System.out.println( "!!!!!!validation error: " + oDoc.GetErrorPos() );
                    System.out.println( "!!!!!!validation error: " + oDoc.GetBadData() );
                }
            }
            app.Quit();
        }
    }
    finally
```
If you have difficulties compiling this sample, please try the following commands on the (Start | Run | cmd) command line. Please make sure you are currently in the folder that contains the sample java file.

**compilation**

```
javac -classpath c:\yourpath\there\XMLSpyInterface.jar testspyinterface.java
```

**Execution**

```
java -classpath c:\yourpath\there\XMLSpyInterface.jar testspyinterface
```

### 3.5.2 SpyApplication (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: [Programming Languages | Java](#).

```java
public class SpyApplication {
    public void ReleaseInstance();
    public void ShowApplication( boolean bShow );
    public void Quit();
    public void AddMacroMenuItem( String sMacro, String sDisplayText );
    public void ClearMacroMenu();
    public SpyDoc GetActiveDocument();
    public SpyProject GetCurrentProject();
    public SpyDocuments GetDocuments();
    public SpyElementList GetDatabaseImportElementList( SpyDatabaseConnection oImportSettings );
    public SpyDatabaseConnection GetDatabaseSettings();
    public SpyElementList GetDatabaseTables( SpyDatabaseConnection oImportSettings );
    public SpyExportSettings GetExportSettings();
}
```
public SpyElementList GetTextImportElementList(SpyTextImportExportSettings oImportSettings);
public SpyTextImportExportSettings GetTextImportExportSettings();
public SpyDoc ImportFromDatabase(SpyDatabaseConnection oImportSettings, SpyElementList oElementList);
public SpyDoc ImportFromSchema(SpyDatabaseConnection oImportSettings, String strTable, SpyDoc oSchemaDoc);
public SpyDoc ImportFromText(SpyTextImportExportSettings oImportSettings, SpyElementList oElementList);
public SpyDoc ImportFromWord(String sFile);
public void NewProject(String sPath, boolean bDiscardCurrent);
public void OpenProject(String sPath, boolean bDiscardCurrent, boolean bDialog);
public long ShowForm(String sName);
public void URLDelete(String sURL, String sUser, String sPassword);
public void URLMakeDirectory(String sURL, String sUser, String sPassword);

public int GetWarningNumber();
public String GetWarningText();

// since Version 2004R4
public SpyApplication GetApplication();
public SpyApplication GetParent();
public SpyDialogs GetDialogs();
public boolean GetVisible();
public void SetVisible(boolean i_bVisibility);
public long GetWindowHandle();

public void ReloadSettings();
public SpyFindInFilesResults FindInFiles(SpyFindInFilesDlg dlgSettings);
public boolean ShowFindInFiles(SpyFindInFilesDlg dlgSettings);
public void Selection(String sVal);

public long Status();
public int MajorVersion();
public int MinorVersion();
public String Edition();
public boolean IsAPISupported();
public long ServicePackVersion();
public void CreateXMLSchemaFromDBStructure(SpyDatabaseConnection oConnection, SpyElementList oTables);
}

3.5.3 SpyCodeGeneratorDlg (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.
Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

// since version 2004R4
public class SpyCodeGeneratorDlg
{
    public void ReleaseInstance();
    public SpyApplication GetApplication();
    public SpyDialogs GetParent();
    public long GetProgrammingLanguage();
    public void SetProgrammingLanguage(long i_eVal);
    public String GetTemplateFileName();
    public void SetTemplateFileName(String i_strVal);
    public String GetOutputPath();
    public void SetOutputPath(String i_strVal);
    public long GetOutputPathDialogAction();
    public void SetOutputPathDialogAction(long i_eVal);
    public long GetPropertySheetDialogAction();
    public void SetPropertySheetDialogAction(long i_eVal);
    public long GetOutputResultDialogAction();
    public void SetOutputResultDialogAction(long i_eVal);
    public long GetCPPSettings_DOMType();
    public void SetCPPSettings_DOMType(long i_eVal);
    public long GetCPPSettings_LibraryType();
    public void SetCPPSettings_LibraryType(long i_eVal);
    public boolean GetCPPSettings_UseMFC();
    public void SetCPPSettings_UseMFC(boolean i_bVal);
    public long GetCSharpSettings_ProjectType();
    public void SetCSharpSettings_ProjectType(long i_eVal);
}

3.5.4 SpyDatabaseConnection (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SpyDatabaseConnection
{
    public void ReleaseInstance();
    public String GetADOConnection();
    public void SetADOConnection(String sValue);
    public boolean GetAsAttributes();
    public void SetAsAttributes(boolean bValue);
public boolean GetCreateMissingTables();
public void SetCreateMissingTables( boolean bValue );
public boolean GetCreateNew();
public void SetCreateNew( boolean bValue );
public boolean GetExcludeKeys();
public void SetExcludeKeys( boolean bValue );
public String GetFile();
public void SetFile( String sValue );
public boolean GetCreateNew();
public void SetCreateNew( boolean bValue );
public String GetFile();
public void SetFile( String sValue );
public boolean GetExcludeKeys();
public void SetExcludeKeys( boolean bValue );
public long GetNumberDateTimeFormat();
public void SetNumberDateTimeFormat( long nValue );
public String GetODBCConnection();
public void SetODBCConnection( String sValue );
public String GetSQLSelect();
public void SetSQLSelect( String sValue );
public long GetTextFieldLen();
public void SetTextFieldLen( long nValue );

// since version 2004R4
public long GetDatabaseKind();
public void SetDatabaseKind( long nValue );

// since version 2008R2
public boolean GetCommentIncluded();
public void SetCommentIncluded( boolean bValue );
public String GetNullReplacement();
public void SetNullReplacement( String sValue );
public String GetDatabaseSchema();
public void SetDatabaseSchema( String sValue );

// since version 2010r3
public boolean GetPrimaryKeys();
public void SetPrimaryKeys( boolean bValue )
public boolean GetForeignKeys();
public void SetForeignKeys( boolean bValue )
public boolean GetUniqueKeys();
public void SetUniqueKeys( boolean bValue )
public long GetSchemaExtensionType();
public void SetSchemaExtensionType( long nValue )
public long GetSchemaFormat();
public void SetSchemaFormat( long nValue )
public long GetImportColumnsType();
public void SetImportColumnsType( long nValue )
}

3.5.5 SpyDialogs (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.
For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

// Since version 2004R4
public class SpyDialogs
{
    public SpyApplication GetApplication();
    public SpyApplication GetParent();
    public SpyCodeGeneratorDlg GetCodeGeneratorDlg();
    public SpyFileSelectionDlg GetFileSelectionDlg();
    public SpySchemaDocumentationDlg GetSchemaDocumentationDlg();
    public SpyGenerateSampleXMLDlg GetGenerateSampleXMLDlg();
    public SpyDTDSchemaGeneratorDlg GetDTDSchemaGeneratorDlg();
    public SpyFindInFilesDlg GetFindInFilesDlg();
}

3.5.6 SpyDoc (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SpyDoc
{
    public void ReleaseInstance();
    public void SetEncoding( String strEncoding );
    public void SetPathName( String strPath );
    public String GetPathName();
    public String GetTitle();
    public boolean IsModified();
    public void Save();
    public void Close( boolean bDiscardChanges );
    public void UpdateViews();
    public long GetCurrentViewMode();
    public boolean SwitchViewMode( long nMode );
    public SpyGridView GetGridView();
    public void SetActiveDocument();
    public void StartChanges();
    public void EndChanges();
    public void TransformXSL();
    public void AssignDTD( String sDTDFile, boolean bDialog );
    public void AssignSchema( String sSchemaFile, boolean bDialog );
    public void AssignXSL( String sXSLFile, boolean bDialog );
    public void ConvertDTDOrSchema( long nFormat, long nFrequentElements );
public SpyXMLData CreateChild(long nKind);
public void CreateSchemaDiagram(long nKind, String sName, String sFile);
public SpyDocEditView GetDocEditView();
public void ExportToDatabase(SpyXMLData oFromChild, SpyExportSettings oExportSettings, SpyDatabaseConnection oDatabaseConnection);
public void ExportToText(SpyXMLData oFromChild, SpyExportSettings oExportSettings, SpyTextImportExportSettings oTextSettings);
public void GenerateDTDOrSchema(long nFormat, int nValuesList, long nDetection, long nFrequentElements);
public SpyElementList GetExportElementList(SpyXMLData oFromChild, SpyExportSettings oExportSettings);
public SpyXMLData GetRootElement();
public String SaveInString(SpyXMLData oData, boolean bMarked);
public void SaveToURL(String sUrl, String sUser, String sPassword);
public String GetErrorString(); // See IsValid() or IsWellFormed()
public int GetErrorPos(); // See IsValid() or IsWellFormed()
public SpyXMLData GetBadData(); // See IsValid() or IsWellFormed()
public boolean IsValid();
public boolean IsWellFormed();
public boolean IsValid();
public boolean SaveAs(String i_strFileNameOrPath);
public String GetEncoding();
public SpyXMLData GetDataRoot();
public SpyDoc GenerateSampleXML(SpyGenerateSampleXMLDlg ipGenerateXMLDlg);
public boolean UpdateXMLData();
public String GetAsXMLString();
public void SetAsXMLString(String newVal);
public SpyDoc GenerateDTDOrSchemaEx(SpyDTDSchemaGeneratorDlg ipDTDSchemaGeneratorDlg);
public SpyDoc ConvertDTDOrSchemaEx(long nFormat, long nFrequentElements, String sOutputPath, long nOutputPathDialogAction);
public SpyTextView GetTextView();
public String[] GetSuggestions();
public void SetSuggestions(String[] aList);
public void SetSelection(String sVal);
// Since version 2009
public void TransformXSLEX( long nDialogAction );

// Since version 2010r3
public String CreateDBStructureFromXMLSchema( SpyDatabaseConnection oConnection, SpyElementList oTables, boolean bDropTableWithExistingName );
public SpyElementList GetDBStructureList( SpyDatabaseConnection oConnection );

3.5.7 SpyDocuments (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SpyDocuments
{
    public void ReleaseInstance();
    public long Count();
    public SpyDoc GetItem( long nNo );
    public SpyDoc NewFile( String strFile, String strType );
    public SpyDoc NewFileFromText( String nSource, String strType );
    public SpyDoc OpenFile( String sPath, boolean bDialog );
    public SpyDoc OpenURL( String sUrl, long nURIType, long nLoading, String sUser, String sPassword );
    public SpyDoc OpenURLDialog( String sURL, long nURIType, long nLoading, String sUser, String sPassword );
    // Since version 2011r2
    public SpyDoc NewAuthenticFile( String strSPSPath, String strXMLPath );
    public SpyDoc OpenAuthenticFile( String strSPSPath, String strXMLPath );
}

3.5.8 SpyDTDSchemaGeneratorDlg (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SpyDTDSchemaGeneratorDlg
{
    public void ReleaseInstance();
3.5.9 SpyElementList (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: **Programming Languages | Java.**

```java
public class SpyElementList {
    public void ReleaseInstance();
    public long GetCount();
    public SpyElementListItem getItem( long nIndex );
    public void RemoveElement( long nIndex );
}
```

3.5.10 SpyElementListItem (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.
For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SpyElementListItem
{
    public void ReleaseInstance();
    public long GetElementKind();
    public void SetElementKind( long nKind );
    public long GetFieldCount();
    public String GetName();
    public long GetRecordCount();
}

3.5.11 SpyExportSettings (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SpyExportSettings
{
    public void ReleaseInstance();
    public boolean GetCreateKeys();
    public SpyElementList GetElementList();
    public void SetElementList( SpyElementList obj );
    public boolean GetEntitiesToText();
    public void SetEntitiesToText( boolean bValue );
    public boolean GetExportAllElements();
    public void SetExportAllElements( boolean bValue );
    public boolean GetFromAttributes();
    public void SetFromAttributes( boolean bValue );
    public boolean GetFromSingleSubElements();
    public void SetFromSingleSubElements( boolean bValue );
    public boolean GetFromTextValues();
    public void SetFromTextValues( boolean bValue );
    public boolean GetIndependentPrimaryKey();
    public void SetIndependentPrimaryKey( boolean bValue );
    public long GetNamespace();
    public void SetNamespace( long nValue );
    public int GetSubLevelLimit();
    public void SetSubLevelLimit( int nValue );
}
3.5.12 SpyFileSelectionDlg (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
// Since version 2004R4
public class SpyFileSelectionDlg
{
    public void ReleaseInstance();
    public SpyApplication GetApplication();
    public SpyDialogs GetParent();
    public String GetFullName();
    public void SetFullName( String i_strName );
    public long GetDialogAction();
    public void SetDialogAction( long i_eAction );
}
```

3.5.13 SpyFindInFilesDlg (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SpyFindInFilesDlg
{
    public void ReleaseInstance();
    public SpyApplication GetApplication();
    public String GetFind();
    public void SetFind( String sNewVal );
    public boolean GetRegularExpression();
    public void SetRegularExpression( boolean bNewVal );
    public String GetReplace();
    public void SetReplace( String sNewVal );
    public boolean GetReplaceOnDisk();
    public void SetReplaceOnDisk( boolean bNewVal );
    public boolean GetDoReplace();
    public void SetDoReplace( boolean bNewVal );
    public boolean GetMatchWholeWord();
    public void SetMatchWholeWord( boolean bNewVal );
    public boolean GetMatchCase();
```
public void SetMatchCase ( boolean bNewVal );
public long GetSearchLocation ();
public void SetSearchLocation ( long nPosition );
public String GetStartFolder ();
public void SetStartFolder ( String sNewVal );
public boolean GetIncludeSubfolders ();
public void SetIncludeSubfolders ( boolean bNewVal );
public boolean GetSearchInProjectFilesDoExternal ();
public void SetSearchInProjectFilesDoExternal ( boolean bNewVal );
public String GetFileExtension ();
public void SetFileExtension ( String sNewVal );
public boolean GetIncludeSubfolders ();
public void SetIncludeSubfolders ( boolean bNewVal );
public boolean GetSearchInProjectFilesDoExternal ();
public void SetSearchInProjectFilesDoExternal ( boolean bNewVal );
public boolean GetAdvancedXMLSearch ();
public void SetAdvancedXMLSearch ( boolean bNewVal );
public boolean GetXMLElementNames ();
public void SetXMLElementNames ( boolean bNewVal );
public boolean GetXMLElementContents ();
public void SetXMLElementContents ( boolean bNewVal );
public boolean GetXMLAttributeNames ();
public void SetXMLAttributeNames ( boolean bNewVal );
public boolean GetXMLAttributeContents ();
public void SetXMLAttributeContents ( boolean bNewVal );
public boolean GetXMLComments ();
public void SetXMLComments ( boolean bNewVal );
public boolean GetXMLData ();
public void SetXMLData ( boolean bNewVal );
public boolean GetXMLPI ();
public void SetXMLPI ( boolean bNewVal );
public boolean GetXMLRest ();
public void SetXMLRest ( boolean bNewVal );
public boolean GetShowResult ();
public void SetShowResult ( boolean bNewVal );
}

3.5.14 SpyFindInFilesMatch (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.
3.5.15 SpyFindInFilesResult (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SpyFindInFilesResult {
    public void ReleaseInstance();
    public long Count();
    public SpyFindInFilesMatch GetItem( long nNo );
    public String GetPath();
    public SpyDoc GetDocument();
}
```

3.5.16 SpyFindInFilesResults (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SpyFindInFilesResults {
    public void ReleaseInstance();
    public long Count();
    public SpyFindInFilesResult GetItem( long nNo );
}
```

3.5.17 SpyGenerateSampleXMLDlg (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SpyGenerateSampleXMLDlg {
}
```
The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.
3.5.19  SpyProject (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SpyProject {
    public void ReleaseInstance();
    public void CloseProject( boolean bDiscardChanges, boolean bCloseFiles, boolean bDialog );
    public String GetProjectFile();
    public void SetProjectFile( String sFile );
    public SpyProjectItems GetRootItems();
    public void SaveProject();
    public void SaveProjectAs( String sPath, boolean bDialog );
}
```

3.5.20  SpyProjectItem (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SpyProjectItem {
    public void ReleaseInstance();
    public SpyProjectItems GetChildItems();
    public String GetFileExtensions();
    public void SetFileExtensions( String sExtensions );
    public long GetItemType();
    public String GetName();
    public SpyDoc Open();
    public SpyProjectItem GetParentItem();
    public String GetPath();
    public String GetValidateWith();
    public void SetValidateWith( String sVal );
    public String GetXMLForXSLTransformation();
    public void SetXMLForXSLTransformation( String sVal );
    public String GetXSLForXMLTransformation();
    public void SetXSLForXMLTransformation( String sVal );
```
public String GetXSLTransformationFileExtension();
public void SetXSLTransformationFileExtension( String sVal );
public String GetXSLTransformationFolder();
public void SetXSLTransformationFolder( String sVal );
}

3.5.21 SpyProjectItems (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SpyProjectItems
{
    public void ReleaseInstance();
    public void AddFile( String sPath );
    public void AddFolder( String sName );
    public void AddURL( String sURL, long nURLType, String sUser, String sPassword, boolean bSave );
    public long Count();
    public SpyProjectItem GetItem( long nNumber );
    public void RemoveItem( SpyProjectItem oItemToRemove );
}

3.5.22 SpySchemaDocumentationDlg (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

// Since version 2004R4
public class SpySchemaDocumentationDlg
{
    public void ReleaseInstance();
    public SpyApplication GetApplication();
    public SpyDialogs GetParent();

    public String GetOutputFile();
    public void SetOutputFile( String i_strVal );
    public long GetOutputFormat();
    public void SetOutputFormat( long i_eVal );
public boolean GetShowResult();
public void SetShowResult( boolean i_bVal );
public long GetOptionsDialogAction();
public void SetOptionsDialogAction( long i_eVal );
public long GetOutputFileDialogAction();
public void SetOutputFileDialogAction( long i_eVal );
public boolean GetShowProgressBar();
public void SetShowProgressBar( boolean i_bVal );
public void IncludeAll( boolean i_bInclude );
public boolean GetIncludeIndex();
public void SetIncludeIndex( boolean i_bVal );
public boolean GetIncludeGlobalElements();
public void SetIncludeGlobalElements( boolean i_bVal );
public boolean GetIncludeLocalElements();
public void SetIncludeLocalElements( boolean i_bVal );
public boolean GetIncludeGroups();
public void SetIncludeGroups( boolean i_bVal );
public boolean GetIncludeComplexTypes();
public void SetIncludeComplexTypes( boolean i_bVal );
public boolean GetIncludeSimpleTypes();
public void SetIncludeSimpleTypes( boolean i_bVal );
public boolean GetIncludeAttributeGroups();
public void SetIncludeAttributeGroups( boolean i_bVal );
public boolean GetIncludeRedefines();
public void SetIncludeRedefines( boolean i_bVal );
public void AllDetails( boolean i_bDetailsOn );
public boolean GetShowDiagram();
public void SetShowDiagram( boolean i_bVal );
public boolean GetShowNamespace();
public void SetShowNamespace( boolean i_bVal );
public boolean GetShowType();
public void SetShowType( boolean i_bVal );
public boolean GetShowChildren();
public void SetShowChildren( boolean i_bVal );
public boolean GetShowUsedBy();
public void SetShowUsedBy( boolean i_bVal );
public boolean GetShowProperties();
public void SetShowProperties( boolean i_bVal );
public boolean GetShowSingleFacets();
public void SetShowSingleFacets( boolean i_bVal );
public boolean GetShowPatterns();
public void SetShowPatterns( boolean i_bVal );
public boolean GetShowEnumerations();
public void SetShowEnumerations( boolean i_bVal );
public boolean GetShowAttributes();
public void SetShowAttributes( boolean i_bVal );
public boolean GetShowIdentityConstraints();
public void SetShowIdentityConstraints( boolean i_bVal );
public boolean GetShowAnnotations();
public void SetShowAnnotations( boolean i_bVal );
public boolean GetShowSourceCode();
public void SetShowSourceCode( boolean i_bVal );

// Since version 2009
public boolean GetEmbedDiagrams();
public void SetEmbedDiagrams( boolean i_bVal );
public long GetDiagramFormat();
public void SetDiagramFormat( long i_nVal );
public boolean GetIncludeGlobalAttributes();
public boolean GetIncludeLocalAttributes();
public boolean GetIncludeReferencedSchemas();
public boolean GetIncludeReferencedSchemas( boolean i_bVal );
public boolean GetMultipleOutputFiles();
public void SetMultipleOutputFiles( boolean i_bVal );

// Since version 2010
public boolean GetEmbedCSSInHTML();
public void SetEmbedCSSInHTML( boolean i_bVal );
public void SetCreateDiagramsFolder();
public void SetCreateDiagramsFolder( boolean i_bVal );

// Since version 2010r3
public boolean GetGenerateRelativeLinks();
public void SetGenerateRelativeLinks( boolean i_bVal );

// Since version 2011r2
public boolean GetUseFixedDesign();
public void SetUseFixedDesign( boolean i_bVal );
public String GetSPSFile();
public void SetSPSFile( String i_strVal );
}

3.5.23 SpyTextImportExportSettings (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SpyTextImportExportSettings {
}
public void ReleaseInstance();
public String GetDestinationFolder();
public void SetDestinationFolder( String sVal );
public long GetEnclosingCharacter();
public void SetEnclosingCharacter( long nEnclosing );
public String GetEncoding();
public void SetEncoding( String sVal );
public long GetEncodingByteOrder();
public void SetEncodingByteOrder( long nByteOrder );
public long GetFieldDelimiter();
public void SetFieldDelimiter( long nDelimiter );
public String GetFileExtension();
public void SetFileExtension( String sVal );
public boolean GetHeaderRow();
public void SetHeaderRow( boolean bVal );
public String GetImportFile();
public void SetImportFile( String sVal );
}

3.5.24 SpyTextView (obsolete)

public class SpyTextView
{
  public void ReleaseInstance();
  public SpyApplication GetApplication();
  public SpyDoc GetParent();
  public long LineFromPosition( long nCharPos );
  public long PositionFromLine( long nLine );
  public long LineLength( long nLine );
  public String GetSelText();
  public void SetSelText( String sText );
  public String GetRangeText( long nPosFrom, long nPosTill );
  public void ReplaceText( long nPosFrom, long nPosTill, String sText );
  public void MoveCaret( long nDiff );
  public void GoToLineChar( long nLine, long nChar );
  public void SelectText( long nPosFrom, long nPosTill );
  public long GetSelectionStart();
  public void SetSelectionStart( long nNewVal );
  public long GetSelectionEnd();
  public void SetSelectionEnd( long nNewVal );
  public String GetText();
  public void SetText( String sText );
  public long LineCount();
  public long Length();
}

3.5.25 SpyXMLData (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.
public class SpyXMLData
{
    public void ReleaseInstance();
    public void AppendChild(SpyXMLData oNewData);
    public void EraseAllChildren();
    public void EraseCurrentChild();
    public SpyXMLData GetCurrentChild();
    public SpyXMLData GetFirstChild(long nKind);
    public SpyXMLData GetNextChild();
    public boolean GetHasChildren();
    public void InsertChild(SpyXMLData oNewData);
    public boolean IsSameNode(SpyXMLData oToComp);
    public long GetKind();
    public boolean GetMayHaveChildren();
    public String GetName();
    public void SetName(String sValue);
    public SpyXMLData GetParent();
    public String GetTextValue();
    public void SetTextValue(String sValue);
}

3.5.26 Authentic (obsolete)

SpyAuthenticRange (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.
public void SetLastTextPosition( long nTextPosition );
public String GetText();
public void SetText( String strText );
public boolean PerformAction( long eAction, String strElementName );
public boolean CanPerformAction( long eAction, String strElementName );
public String[] CanPerformActionWith( long eAction );
public SpyAuthenticRange GoTo( long eKind, long nCount, long nFrom );
public SpyAuthenticRange SelectNext( long eKind );
public SpyAuthenticRange SelectPrevious( long eKind );
public SpyAuthenticRange MoveBegin( long eKind, long nCount );
public SpyAuthenticRange MoveEnd( long eKind, long nCount );
public SpyAuthenticRange ExpandTo( long eKind );
public SpyAuthenticRange CollapsToBegin();
public SpyAuthenticRange CollapsToEnd();
public SpyAuthenticRange GoToNextCursorPosition();
public SpyAuthenticRange GoToPreviousCursorPosition();
public boolean IsEmpty();
public boolean IsEqual( SpyAuthenticRange ipCmp );
public SpyAuthenticRange Clone();
public SpyAuthenticRange SetFromRange( SpyAuthenticRange ipSrc );
public boolean Delete();
public boolean Cut();
public boolean Copy();
public boolean Paste();
public SpyXMLData GetFirstXMLData();
public void SetFirstXMLData( SpyXMLData objXMLDataPtr );
public long GetFirstXMLDataOffset();
public void SetFirstXMLDataOffset( long nOffset );
public SpyXMLData GetLastXMLData();
public void SetLastXMLData( SpyXMLData objXMLDataPtr );
public long GetLastXMLDataOffset();
public void SetLastXMLDataOffset( long nOffset );
public String[] GetElementHierarchy();
public String[] GetElementAttributeNames( String strElementName );
public boolean HasElementAttribute( String strElementName, String strAttributeName );
public String GetElementAttributeValue( String strElementName, String strAttributeName, String strNewValue );
public String[] GetEntityNames();
public void InsertEntity( String strEntityName );
public boolean IsInDynamicTable();
public boolean AppendRow();
public boolean InsertRow();
public boolean DuplicateRow();
public boolean DeleteRow();
public boolean MoveRowUp();
public boolean MoveRowDown();

// Since version 2004R4
public boolean IsCopyEnabled();
public boolean IsCutEnabled();
SpyAuthenticView (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

// Since version 2004R3
public class SpyAuthenticView
{
    public void ReleaseInstance();
    public SpyApplication GetApplication();
    public SpyDoc GetParent();
    public SpyAuthenticRange GetSelection();
    public void SetSelection( SpyAuthenticRange obj );
    public SpyAuthenticRange GetDocumentBegin();
    public SpyAuthenticRange GetDocumentEnd();
    public SpyAuthenticRange GetWholeDocument();
    public long GetMarkupVisibility();
    public void SetMarkupVisibility( long eSpyAuthenticMarkupVisibility );
    public SpyAuthenticRange GoTo( long eKind, long nCount, long nFrom );
    public void Print( boolean bWithPreview, boolean bPromptUser );
    public boolean Undo();
    public boolean Redo();
    public void UpdateXMLInstanceEntities();

    // Since version 2004R4
    public String GetAsXMLString();
    public void SetAsXMLString( String i_strXML );
    public SpyXMLData GetXMLDataRoot();
    public boolean IsUndoEnabled();
    public boolean IsRedoEnabled();
}

SpyDocEditSelection (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.
public class SpyDocEditSelection{
    public void ReleaseInstance();
    public SpyXMLData GetEnd();
    public long GetEndTextPosition();
    public SpyXMLData GetStart();
    public long GetStartTextPosition();
}

SpyDocEditView (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SpyDocEditView{
    public void ReleaseInstance();
    public void ApplyTextState( String sElementName );
    public SpyDocEditSelection GetCurrentSelection();
    public void EditClear();
    public void EditCopy();
    public void EditCut();
    public void EditPaste();
    public void EditRedo();
    public void EditSelectAll();
    public void EditUndo();
    public SpyXMLData GetNextVisible( SpyXMLData oElement );
    public SpyXMLData GetPreviousVisible( SpyXMLData oElement );
    public boolean GetIsEditClearEnabled();
    public boolean GetIsEditCopyEnabled();
    public boolean GetIsEditCutEnabled();
    public boolean GetIsEditPasteEnabled();
    public boolean GetIsEditUndoEnabled();
    public boolean GetIsRowAppendEnabled();
    public boolean GetIsRowDeleteEnabled();
    public boolean GetIsRowDuplicateEnabled();
    public boolean GetIsRowInsertEnabled();
    public boolean GetIsRowMoveDownEnabled();
    public boolean GetIsRowMoveUpEnabled();
    public boolean IsTextStateApplied( String sElementName );
public boolean IsTextStateEnabled( String sElementName );
public void LoadXML( String sXML );
public void MarkUpView( long nKind );
public void RowAppend();
public void RowDelete();
public void RowDuplicate();
public void RowInsert();
public void RowMoveDown();
public void RowMoveUp();
public String SaveXML();
public void SelectionMoveTabOrder( boolean bForward, boolean bTag );
public boolean SelectionSet( SpyXMLData oStart, long nStartPos, SpyXMLData oEndElement, long nEndPos );
public SpyXMLData GetXMLRoot();
public String[] GetAllowedElements( long nAction, SpyXMLData oStartPtr, SpyXMLData oEndPtr );

3.5.27 Predefined constants (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

This section lists all classes that define the predefined constants used by the Java interface.

SPYApplicationStatus (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SPYApplicationStatus
{
    public final static long spyApplicationStatus_Running = 0;
    public final static long spyApplicationStatus_AfterLicenseCheck = 1;
    public final static long spyApplicationStatus_BeforeLicenseCheck = 2;
    public final static long spyApplicationStatus_ConcurrentLicenseCheckFailed = 3;
    public final static long spyApplicationStatus_ProcessingCommandLine = 4;
}
SPYAttributeTypeDefinition (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SPYAttributeTypeDefinition {
    public final static long spyMergedGlobal = 0;
    public final static long spyDistinctGlobal = 1;
    public final static long spyLocal = 2;
}
```

SPYAuthenticActions (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SPYAuthenticActions {
    public final static long spyAuthenticInsertAt = 0;
    public final static long spyAuthenticApply = 1;
    public final static long spyAuthenticClearSurr = 2;
    public final static long spyAuthenticAppend = 3;
    public final static long spyAuthenticInsertBefore = 4;
    public final static long spyAuthenticRemove = 5;
}
```
SPYAuthenticDocumentPosition (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SPYAuthenticDocumentPosition {
    public final static long spyAuthenticDocumentBegin = 0;
    public final static long spyAuthenticDocumentEnd = 1;
    public final static long spyAuthenticRangeBegin = 2;
    public final static long spyAuthenticRangeEnd = 3;
}
```

SPYAuthenticElementKind (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SPYAuthenticElementKind {
    public final static long spyAuthenticChar = 0;
    public final static long spyAuthenticWord = 1;
    public final static long spyAuthenticLine = 3;
    public final static long spyAuthenticParagraph = 4;
    public final static long spyAuthenticTag = 6;
    public final static long spyAuthenticDocument = 8;
    public final static long spyAuthenticTable = 9;
    public final static long = 10;
}
```
SpyAuthenticTableRow

```java
public final static long spyAuthenticTableRow = 11;
SpyAuthenticTableColumn```

**SpyAuthenticMarkupVisibility (obsolete)**

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: [Programming Languages | Java](#).

```java
public class SPYAuthenticMarkupVisibility {
    public final static long spyAuthenticMarkupHidden = 0;
    public final static long spyAuthenticMarkupSmall = 1;
    public final static long spyAuthenticMarkupLarge = 2;
    public final static long spyAuthenticMarkupMixed = 3;
}
```

**SpyDatabaseKind (obsolete)**

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: [Programming Languages | Java](#).

```java
public class SPYLoading {
    public final static long spyDB_Access = 0;
    public final static long spyDB_SQLServer = 1;
    public final static long spyDB_Oracle = 2;
    public final static long spyDB_Sybase = 3;
    public final static long spyDB_MySQL = 4;
}
```
public final static long spyDB_DB2 = 5;
public final static long spyDB_Other = 6;
public final static long spyDB_Unspecified = 7;
}

SPYDialogAction (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SPYDialogAction {
    public final static long spyDialogOK = 0;
    public final static long spyDialogCancel = 1;
    public final static long spyDialogUserInput = 2;
}

SPYDOMType (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SPYDOMType {
    public final static long spyDOMType_msxml4 = 0;
    public final static long spyDOMType_xerces = 1;
}
SPYDTDSchemaFormat (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SPYDTDSchemaFormat {
    public final static long spyDTD = 0;
    public final static long spyDCD = 1;
    public final static long spyXMLData = 2;
    public final static long spyBizTalk = 3;
    public final static long spyW3C = 4;
}
```

SPYEncodingByteOrder (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SPYEncodingByteOrder {
    public final static long spyNONE = 0;
    public final static long spyLITTLE_ENDIAN = 1;
    public final static long spyBIG_ENDIAN = 2;
}
```

SPYExportNamespace (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.
public class SPYExportNamespace
{
    public final static long spyNoNamespace = 0;
    public final static long spyReplaceColonWith Underscore = 1;
}

SPYFindInFilesSearchLocation (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SPYFindInFilesSearchLocation
{
    public final static long spyFindInFiles_Documents = 0;
    spyFindInFiles_Documents
    public final static long spyFindInFiles_Project = 1;
    spyFindInFiles_Project
    public final static long spyFindInFiles_Folder = 2;
    spyFindInFiles_Folder
}

SPYFrequentElements (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SPYFrequentElements
{
    public final static long spyGlobalElements = 0;
    spyGlobalElements
    public final static long spyGlobalComplexType = 1;
    spyGlobalComplexType
}

SPYImageKind (obsolete)

The objects described in this section (Application API for Java) are obsolete
from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SPYImageKind {
    public final static long spyImageType_PNG = 0;
    public final static long spyImageType_EMF = 1;
}
```

SPYImportColumnsType (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SPYLibType {
    public final static long spyLibType_static = 0;
    public final static long spyLibType_dll = 1;
}
```

SPYLoading (obsolete)

The objects described in this section (Application API for Java) are obsolete
from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```
public class SPYLoading {
    public final static long spyUseCacheProxy = 0;
    public final static long spyReload = 1;
}
```

SPYNumberDateTimeFormat (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```
public class SPYNumberDateTimeFormat {
    public final static long spySystemLocale = 0;
    public final static long spySchemaCompatible = 1;
}
```

SPYProgrammingLanguage (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```
public class SPYLoading {
    public final static long spyUndefinedLanguage = -1;
    public final static long spyJava = 0;
    public final static long spyCpp = 1;
    public final static long spyCSharp = 2;
}
```
SPYProjectItemTypes (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SPYProjectItemTypes {
    public final static long spyUnknownItem = 0;
    public final static long spyFileItem = 1;
    public final static long spyFolderItem = 2;
    public final static long spyURLItem = 3;
}
```

SPYProjectType (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SPYProjectType {
    public final static long spyVisualStudioProject = 0;
    public final static long spyVisualStudio2003Project = 1;
    public final static long spyBorlandProject = 2;
    public final static long spyMonoMakefile = 3;
}
```
SPYSampleXMLGenerationOptimization (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SPYSampleXMLGenerationOptimization {
    public final static long spySampleXMLGen_Optimized = 0;
    public final static long spySampleXMLGen_NonMandatoryElements = 1;
    public final static long spySampleXMLGen_Everything = 2;
}
```

SPYSampleXMLGenerationSchemaOrDTDAssignment (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SPYSampleXMLGenerationSchemaOrDTDAssignment {
    public final static long spySampleXMLGen_AssignRelatively = 0;
    public final static long spySampleXMLGen_AssignAbsolutely = 1;
    public final static long spySampleXMLGen_DoNotAssign = 2;
}
```

SPYSchemaDefKind (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.
public class SPYSchemaDefKind
{
    public final static long spyKindElement = 0;
    public final static long spyKindComplexType
        = 1;
    public final static long spyKindSimpleType
        = 2;
    public final static long spyKindGroup
        = 3;
    public final static long spyKindModel
        = 4;
    public final static long spyKindAny
        = 5;
    public final static long spyKindAttr
        = 6;
    public final static long spyKindAttrGroup
        = 7;
    public final static long spyKindAttrAny
        = 8;
    public final static long spyKindIdentityUnique
        = 9;
    public final static long spyKindIdentityKey
        = 10;
    public final static long spyKindIdentityKeyRef
        = 11;
    public final static long spyKindIdentitySelector
        = 12;
    public final static long spyKindIdentityField
        = 13;
    public final static long spyKindNotation
        = 14;
    public final static long spyKindInclude
        = 15;
    public final static long spyKindImport
        = 16;
    public final static long spyKindRedefine
        = 17;
    public final static long spyKindFacet
        = 18;
    public final static long spyKindSchema
        = 19;
    public final static long spyKindCount
        = 20;
}

SPYSchemaDocumentationFormat (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.
public final static long spySchemaDoc_PDF = 3;
}

SPYSchemaExtensionType (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SPYSchemaExtensionType
{
    public final static long spySchemaExtension_None = 0;
    public final static long spySchemaExtension_SQL_XML = 1;
    public final static long spySchemaExtension_MS_SQL_Server = 2;
    public final static long spySchemaExtension_Oracle = 3;
}

SPYSchemaFormat (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SPYSchemaFormat
{
    public final static long spySchemaFormat_Hierarchical = 0;
    public final static long spySchemaFormat_Flat = 1;
}

SPYTextDelimiters (obsolete)

The objects described in this section (Application API for Java) are obsolete
from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SPYTextDelimiters {
    public final static long spyTabulator = 0;
    public final static long spySemicolon = 1;
    public final static long spyComma = 2;
    public final static long spySpace = 3;
}
```

SPYTextEnclosing (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SPYTextEnclosing {
    public final static long spyNoEnclosing = 0;
    public final static long spySingleQuote = 1;
    public final static long spyDoubleQuote = 2;
}
```

SPYTypeDetection (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

```java
public class SPYTypeDetection {
}
```
public final static long spyBestPossible = 0;
public final static long spyNumbersOnly = 1;
public final static long spyNoDetection = 2;

SPYURLTypes (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SPYURLTypes {
    public final static long spyURLTypeAuto = (-1);
    public final static long spyURLTypeXML = 0;
    public final static long spyURLTypeDTD = 1;
}

SpyViewModes (obsolete)

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: Programming Languages | Java.

public class SPYViewModes {
    public final static long spyViewGrid = 0;
    public final static long spyViewText = 1;
    public final static long spyViewBrowser = 2;
    public final static long spyViewSchema = 3;
    public final static long spyViewContent = 4;
    public final static long spyViewAuthentic = 4;
public final static long spyViewWSDL = 5;
public final static long spyViewZIP = 6;
public final static long spyViewEditionInfo = 7;
}

**SPYWhitespaceComparison (obsolete)**

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: [Programming Languages | Java](#).

```java
public class SPYWhitespaceComparison {
    public final static long spyCompareAsIs = 0;
    public final static long spyCompareNormalized = 1;
    public final static long spyStripAll = 2;
}
```

**SPYXMLDataKind (obsolete)**

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: [Programming Languages | Java](#).

```java
public class SPYXMLDataKind {
    public final static long spyXMLDataXMLDocStruct = 0;
    public final static long spyXMLDataXMLEntityDocStruct = 1;
    public final static long spyXMLDataDTDDocStruct = 2;
    public final static long spyXMLDataXML = 3;
    public final static long spyXMLDataElement = 4;
    public final static long spyXMLDataAttr = 5;
```
```java
public final static long spyXMLDataText = 6;
public final static long spyXMLDataCData = 7;
public final static long spyXMLDataComment = 8;
public final static long spyXMLDataPI = 9;
public final static long spyXMLDataDefDoctype = 10;
public final static long spyXMLDataDefExternalID = 11;
public final static long spyXMLDataDefElement = 12;
public final static long spyXMLDataDefAttlist = 13;
public final static long spyXMLDataDefEntity = 14;
public final static long spyXMLDataDefNotation = 15;
public final static long spyXMLDataKindsCount = 16;
```
4 ActiveX Integration

The XMLSpy 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 XMLSpy Integration Package must be installed (see https://www.altova.com/components/download). Ensure that you install XMLSpy first, and then the XMLSpy 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 XMLSpy (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 XMLSpy 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 XMLSpy graphical user interface according to your custom requirements.

Integration at document level means embedding XMLSpy into your own application piece-by-piece. This includes implementing not only the main XMLSpy 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 XMLSpy as a Visual Studio plug-in, see XMLSpy in Visual Studio.
4.1 Prerequisites

To integrate the XMLSpy ActiveX control into a custom application, the following must be installed on your computer:

- XMLSpy
- The XMLSpy Integration Package, available for download at https://www.altova.com/components/download

To integrate the 64-bit ActiveX control, install the 64-bit versions of XMLSpy and XMLSpy Integration Package. For applications developed under Microsoft .NET platform with Visual Studio, both the 32-bit and 64-bit versions of XMLSpy and XMLSpy Integration Package must be installed, as explained below.

Microsoft .NET (C#, VB.NET) with Visual Studio

To integrate the XMLSpy 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
- XMLSpy 32-bit and XMLSpy 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:

- XMLSpy 32-bit and XMLSpy 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)
- XMLSpy 64-bit and XMLSpy 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 XMLSpy 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
- XMLSpy and XMLSpy Integration Package

Note: To run the 64-bit version of the XMLSpy ActiveX control, use a 64-bit version of Eclipse, as well as the 64-bit version of XMLSpy and the XMLSpy Integration Package.

XMLSpy 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):
- XMLSpy
- The XMLSpy Integration Package
- The custom integration code or application.
4.2 Adding the ActiveX Controls to the Toolbox

To use the XMLSpy 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 XMLSpyControl, XMLSpyControl Document, and XMLSpyControl Placeholder.

In case the controls above are not available, follow the steps below:

1. On the COM Components tab, click Browse, and select the XMLSpyControl.ocx file from the XMLSpy installation folder. Remember that the XMLSpy 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 XMLSpy ActiveX controls become available in the Visual Studio Toolbox.
Note: For an application-level integration, only the XMLSpyControl ActiveX control is used (see Integration at Application Level). The XMLSpyControl Document and XMLSpyControl Placeholder controls are used for document-level integration (see Integration at Document Level).
4.3 Integration at Application Level

Integration at application level allows you to embed the complete interface of XMLSpy into a window of your application. With this type of integration, you get the whole user interface of XMLSpy, including all menus, toolbars, the status bar, document windows, and helper windows. Customization of the application's user interface is restricted to what XMLSpy provides. This includes rearrangement and resizing of helper windows and customization of menus and toolbars.

The only ActiveX control you need to integrate is `XMLSpyControl`. Do not instantiate or access `XMLSpyControlDocument` or `XMLSpyControlPlaceHolder` ActiveX controls when integrating at application-level.

If you have any initialization to do or if you want to automate some behaviour of XMLSpy, use the properties, methods, and events described for `XMLSpyControl`. Consider using `XMLSpyControl.Application` for more complex access to XMLSpy functionality.

For an example that shows how the XMLSpy 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 XMLSpy 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 `XMLSpyControl` from the toolbox onto your new form.
5. Select the `XMLSpyControl` on the form, and, in the Properties window, set the `IntegrationLevel` property to `ICActiveXIntegrationOnApplicationLevel`. 
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 dialog](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).
4.4 Integration at Document Level

Compared to integration at application level, integration at document level is a more complex, yet more flexible way to embed XMLSpy functionality into your application by means of ActiveX controls. With this approach, your code can access selectively the following parts of the XMLSpy user interface:

- Document editing window
- Project window
- Entry helper windows
- Validator output window
- XPath profiler window
- XPath dialog window
- XSLT/XQuery debugger windows

As mentioned in Integration at Application Level, for an ActiveX integration at application level, only one control is required, namely the XMLSpyControl. However, for an ActiveX integration at document level, functionality XMLSpy is provided by the following ActiveX controls:

1. XMLSpyControl
2. XMLSpyControl Document
3. XMLSpyControl Placeholder

These controls are supplied by the XMLSpyControl.ocx file available in the application installation folder of XMLSpy. 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 XMLSpyControl in your application. Instantiating this control is mandatory; it enables support for the XMLSpyControl Document and XMLSpyControl 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 XMLSpyControl; this might lead to unexpected results. Use the corresponding open methods of XMLSpyControl Document and XMLSpyControl PlaceHolder instead.

2. Create at least one instance of XMLSpyControl Document in your application. This control supplies the document editing window of XMLSpy 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 XMLSpyControl Placeholder control for each
additional window (other than the document window) that must be available to your application.

Instances of XMLSpyControl PlaceHolder allow you to selectively embed additional windows of XMLSpy 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 XMLSpyControlPlaceholderWindow.

Note: Use only one XMLSpyControl PlaceHolder for each window identifier.

For placeholder controls that select the XMLSpy project window, additional methods are available. Use OpenProject to load a XMLSpy project. Use the property Project and the methods and properties from the XMLSpy 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 XMLSpy 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 XMLSpyControl from the toolbox onto your new form.
4. Set the IntegrationLevel property of the XMLSpyControl to ICAcativeXIntegrationOnDocumentLevel, and the Visible property to False. You can do this either from code or from the Properties window.
5. Drag the XMLSpyControl Document from the toolbox onto the form. This control provides the main document window of XMLSpy to your application, so you may need to resize it to a reasonable size for a document.
6. Optionally, add one or more XMLSpyControl 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 `XMLSpyControl Placeholder` control to a valid window identifier. For the list of valid values, see `XMLSpyControlPlaceholderWindow`.

8. Add commands to your application (at minimum, you will need to open, save and close documents), as shown below.

**Querying XMLSpy Commands**

When you integrate at document level, no XMLSpy 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 `XMLSpyControl`.
- 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 XMLSpy, use the `Exec` method.
- To query if a command is currently enabled or disabled, use the `QueryStatus` method.

This enables you to flexibly integrate XMLSpy commands into your application's menus and toolbars.

Your installation of XMLSpy also provides you with command label images used within XMLSpy. See the folder `<ApplicationFolder>\Examples\ActiveXImages` of your XMLSpy 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 XMLSpy, use the properties, methods, and events described for the XMLSpyControl, XMLSpyControl Document, and XMLSpyControl Placeholder.

For more complex access to XMLSpy functionality, consider using the following properties:

- XMLSpyControl.Application
- XMLSpyControlDocument.Document
- XMLSpyControlPlaceHolder.Project

These properties give you access to the XMLSpy automation interface (XMLSpyAPI)

**Note:** To open a document, always use XMLSpyControlDocument.Open or XMLSpyControlDocument.New on the appropriate document control. To open a project, always use XMLSpyControlPlaceholder.OpenProject on a placeholder control embedding a XMLSpy project window.

For examples that show how to instantiate and access the necessary controls in different programming environments, see ActiveX Integration Examples.
4.5 **ActiveX Integration Examples**

This section contains examples of XMLSpy 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 XMLSpy installation.

4.5.1 **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).

**Running the Sample C# Solution**

The sample Visual Studio solution available in the folder `<ApplicationFolder>\Examples\ActiveX\C#` illustrates how to consume the XMLSpy 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 `XMLSpyControlDocument` and one instance of `XMLSpyControlPlaceHolder` 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>XMLSpyControl</td>
<td>IntegrationLevel</td>
<td>ICAcitiveXIntegrationOnDocu mentalityLevel</td>
</tr>
<tr>
<td>XPathDialog</td>
<td>PlaceholderWindowID</td>
<td>16</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 created and contains an editing window with an empty XML document and a XPath Dialog window of XMLSpy at the bottom. Use File | Open to open any XML file from the XMLSpy examples folder. The file is loaded and displayed. After you load the document, you can start using the XPath dialog. Note that you may need to slightly drag the lower-right corner of the form to cause the dialog to redraw itself and display its contents.

### 4.5.2 HTML

The code listings in this section show how to integrate the XMLSpyControl at application level and document level. Source code for all examples is available in the folder `<ApplicationFolder>\Examples\ActiveX\HTML` of your XMLSpy 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.

#### HTML Integration at Application Level

This example shows a simple integration of the XMLSpy control at application-level into a HTML page. The integration is described in the following sections:

- Instantiate a XMLSpyControl 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 XMLSpy installation:

`<ApplicationFolder>\Examples\ActiveX\HTML\XMLSpyActiveX_ApplicationLevel.htm`

**Instantiate the Control**

The HTML **Object** tag is used to create an instance of the XMLSpyControl. The **Classid** is that of XMLSpyControl. Width and height specify the window size. No additional parameters are necessary, since application-level is the default.

```html
<object id="objXMLSpyControl"
Classid="clsid:a258bba2-3835-4c16-8590-72b44f52c471"
width="1000"
height="700"
viewastext>
</object>
```

**Add Button to Open Default Document**

As a simple example of how to automate some tasks, we add a button to the page:
When clicked, a predefined document will be opened in the XMLSpyControl. The `MakeAbsolutePath` method creates an absolute path using the location of the script as a base path.

```javascript
function BtnOpenMEFile()
{
    var strPath = MakeAbsolutePath("OrgChart.pxf");
    var objDoc = objXMLSpyControl.Open(strPath);

    if (objDoc == null)
        alert("Unable to locate Orgchart.pxf at: " + objXMLSpyControl.BaseHref);
}
```

**Add Buttons for Code Generation**

For direct access, we want to have a button that will validate the current document. The method is similar to that used in the previous section.

First comes the button:

```html
<input type="button" value="Open OrgChart.pxf" onclick="BtnOpenMEFile()"/>
```

Then we provide the script that will validate the current document.

```javascript
// check validity of current document.
// if validation fails, show validation result in alert box .
function BtnValidate()
{
    // get top-level object of automation interface
    var objApp = objXMLSpyControl.Application;

    // get the active document
    var objDocument = objApp.ActiveDocument;

    if (objDocument == null)
        alert("no active document found");
    else
    {
        // define as arrays to support their usage as return parameters
        var errorText = new Array(1);
        var errorPos = new Array(1);
        var badData = new Array(1);

        var valid = objDocument.IsValid(errorText, errorPos, badData);

        if (! valid)
        {
```
// compose the error description
var text = errorText;

// access that XMLData object only if filled in
if (badData[0] !== null)
    text += "(" + badData[0].Name + "/" + badData[0].TextValue + 
")";

    alert("Validation error[" + errorPos + "]: " + text);
} else
    alert("Document is valid");
}

Connect to Custom Events
The example implements two event callbacks for XMLSpyControl custom events to show the principle:

<!-- ----------------------------------------------------------- -->
<!--  custom event 'OnDocumentOpened" of XMLSpyControl object  -->
<SCRIPT FOR="objXMLSpyControl" event="OnDocumentOpened( objDocument )"
LANGUAGE="javascript">
    // alert("Document " + objDocument.Name + ", opened!");
</SCRIPT>

<!-- ----------------------------------------------------------- -->
<!--  custom event 'OnDocumentClosed" of XMLSpyControl object  -->
<SCRIPT FOR="objXMLSpyControl" event="OnDocumentClosed( objDocument )"
LANGUAGE="javascript">
    // alert("Document " + objDocument.Name + ", closed!");
</SCRIPT>

HTML Integration at Document Level
This example shows an integration of the XMLSpy control at document-level into a HTML page. The following topics are covered:

- Instantiate a XMLSpyControl ActiveX control object in HTML code
- Instantiate a XMLSpyControlDocument ActiveX control to allow editing a XMLSpy file
- Instantiate one XMLSpyControlPlaceHolder to alternatively host one of the XMLSpy helper windows
- Instantiate one XMLSpyControlPlaceHolder ActiveX control to show the XMLSpy validation output window
- Create a simple custom toolbar for some heavy-used XMLSpy commands
- Add some more buttons that use the COM automation interface of XMLSpy
- Use event handlers to update command buttons

This example is available in its entirety in the file XMLSpyActiveX_ApplicationLevel.htm within
the `<ApplicationFolder>\Examples\ActiveX\HTML\` folder of your XMLSpy installation.

**Instantiate the XMLSpyControl**

The HTML `OBJECT` tag is used to create an instance of the XMLSpyControl. The `Classid` is that of XMLSpyControl. 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.

```
<object id="objXMLSpyX" classid="clsid:a258bba2-3835-4c16-8590-72b44f52c471"
    width="0" height="0" VIEWASTEXT>
    <param name="IntegrationLevel" value="1">
</object>
```

**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.

```
<object id="objDoc1" classid="clsid:52A552E6-2A88-4e3e-B545-BE998233DDA0"
    width="600" height="500" VIEWASTEXT>
    <param name="NewDocument">
</object>
```

**Create Placeholder for Helper Windows**

The XMLSpyControlPlaceHolder 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 Output window by default (PlaceholderWindowID is 3).

```
<!-- create helper window placeholder control.           -->
<!-- the editor with focus will automatically direct its -->
<!-- output to the appropriate helper window.            -->
<object id="objOutputWindow" classid="clsid:135DEEF4-6DF0-47c2-8F8C-F145F5F3F672"
    width="200" height="200" VIEWASTEXT>
    <param name="PlaceholderWindowID" value="3">
</object>
```

The HTML page also includes an example object that can host any entry helper window:

```
<!-- create helper window placeholder object.          -->
<!-- the editor with focus will automatically direct its -->
<!-- output to the appropriate helper window.           -->
<object id="objEHWindow" classid="clsid:135DEEF4-6DF0-47c2-8F8C-F145F5F3F672"
    width="200" height="200" VIEWASTEXT>
    <param name="PlaceholderWindowID" value="0">
</object>
```
1405

ActiveX Integration

ActiveX Integration Examples

It is possible to repopulate the object above with a different helper window by clicking the following
buttons:
<!-- create buttons to switch between helper windows shown -->
<!-- in this placeholde window.
-->
<tr border="0">
<td align="center">
<input type="button" value="EH - Elements" onclick="BtnHelperWindow(0)">
</td>
</tr>
<tr align="center">
<td align="center">
<input type="button" value="EH - Attributes"
onclick="BtnHelperWindow(1)">
</td>
</tr>
<tr align="center">
<td align="center">
<input type="button" value="EH - Entities" onclick="BtnHelperWindow(2)">
</td>
</tr>

When clicked, these buttons call the BtnHelperWindow method, which reassigns the
PlaceholderWindowID of the control, and thus causes 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
XMLSpyControlPlaceholderWindow.
Create a Custom Toolbar
The example HTML page also includes a custom toolbar (intended as a replica of the XMLSpy
menu). The custom toolbar consists of buttons with images of XMLSpy commands, for example:
<button id="btnWellFormed" title="Check Well-formedness"
onclick="BtnDoCommand(34049)">
<img src="..\Images\IDC_CHECK_WELL_FORM.gif" width="16" height="16" />
</button>
<button id="btnValidate" title="Validate" onclick="BtnDoCommand(34174)">
<img src="..\Images\IDC_VALIDATE.gif" width="16" height="16" />
</button>

The names of button images correspond to the command ID numbers, see Command Reference.

Altova XMLSpy 2019 Professional Edition

© 2019 Altova Gmb H


On clicking the button, the corresponding command ID is sent to the main control and executed:

```javascript
// perform any command specified by cmdID.
// command routing includes application, active document and view.
function BtnDoCommand(cmdID)
{
    objXMLSpyX.Exec(cmdID);
    msgtext.innerText = "Command " + cmdID + " performed."
}
```

**Create More Buttons**

In the example, we add some more buttons to show some automation code.

```
<!-- add some buttons associated with above editor. -->
<!-- generation of code is now implemented using the XMLSpy automation -->
<!-- interface to select a target folder without prompting the user. -->
<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 OrgChart.pxf"
        onclick="BtnOpenFile(objDoc1, 'OrgChart.pxf')"
</p>
```

The corresponding JavaScript looks like this:

```javascript
// open a new empty document in the specified document control window.
function BtnNewFile(objDocCtrl)
{
    objDocCtrl.Open(""");
    objDocCtrl.setActive();
}

// Saves the current file in the specified document control window.
function BtnSaveFile(objDocCtrl)
{
    if(objDocCtrl.Path.length > 0)
        objDocCtrl.Save();
    else
    {
        if(strPath.value.length > 0)
            { objDocCtrl.Path = strPath.value;
                objDocCtrl.Save();
            }
        else
            { alert("Please set path for the document first!");
```
strPath.focus();
}
}

objDocCtrl setActive();
}

// ---------------------------------------------------------
// open a document in the specified document control window.
function BtnOpenFile(objDocCtrl, strFileName)
{
    // do not use XMLSpyX.Application.OpenDocument(...) to open a document,
    // since then XMLSpyControl wouldn't know a control window to show
    // the document in. Instead:

    var strPath = MakeAbsolutePath(strFileName);
    DoOpenFile(objDocCtrl, strPath);
}

Create Event Handler to Update Button Status

Availability of a command may vary with every mouse click or keystroke. The custom event
OnUpdateCmdUI of XMLSpyControl gives us an opportunity to update the enabled/disabled state
of buttons associated with XMLSpy commands. The method XMLSpyControl.QueryStatus
is used to query whether a command is enabled or not.

<!-- custom event 'OnUpdateCmdUI" of XMLSpyControl object -->
function objXMLSpyX::OnUpdateCmdUI()
{
    if ( document.readyState == "complete" )// 'complete'
    {
        // update status of buttons
        btnWellFormed.disabled = ! (objDoc1.QueryStatus(34049) & 0x02);   // not enabled
        btnValidate.disabled = ! (objDoc1.QueryStatus(34174) & 0x02);   // not enabled
    }
}

4.5.3 Java

XMLSpy 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 XMLSpy installation, after you have installed both XMLSpy and the XMLSpy
Integration Package (see also Prerequisites).

- AltovaAutomation.dll: a JNI wrapper for Altova automation servers (in case of the 32-
  bit installation of XMLSpy)
- AltovaAutomation_x64.dll: a JNI wrapper for Altova automation servers (in case of the
  64-bit installation of XMLSpy)
- AltovaAutomation.jar: Java classes to access Altova automation servers
- XMLSpyActiveX.jar: Java classes that wrap the XMLSpy ActiveX interface
• **XMLSpyActiveX_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 XMLSpy 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 `XMLSpyControl`, 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:
  
  - XMLSpyControl: Java class to access the application
  - XMLSpyControlEvents: Events interface for the XMLSpyControl
  - XMLSpyControlEventsDefaultHandler: Default handler for XMLSpyControlEvents

**Exceptions to mapping rules**

There are some exceptions to the rules listed above. These are listed below:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Java name</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMLSpyControlDocument, method New</td>
<td>newDocument</td>
</tr>
<tr>
<td>Document, method SetEncoding</td>
<td>setFileEncoding</td>
</tr>
<tr>
<td>AuthentictView, method Goto</td>
<td>gotoElement</td>
</tr>
</tbody>
</table>
This section
This section shows how some basic XMLSpy ActiveX functionality can be accessed from Java code. It is organized into the following sub-sections:

- Example Java Project
- Creating the ActiveX Controls
- Loading Data in the Controls
- Basic Event Handling
- Menus
- UI Update Event Handling
- Creating an XML Tree

Example Java Project
The XMLSpy installation package contains an example Java project, located in the ActiveX Examples folder of the application folder: `<ApplicationFolder>\Examples\ActiveX\Java`.

The Java example shows how to integrate the XMLSpyControl 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.</td>
</tr>
<tr>
<td></td>
<td>Expects folder where Java Virtual Machine resides as parameter.</td>
</tr>
<tr>
<td>XMLSpyActiveX.jar</td>
<td>Java classes of the XMLSpy ActiveX control</td>
</tr>
<tr>
<td>XMLSpyActiveX_JavaDoc.zip</td>
<td>Javadoc file containing help documentation for the Java API</td>
</tr>
<tr>
<td>XMLSpyContainer.java</td>
<td>Java example source code</td>
</tr>
<tr>
<td>XMLSpyContainerEventHandler.java</td>
<td>Java example source code</td>
</tr>
</tbody>
</table>
What the example does
The example places one XMLSpy document editor window, the XMLSpy project window, the XMLSpy XPath window and an XMLSpy entry helper in an AWT frame window. It reads out the File menu defined for XMLSpy 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 XMLSpy, which is registered as an automation server, or activates XMLSpy if it is already running.
- **Loading Data in the Controls**: Locates one of the example documents installed with XMLSpy 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 XMLSpy events.
- **Creating an XML Tree**: Shows how to create an XML tree and prepare it for modal activation.

Updating the path to the Examples folder
Before running the provided sample, you may need to edit the `XMLSpyContainer.java` file. Namely, check that the following path refers to the actual folder where the XMLSpy example files are stored on your operating system:

```java
final String strExamplesFolder = System.getenv( "USERPROFILE" ) + "\Documents \Altova\XMLSpy2019\XMLSpyExamples\";
```

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 `XMLSpyContainer.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.

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
/**
 * XMLSpy manager control - always needed
 */
public static XMLSpyControl xmlSpyControl = null;

/**
 * XMLSpy document editing control
 */
public static XMLSpyControlDocument xmlSpyDocument = null;

/**
 * Tool windows - XMLSpy place-holder controls
 */
private static XMLSpyControlPlaceHolder xmlSpyProjectToolWindow = null;
private static XMLSpyControlPlaceHolder xmlSpyXpathToolWindow = null;
private static XMLSpyControlPlaceHolder xmlSpyEHAttributeToolWindow = null;

// Create the XMLSpy 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.
xmlSpyControl = new XMLSpyControl(
    ICACTiveXIntegrationLevel.ICActiveXIntegrationOnDocumentLevel.getValue()
);
xmlSpyDocument = new XMLSpyControlDocument();
xmlSpyDocument.setPreferredSize(new Dimension ( 640, 480 ));
xmlSpyProjectToolWindow = new XMLSpyControlPlaceHolder(
```
ActiveX Integration Examples

Loading Data in the Controls
The code listing below show how data can be loaded in the ActiveX controls.

```java
// Locate samples installed with the product.
final String strExamplesFolder = System.getenv( "USERPROFILE" ) + "\Documents\Altova\XMLSpy2019\Examples\";
xmlSpyProjectToolWindow.openProject( strExamplesFolder + "Examples.spp" );
```

Basic Event Handling
The code listing below shows how basic events can be handled. When calling the XMLSpyControl's `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 XMLSpyDocumentControl's `open` method.

```java
// Open the PXF file when button is pressed
btnOpenPxf.addActionListener( new ActionListener() { 
    public void actionPerformed(ActionEvent e) { 
        try {
            xmlSpyControl.open( strExamplesFolder + "OrgChart.pxf" );
        } catch (AutomationException e1) {
            e1.printStackTrace();
        }
    }
});
```

```java
public void onOpenedOrFocused( String i_strFileName, boolean i_bOpenWithThisControl, boolean i_bFileAlreadyOpened ) throws AutomationException {
    // Handle the New/Open events coming from the Project tree or from the menus
    if ( !i_bFileAlreadyOpened ) {
        // This is basically an SDI interface, so open the file in the already existing document control
        try {
            // Code to handle the event
        }
    }
```
Menus

The code listing below shows how menu items can be created. Each XMLSpyCommand 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 XMLSpyControl 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
            XMLSpyCommand xmlSpyMenu =
                xmlSpyControl.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 XMLSpyCommands object */
public void fillMenu(Menu newMenu, XMLSpyCommands xmlSpyMenu) throws AutomationException
{
    // For each command/submenu in the xmlSpyMenu
    for (int i = 0; i < xmlSpyMenu.getCount(); ++i )
    {
        XMLSpyCommand xmlSpyCommand = xmlSpyMenu.getItem(i);
        if (xmlSpyCommand.getIsSeparator())
            newMenu.addSeparator();
        else
        {
            XMLSpy Commands subCommands = xmlSpyCommand.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( xmlSpyCommand.getLabel().replace( "&", "" ) );
mi.setActionCommand( "" + xmlSpyCommand.getID() );
mi.addActionListener( this );
newMenu.add( mi );
menuMap.put( xmlSpyCommand.getID(), mi );

else
{
    // Submenu -> create submenu and repeat recursively
    Menu newSubMenu = new Menu();
    fillMenu( newSubMenu, subCommands );
    newSubMenu.setLabel( xmlSpyCommand.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 XMLSpy
 */
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
                XMLSpyContainer.initXmlSpyDocument();
                break;
            default:
                XMLSpyContainer.xmlSpyControl.exec( iCmd );
                break;
        }
    }
    catch ( Exception ex )
    {
        ex.printStackTrace();
    }
}

UI Update Event Handling
The code listing below shows how a UI-Update event handler can be created.

/**
 * Call-back from the XMLSpyControl.
 * Called to enable/disable commands
 */
```java
@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( XMLSpyContainer.xmlSpyControl.queryStatus( pair.getKey() ) > 2 );

/**
 * Call-back from the XMLSpyControl.
 * Usually called while enabling/disabling commands due to UI updates
 */
@Override
public boolean onIsActiveEditor( String i_strFilePath ) throws AutomationException
{
    try {
        return XMLSpyContainer.xmlSpyDocument.getDocument().getFullName().equalsIgnoreCase( i_strFilePath );
    } catch ( Exception e ) {
        return false;
    }
}
}
```

Creating an XML Tree

The listing below loads an XML data object as nodes in a tree.

```java
// access required XMLSpy Java-COM classes
import com.altova.automation.XMLSpy.XMLData;

// access AWT and Swing components
import java.awt.*;
import javax.swing.*;
import javax.swing.tree.*;

/**
 * A simple example of a tree control loading the structure from an XMLData
 * object.
 * The class receives an XMLData object, loads its nodes in a JTree, and
 * prepares
 * for modal activation.
 */
public class XMLTreeDialog extends JDialog
{
    /**
     * The tree control
     */
    private JTree myTree;
```
/**
 * Root node of the tree control
 */
private DefaultMutableTreeNode top;

/**
 * Constructor that prepares the modal dialog containing the filled tree control
 * @param xml The data to be displayed in the tree
 * @param parent Parent frame
 */
public XMLTreeDialog( XMLData xml, Frame parent )
{
    // Construct the modal dialog
    super( parent, "XML tree", true );
    // Arrange controls in the dialog
    top = new DefaultMutableTreeNode("root");
    myTree = new JTree(top);
    getContentPane().add( new JScrollPane( myTree ) );
    // Build up the tree
    fillTree( top, xml );
    myTree.expandRow( 0 );
}

/**
 * Loads the nodes of an XML element under a given tree node
 * @param node Target tree node
 * @param elem Source XML element
 */
private void fillTree( DefaultMutableTreeNode node, XMLData elem)
{
    try
    {
        // There are several ways to iterate through child elements: either using the getFirstChild/getNextChild,
        // or by incrementing an index up to countChildren and calling getChild [as shown below].
        // If you only want to get children of one kind, you should use countChildrenKind/getChildKind,
        // or provide a kind to the getFirstChild before iterating with the getNextChild.
        int nSize = elem.countChildren();
        for ( int i = 0 ; i < nSize ; ++i )
        {
            // Create a new tree node for each child element, and continue recursively
            XMLData newElem = elem.getChild(i) ;
            DefaultMutableTreeNode newNode = new DefaultMutableTreeNode( newElem.getName() ) ;
            node.add( newNode ) ;
            fillTree( newNode, newElem ) ;
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
4.6 Command Reference

This section lists the names and identifiers of all menu commands that are available within XMLSpy. Every sub-section lists the commands from the corresponding top-level menu of XMLSpy. The command tables are organized as follows:

- The "Menu Item" column shows the command's menu text as it appears in XMLSpy, 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 ActiveX\Images folder of the XMLSpy 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 XMLSpyControl.Exec or the XMLSpyControlDocument.Exec methods. To query the status of a command, use the XMLSpyControl.QueryStatus or XMLSpyControlDocument.QueryStatus methods.

Depending on the edition of XMLSpy you have installed, some of these commands might not be supported.

4.6.1 "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>Reload</td>
<td>IDC_FILE_RELOAD</td>
<td>34065</td>
</tr>
<tr>
<td>Encoding…</td>
<td>IDC_ENCODING</td>
<td>34061</td>
</tr>
<tr>
<td>Close</td>
<td>ID_FILE_CLOSE</td>
<td>57602</td>
</tr>
<tr>
<td>Close All</td>
<td>IDC_CLOSE_ALL</td>
<td>34050</td>
</tr>
<tr>
<td>Close All But Active</td>
<td>IDC_CLOSE_OTHERS</td>
<td>34271</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_SAVE_ALL</td>
<td>34208</td>
</tr>
<tr>
<td>Send by Mail…</td>
<td>ID_FILE_SEND_MAIL</td>
<td>57612</td>
</tr>
<tr>
<td>Print…</td>
<td>ID_FILE_PRINT</td>
<td>57607</td>
</tr>
<tr>
<td>Print Preview</td>
<td>IDC_PRINT_PREVIEW</td>
<td>34104</td>
</tr>
<tr>
<td>Print Setup…</td>
<td>ID_FILE_PRINT_SETUP</td>
<td>57606</td>
</tr>
<tr>
<td>Recent File</td>
<td>ID_FILE_MRU_FILE1</td>
<td>57616</td>
</tr>
</tbody>
</table>
### 4.6.2 "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>Exit</td>
<td>ID_APP_EXIT</td>
<td>57665</td>
</tr>
<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>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>Copy as XML Text</td>
<td>IDC_COPY_AS_XML_TEXT</td>
<td>33443</td>
</tr>
<tr>
<td>Copy as Structured Text</td>
<td>IDC_COPY_AS_STRUCTURED_TEXT</td>
<td>33442</td>
</tr>
<tr>
<td>Copy XPath</td>
<td>IDC_COPY_XPATH</td>
<td>33444</td>
</tr>
<tr>
<td>Copy XPath</td>
<td>IDC_COPY_XPATH</td>
<td>33445</td>
</tr>
<tr>
<td>File Path...</td>
<td>IDC_EDIT_INSERT_PATH_STRING</td>
<td>34013</td>
</tr>
<tr>
<td>XInclude...</td>
<td>IDC_EDIT_INSERT_XINCLUDE_STRING</td>
<td>34017</td>
</tr>
<tr>
<td>Encoded External File...</td>
<td>IDC_EDIT_INSERT_ENCODED_BINARY_STRING</td>
<td>34273</td>
</tr>
<tr>
<td>Pretty-Print</td>
<td>IDCPRETTY_PRINT</td>
<td>34101</td>
</tr>
<tr>
<td>Strip Whitespaces</td>
<td>IDC_STRIP_WHITESPACES</td>
<td>34296</td>
</tr>
<tr>
<td>Select All</td>
<td>ID_EDIT_SELECT_ALL</td>
<td>57642</td>
</tr>
<tr>
<td>Find...</td>
<td>ID_EDIT_FIND</td>
<td>57636</td>
</tr>
<tr>
<td>Find Next</td>
<td>ID_EDIT_REPEAT</td>
<td>57640</td>
</tr>
<tr>
<td>Replace...</td>
<td>ID_EDIT_REPLACE</td>
<td>57641</td>
</tr>
<tr>
<td>Find in Files...</td>
<td>IDC_FIND_IN_FILES</td>
<td>34000</td>
</tr>
<tr>
<td>Insert/Remove Bookmark</td>
<td>IDC_TOGGLE_BOOKMARK</td>
<td>34162</td>
</tr>
<tr>
<td>Remove All Bookmarks</td>
<td>IDC_REMOVEALLBOOKMARKS</td>
<td>34132</td>
</tr>
<tr>
<td>Go to Next Bookmark</td>
<td>IDC_GOTONEXTBOOKMARK</td>
<td>34070</td>
</tr>
<tr>
<td>Go to Previous Bookmark</td>
<td>IDC_GOTOPREVBOOKMARK</td>
<td>34071</td>
</tr>
</tbody>
</table>
### "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>Comment In/Out</td>
<td>IDC_TOGGLE_XML_COMMENT</td>
<td>34029</td>
</tr>
<tr>
<td>New Project</td>
<td>IDC_ICPROJECTGUI_NEW</td>
<td>37200</td>
</tr>
<tr>
<td>Open Project...</td>
<td>IDC_ICPROJECTGUI_OPEN</td>
<td>37201</td>
</tr>
<tr>
<td>Reload Project</td>
<td>IDC_ICPROJECTGUI_RELOAD</td>
<td>37202</td>
</tr>
<tr>
<td>Close Project</td>
<td>IDC_ICPROJECTGUI_CLOSE</td>
<td>37203</td>
</tr>
<tr>
<td>Save Project</td>
<td>IDC_ICPROJECTGUI_SAVE</td>
<td>37204</td>
</tr>
<tr>
<td>Save Project As...</td>
<td>IDC_ICPROJECTGUI_SAVE_AS</td>
<td>37207</td>
</tr>
<tr>
<td>Enable Source Control</td>
<td>ID_SCC_ENABLE</td>
<td>38602</td>
</tr>
<tr>
<td>Add Files to Project...</td>
<td>IDC_ICPROJECTGUI_ADD_FILES_TO_PROJECT</td>
<td>37205</td>
</tr>
<tr>
<td>Add Global Resource to Project...</td>
<td>IDC_ICPROJECTGUI_ADD_GLOBAL_RESOURCE_TO_PROJECT</td>
<td>37239</td>
</tr>
<tr>
<td>Add URL to Project...</td>
<td>IDC_ICPROJECTGUI_ADD_URL_TO_PROJECT</td>
<td>37206</td>
</tr>
<tr>
<td>Add Active File to Project</td>
<td>IDC_ICPROJECTGUI_ADD_ACTIVE_FILE_TO_PROJECT</td>
<td>37208</td>
</tr>
<tr>
<td>Add Active and Related Files to Project</td>
<td>IDC_ICPROJECTGUI_ADD_ACTIVE_AND_RELATED_FILES_TO_PROJECT</td>
<td>37209</td>
</tr>
<tr>
<td>Add Project Folder to Project...</td>
<td>IDC_ICPROJECTGUI_ADD_FOLDER_TO_PROJECT</td>
<td>37210</td>
</tr>
<tr>
<td>Add External Folder to Project...</td>
<td>IDC_ICPROJECTGUI_ADD_EXT_FOLDER_TO_PROJECT</td>
<td>37211</td>
</tr>
<tr>
<td>Add External Web Folder to Project</td>
<td>IDC_ICPROJECTGUI_ADD_EXT_URL_FOLDER_TO_PROJECT</td>
<td>37212</td>
</tr>
<tr>
<td>Script settings...</td>
<td>IDC_PROJECT_SCRIPT_SETTINGS</td>
<td>34136</td>
</tr>
<tr>
<td>Properties...</td>
<td>IDC_ICPROJECTGUI_PROJECT_PROPERTIES</td>
<td>37223</td>
</tr>
<tr>
<td>Recent Project</td>
<td>IDC_ICPROJECTGUI_RECENT</td>
<td>37224</td>
</tr>
</tbody>
</table>
4.6.4 "XML" Menu

The "XML" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>IDC_INSERT_ATTRIBUTE</td>
<td>33449</td>
</tr>
<tr>
<td>Element</td>
<td>IDC_INSERT_STRUCT</td>
<td>33459</td>
</tr>
<tr>
<td>Text</td>
<td>IDC_INSERT_TEXT</td>
<td>33460</td>
</tr>
<tr>
<td>CDATA</td>
<td>IDC_INSERT_CDATA</td>
<td>33450</td>
</tr>
<tr>
<td>Comment</td>
<td>IDC_INSERT_COMMENT</td>
<td>33451</td>
</tr>
<tr>
<td>XML</td>
<td>IDC_INSERT_XML</td>
<td>33461</td>
</tr>
<tr>
<td>Processing Instruction</td>
<td>IDC_INSERT_PI</td>
<td>33452</td>
</tr>
<tr>
<td>XInclude...</td>
<td>IDC_INSERT_XINCLUDE</td>
<td>34019</td>
</tr>
<tr>
<td>DOCTYPE</td>
<td>IDC_INSERT_DEF_DOCTYPE</td>
<td>33453</td>
</tr>
<tr>
<td>ExternalID</td>
<td>IDC_INSERT_DEF_EXTERNAL_ID</td>
<td>33456</td>
</tr>
<tr>
<td>ELEMENT</td>
<td>IDC_INSERT_DEF_ELEMENT</td>
<td>33454</td>
</tr>
<tr>
<td>ATTLIST</td>
<td>IDC_INSERT_DEF_ATTLIST</td>
<td>33452</td>
</tr>
<tr>
<td>ENTITY</td>
<td>IDC_INSERT_DEF_ENTITY</td>
<td>33455</td>
</tr>
<tr>
<td>NOTATION</td>
<td>IDC_INSERT_DEF_NOTATION</td>
<td>33457</td>
</tr>
<tr>
<td>Encoded External File...</td>
<td>IDC_INSERT_ENCODED_BINARY</td>
<td>34274</td>
</tr>
<tr>
<td>Attribute</td>
<td>IDC_APPEND_ATTRIBUTE</td>
<td>33415</td>
</tr>
<tr>
<td>Element</td>
<td>IDC_APPEND_STRUCT</td>
<td>33425</td>
</tr>
<tr>
<td>Text</td>
<td>IDC_APPEND_TEXT</td>
<td>33426</td>
</tr>
<tr>
<td>CDATA</td>
<td>IDC_APPEND_CDATA</td>
<td>33416</td>
</tr>
<tr>
<td>Comment</td>
<td>IDC_APPEND_COMMENT</td>
<td>33417</td>
</tr>
<tr>
<td>XML</td>
<td>IDC_APPEND_XML</td>
<td>33427</td>
</tr>
<tr>
<td>Processing Instruction</td>
<td>IDC_APPEND_PI</td>
<td>33424</td>
</tr>
<tr>
<td>XInclude...</td>
<td>IDC_APPEND_XINCLUDE</td>
<td>34026</td>
</tr>
<tr>
<td>DOCTYPE</td>
<td>IDC_APPEND_DEF_DOCTYPE</td>
<td>33419</td>
</tr>
<tr>
<td>ExternalID</td>
<td>IDC_APPEND_DEF_EXTERNAL_ID</td>
<td>33422</td>
</tr>
<tr>
<td>ELEMENT</td>
<td>IDC_APPEND_DEF_ELEMENT</td>
<td>33420</td>
</tr>
<tr>
<td>ATTLIST</td>
<td>IDC_APPEND_DEF_ATTLIST</td>
<td>33418</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>ENTITY</td>
<td>IDC_APPEND_DEF_ENTITY</td>
<td>33421</td>
</tr>
<tr>
<td>NOTATION</td>
<td>IDC_APPEND_DEF_NOTATION</td>
<td>33423</td>
</tr>
<tr>
<td>Encoded External File...</td>
<td>IDC_APPEND_ENCODED_BINARY</td>
<td>34276</td>
</tr>
<tr>
<td>Attribute</td>
<td>IDC_ADD_CHILD_ATTRIBUTE</td>
<td>33402</td>
</tr>
<tr>
<td>Element</td>
<td>IDC_ADD_CHILD_STRUCT</td>
<td>33412</td>
</tr>
<tr>
<td>Text</td>
<td>IDC_ADD_CHILD_TEXT</td>
<td>33413</td>
</tr>
<tr>
<td>CDATA</td>
<td>IDC_ADD_CHILD_CDATA</td>
<td>33403</td>
</tr>
<tr>
<td>Comment</td>
<td>IDC_ADD_CHILD_COMMENT</td>
<td>33404</td>
</tr>
<tr>
<td>XML</td>
<td>IDC_ADD_CHILD_XML</td>
<td>33414</td>
</tr>
<tr>
<td>Processing Instruction</td>
<td>IDC_ADD_CHILD_PI</td>
<td>33411</td>
</tr>
<tr>
<td>XInclude...</td>
<td>IDC_ADD_CHILD_XINCLUDE</td>
<td>34027</td>
</tr>
<tr>
<td>DOCTYPE</td>
<td>IDC_ADD_CHILD_DEF_DOCTYPE</td>
<td>33406</td>
</tr>
<tr>
<td>ExternalID</td>
<td>IDC_ADD_CHILD_DEF_EXTERNAL_ID</td>
<td>33409</td>
</tr>
<tr>
<td>ELEMENT</td>
<td>IDC_ADD_CHILD_DEF_ELEMENT</td>
<td>33407</td>
</tr>
<tr>
<td>ATTLIST</td>
<td>IDC_ADD_CHILD_DEF_ATTLIST</td>
<td>33405</td>
</tr>
<tr>
<td>ENTITY</td>
<td>IDC_ADD_CHILD_DEF_ENTITY</td>
<td>33408</td>
</tr>
<tr>
<td>NOTATION</td>
<td>IDC_ADD_CHILD_DEF_NOTATION</td>
<td>33410</td>
</tr>
<tr>
<td>Encoded External File...</td>
<td>IDC_ADD_CHILD_ENCODED_BINARY</td>
<td>34277</td>
</tr>
<tr>
<td>Attribute</td>
<td>IDC_CONVERT_TO_ATTRIBUTE</td>
<td>33429</td>
</tr>
<tr>
<td>Element</td>
<td>IDC_CONVERT_TO_STRUCT</td>
<td>33439</td>
</tr>
<tr>
<td>Text</td>
<td>IDC_CONVERT_TO_TEXT</td>
<td>33440</td>
</tr>
<tr>
<td>CDATA</td>
<td>IDC_CONVERT_TO_CDATA</td>
<td>33430</td>
</tr>
<tr>
<td>Comment</td>
<td>IDC_CONVERT_TO_COMMENT</td>
<td>33431</td>
</tr>
<tr>
<td>XML</td>
<td>IDC_CONVERT_TO_XML</td>
<td>33441</td>
</tr>
<tr>
<td>Processing Instruction</td>
<td>IDC_CONVERT_TO_PI</td>
<td>33438</td>
</tr>
<tr>
<td>DOCTYPE</td>
<td>IDC_CONVERT_TO_DEF_DOCTYPE</td>
<td>33433</td>
</tr>
<tr>
<td>ExternalID</td>
<td>IDC_CONVERT_TO_DEF_EXTERNAL_ID</td>
<td>33436</td>
</tr>
<tr>
<td>ELEMENT</td>
<td>IDC_CONVERT_TO_DEF_ELEMENT</td>
<td>33434</td>
</tr>
<tr>
<td>ATTLIST</td>
<td>IDC_CONVERT_TO_DEF_ATTLIST</td>
<td>33432</td>
</tr>
</tbody>
</table>
### 4.6.5 "DTD/Schema" Menu

The "DTD/Schema" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign DTD…</td>
<td>IDC_ASSIGN_DTD</td>
<td>34032</td>
</tr>
<tr>
<td>Assign Schema…</td>
<td>IDC_ASSIGN_SCHEMA</td>
<td>34033</td>
</tr>
<tr>
<td>Include Another DTD…</td>
<td>IDC_INCLUDE_DTD</td>
<td>34084</td>
</tr>
<tr>
<td>Go to DTD</td>
<td>IDC_GOTO_DTD</td>
<td>34072</td>
</tr>
<tr>
<td>Go to Schema</td>
<td>IDC_GOTO_SCHEMA</td>
<td>34074</td>
</tr>
<tr>
<td>Go to Definition</td>
<td>IDC_GOTO_DEFINITION</td>
<td>33447</td>
</tr>
<tr>
<td>Generate DTD/Schema…</td>
<td>IDC_GENERATE_DTD_SCHEMA</td>
<td>34068</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTITY</td>
<td>IDC_CONVERT_TO_DEF_ENTITY</td>
<td>33435</td>
</tr>
<tr>
<td>NOTATION</td>
<td>IDC_CONVERT_TO_DEF_NOTATION</td>
<td>33437</td>
</tr>
<tr>
<td>Display as Table</td>
<td>IDC_GRID_VIEW_AS_TABLE</td>
<td>34075</td>
</tr>
<tr>
<td>Insert Row</td>
<td>IDC_TABLE_INSERT_ROW</td>
<td>34158</td>
</tr>
<tr>
<td>Append Row</td>
<td>IDC_TABLE_APPEND_ROW</td>
<td>34157</td>
</tr>
<tr>
<td>Ascending Sort</td>
<td>IDC_TABLE_SORT_ASC</td>
<td>33464</td>
</tr>
<tr>
<td>Descending Sort</td>
<td>IDC_TABLE_SORT_DESC</td>
<td>33465</td>
</tr>
<tr>
<td>Move Left</td>
<td>IDC_MOVE_LEFT</td>
<td>34091</td>
</tr>
<tr>
<td>Move Right</td>
<td>IDC_MOVE_RIGHT</td>
<td>34092</td>
</tr>
<tr>
<td>Enclose in Element</td>
<td>IDC_ENCLOSE_IN_ELEMENT</td>
<td>33446</td>
</tr>
<tr>
<td>Evaluate XPath…</td>
<td>IDC_EVALUATE_XPATH</td>
<td>34007</td>
</tr>
<tr>
<td>Check Well-Formedness</td>
<td>IDC_CHECK_WELL_FORM</td>
<td>34049</td>
</tr>
<tr>
<td>Validate XML</td>
<td>IDC_VALIDATE</td>
<td>32954</td>
</tr>
<tr>
<td>Validate XML on Server</td>
<td>IDC_VALIDATE_RAPTOR</td>
<td>34309</td>
</tr>
<tr>
<td>Update Entry Helpers</td>
<td>IDC_UPDATE_ELEMENT_CHOICE</td>
<td>34173</td>
</tr>
<tr>
<td>Namespace Prefix…</td>
<td>IDC_NAMESPACE</td>
<td>33462</td>
</tr>
<tr>
<td>Create XML Signature…</td>
<td>IDC_XML_SIGNATURE_CREATE</td>
<td>34280</td>
</tr>
<tr>
<td>Verify XML Signature…</td>
<td>IDC_XML_SIGNATURE_VERIFY</td>
<td>34281</td>
</tr>
</tbody>
</table>
### 4.6.6 "Schema design" Menu

The "Schema design" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema Settings...</td>
<td>IDC_SCHEMA_NAMESPACES</td>
<td>33571</td>
</tr>
<tr>
<td>Save Diagram...</td>
<td>IDC_SCHEMA_SAVE_DIAGRAM</td>
<td>33581</td>
</tr>
<tr>
<td>Generate Documentation...</td>
<td>IDC_SCHEMA_DOCUMENTATION</td>
<td>34146</td>
</tr>
<tr>
<td>Configure View...</td>
<td>IDC_SCHEMA_VIEW_CONFIG</td>
<td>33593</td>
</tr>
<tr>
<td>Zoom...</td>
<td>IDC_SCHEMA_ZOOM</td>
<td>34150</td>
</tr>
<tr>
<td>Display All Globals</td>
<td>IDC_SCHEMA_MODE_GLOBALS</td>
<td>34147</td>
</tr>
<tr>
<td>Display Diagram</td>
<td>IDC_SCHEMA_MODE_DIAGRAM</td>
<td>33570</td>
</tr>
<tr>
<td>Enable Oracle Schema Extensions</td>
<td>IDC_SCHEMA_ORACLE_EXTENSIONS</td>
<td>33577</td>
</tr>
<tr>
<td>Oracle Schema Settings...</td>
<td>IDC_SCHEMA_ORACLE_SCHEMA_SETTINGS</td>
<td>33578</td>
</tr>
<tr>
<td>Enable Microsoft SQL Server Schema Extensions</td>
<td>IDC_SCHEMA_SQLSERVER_EXTENSIONS</td>
<td>33588</td>
</tr>
<tr>
<td>Named Schema Relationships...</td>
<td>IDC_SCHEMA_SQLSERVER_GLOBAL_RELATIONSHIPS</td>
<td>33589</td>
</tr>
<tr>
<td>Unnamed Element Relationships...</td>
<td>IDC_SCHEMA_SQLSERVER_LOCAL_RELATIONSHIPS</td>
<td>33590</td>
</tr>
</tbody>
</table>
### ELATIONSHIPS

- **Connect to SchemaAgent Server...**
  - Command: IDC_SCHEMA_SCHEMAAGENT_SERVER_CONNECT
  - ID: 33582
- **Disconnect from SchemaAgent Server**
  - Command: IDC_SCHEMA_SCHEMAAGENT_SERVER_DISCONNECT
  - ID: 33583
- **File Only**
  - Command: IDC_SCHEMAAGENT_SHOW_FILE_ONLY
  - ID: 33504
- **File and All Directly Referenced Schema Files**
  - Command: IDC_SCHEMAAGENT_SHOW_WITH_DIRECTLY_REFERENCED_SCHEMAS
  - ID: 33608
- **File and All Directly Referencing Schema Files**
  - Command: IDC_SCHEMAAGENT_SHOW_WITH_DIRECTLY_REFERENCING_SCHEMAS
  - ID: 33602
- **File and All Directly Related Schema Files**
  - Command: IDC_SCHEMAAGENT_SHOW_WITH_DIRECTLY_RELATED_SCHEMAS
  - ID: 33613
- **SchemaAgent Validation...**
  - Command: IDC_SCHEMA_EXTVALID_MENU
  - ID: 33539
- **Create Schema Subset...**
  - Command: IDC_SCHEMA_CREATE_SUBSET
  - ID: 33650
- **Flatten Schema...**
  - Command: IDC_SCHEMA_FLATTEN
  - ID: 33651

### "XSL/XQuery" Menu

The "XSL/XQuery" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSL Transformation</td>
<td>IDC_TRANSFORM_XSL</td>
<td>33006</td>
</tr>
<tr>
<td>XSL Speed Optimizer</td>
<td>IDC_TRANSFORM_XSLPBO</td>
<td>34306</td>
</tr>
<tr>
<td>XSL-FO Transformation</td>
<td>IDC_TRANSFORM_XSLFO</td>
<td>33007</td>
</tr>
<tr>
<td>XSL Parameters / XQuery Variables...</td>
<td>IDC_TRANSFORM_XSL_PARAMS</td>
<td>33008</td>
</tr>
<tr>
<td>XQuery/Update Execution</td>
<td>IDC_TRANSFORM_XQUERY</td>
<td>34170</td>
</tr>
<tr>
<td>Enable Back Mapping</td>
<td>IDC_ENABLE_BACKMAPPING</td>
<td>34364</td>
</tr>
<tr>
<td>Enable XSLT/ XQuery Profiling...</td>
<td>IDC_PROFILING_OPTIONS</td>
<td>34105</td>
</tr>
<tr>
<td>Assign XSL...</td>
<td>IDC_ASSIGN_XSL</td>
<td>33001</td>
</tr>
<tr>
<td>Assign XSL-FO...</td>
<td>IDC_ASSIGN_XSLFO</td>
<td>33002</td>
</tr>
<tr>
<td>Assign Sample XML File...</td>
<td>IDC_ASSIGN_SAMPLE_XML</td>
<td>33000</td>
</tr>
<tr>
<td>Go to XSL</td>
<td>IDC_GOTO_XSL</td>
<td>33004</td>
</tr>
<tr>
<td>Start Debugger / Go</td>
<td>ID_PROCESS_XSL</td>
<td>34212</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>Stop Debugger</td>
<td>ID_XSLT_DEBUGGER_STOP</td>
<td>33017</td>
</tr>
<tr>
<td>Restart Debugger</td>
<td>ID_XSLT_DEBUGGER_RESTART</td>
<td>33013</td>
</tr>
<tr>
<td>End Debugger Session</td>
<td>ID_XSLT_DEBUGGER_END_SESSION</td>
<td>33011</td>
</tr>
<tr>
<td>Step Into</td>
<td>ID_XSLT_DEBUGGER_STEP</td>
<td>33014</td>
</tr>
<tr>
<td>Step Out</td>
<td>ID_XSLT_DEBUGGER_STEP_OUT</td>
<td>33015</td>
</tr>
<tr>
<td>Step Over</td>
<td>ID_XSLT_DEBUGGER_STEP_OVER</td>
<td>33016</td>
</tr>
<tr>
<td>Show Current Execution Node</td>
<td>ID_XSLT_DEBUGGER_GO_TO_CURRENT_EXECUTION_NODES</td>
<td>33012</td>
</tr>
<tr>
<td>Insert/Remove Breakpoint</td>
<td>IDC_TOGGLE_BREAKPOINT</td>
<td>34246</td>
</tr>
<tr>
<td>Insert/Remove Tracepoint</td>
<td>IDC_TOGGLE_TRACEPOINT</td>
<td>34248</td>
</tr>
<tr>
<td>Enable/Disable Breakpoint</td>
<td>IDC_ENABLE_BREAKPOINT</td>
<td>34245</td>
</tr>
<tr>
<td>Enable/Disable Tracepoint</td>
<td>IDC_ENABLE_TRACEPOINT</td>
<td>34247</td>
</tr>
<tr>
<td>Breakpoints/Tracepoints…</td>
<td>ID_XSLTDEBUGGER_BREAKPOINTS</td>
<td>33009</td>
</tr>
<tr>
<td>Call Stack</td>
<td>ID_XSL_DEBUGWINDOWS_CALLSTACK</td>
<td>34238</td>
</tr>
<tr>
<td>XPath-Watch</td>
<td>ID_XSL_DEBUGWINDOWS_WATCH</td>
<td>34244</td>
</tr>
<tr>
<td>Context</td>
<td>ID_XSL_DEBUGWINDOWS_CONTEXT</td>
<td>34239</td>
</tr>
<tr>
<td>Variables</td>
<td>ID_XSL_DEBUGWINDOWS_VARIABLE</td>
<td>34243</td>
</tr>
<tr>
<td>Messages</td>
<td>ID_XSL_DEBUGWINDOWS_MESSAGES</td>
<td>34240</td>
</tr>
<tr>
<td>Templates</td>
<td>ID_XSL_DEBUGWINDOWSTEMPLATES</td>
<td>34241</td>
</tr>
<tr>
<td>Info</td>
<td>ID_XSLXQUERY_DEBUGWINDOWS_INFO</td>
<td>34237</td>
</tr>
<tr>
<td>Trace</td>
<td>ID_XSL_DEBUGWINDOWS_TRACES</td>
<td>34242</td>
</tr>
<tr>
<td>Debug Settings…</td>
<td>ID_XSLTDEBUGGER_SETTINGS</td>
<td>33010</td>
</tr>
</tbody>
</table>

### 4.6.8 "Authentic" Menu

The "Authentic" 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 Document…</td>
<td>IDC_AUTHENTIC_NEW_FILE</td>
<td>34036</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>Edit Database Data...</td>
<td>IDC_AUTHENTIC_EDIT_DB</td>
<td>34035</td>
</tr>
<tr>
<td>Assign a StyleVision Stylesheet...</td>
<td>IDC_ASSIGN_SPS</td>
<td>34034</td>
</tr>
<tr>
<td>Edit StyleVision Stylesheet</td>
<td>IDC_EDIT_SPS</td>
<td>34060</td>
</tr>
<tr>
<td>Select New Row with XML Data for Editing...</td>
<td>IDC_CHANGE_WORKING_DB_XMLCELL</td>
<td>32861</td>
</tr>
<tr>
<td>XML Signature...</td>
<td>IDC_AUTHENTICGUI_XMLSIGNATURE</td>
<td>32862</td>
</tr>
<tr>
<td>Define XML Entities...</td>
<td>IDC_DEFINE_ENTITIES</td>
<td>32805</td>
</tr>
<tr>
<td>Hide Markup</td>
<td>IDC_MARKUP_HIDE</td>
<td>32855</td>
</tr>
<tr>
<td>Show Small Markup</td>
<td>IDC_MARKUP_SMALL</td>
<td>32858</td>
</tr>
<tr>
<td>Show Large Markup</td>
<td>IDC_MARKUP_LARGE</td>
<td>32856</td>
</tr>
<tr>
<td>Show Mixed Markup</td>
<td>IDC_MARKUP_MIXED</td>
<td>32857</td>
</tr>
<tr>
<td>Toggle Bold</td>
<td>IDC_AUTHENTICGUI_RICHEDIT_TOGGLE_BOLD</td>
<td>32813</td>
</tr>
<tr>
<td>Toggle Italic</td>
<td>IDC_AUTHENTICGUI_RICHEDIT_TOGGLE_ITALIC</td>
<td>32814</td>
</tr>
<tr>
<td>Toggle Underline</td>
<td>IDC_AUTHENTICGUI_RICHEDIT_TOGGLE_UNDERLINE</td>
<td>32815</td>
</tr>
<tr>
<td>Toggle Strikethrough</td>
<td>IDC_AUTHENTICGUI_RICHEDIT_TOGGLE_STRIKETHROUGH</td>
<td>32816</td>
</tr>
<tr>
<td>Foreground Color</td>
<td>IDC_AUTHENTICGUI_RICHEDIT_COLOR_FOREGROUND</td>
<td>32824</td>
</tr>
<tr>
<td>Background Color</td>
<td>IDC_AUTHENTICGUI_RICHEDIT_COLOR_BACKGROUND</td>
<td>32830</td>
</tr>
<tr>
<td>Align Left</td>
<td>IDC_AUTHENTICGUI_RICHEDIT_ALIGN_LEFT</td>
<td>32818</td>
</tr>
<tr>
<td>Center</td>
<td>IDC_AUTHENTICGUI_RICHEDIT_ALIGN_CENTER</td>
<td>32819</td>
</tr>
<tr>
<td>Align Right</td>
<td>IDC_AUTHENTICGUI_RICHEDIT_ALIGN_RIGHT</td>
<td>32820</td>
</tr>
<tr>
<td>Append Row</td>
<td>IDC_ROW_APPEND</td>
<td>32806</td>
</tr>
<tr>
<td>Insert Row</td>
<td>IDC_ROW_INSERT</td>
<td>32809</td>
</tr>
<tr>
<td>Duplicate Row</td>
<td>IDC_ROW_DUPLICATE</td>
<td>32808</td>
</tr>
<tr>
<td>Move Row Up</td>
<td>IDC_ROW_MOVE_UP</td>
<td>32811</td>
</tr>
</tbody>
</table>
### 4.6.9 "DB" Menu

The "DB" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Database</td>
<td>IDC_QUERYDATABASE</td>
<td>34012</td>
</tr>
<tr>
<td>Manage XML Schemas...</td>
<td>IDC_DB_MANAGESCHEMAS</td>
<td>34014</td>
</tr>
<tr>
<td>Assign XML Schema...</td>
<td>IDC_DB_CHOOSEVALIDATIONSCHEMA</td>
<td>34016</td>
</tr>
<tr>
<td>Manage XML Schemas...</td>
<td>IDC_DB_MANAGESCHEMAS</td>
<td>34014</td>
</tr>
<tr>
<td>Manage XML Schemas...</td>
<td>IDC_DB_MANAGESCHEMAS</td>
<td>34014</td>
</tr>
<tr>
<td>Browse Oracle XML Documents...</td>
<td>ID_CONVERT_ORACLEXMLDB_BROWSE</td>
<td>34205</td>
</tr>
</tbody>
</table>

### 4.6.10 "Convert" Menu

The "Convert" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import Text File...</td>
<td>IDC_IMPORT_TEXT</td>
<td>34082</td>
</tr>
<tr>
<td>Import Database Data...</td>
<td>IDC_IMPORT_DATABASE</td>
<td>34080</td>
</tr>
<tr>
<td>Import Microsoft Word Document...</td>
<td>IDC_IMPORT_WORD</td>
<td>34083</td>
</tr>
<tr>
<td>Create XML Schema from DB Structure</td>
<td>IDC_CREATE_DB_SCHEMA</td>
<td>34054</td>
</tr>
<tr>
<td>DB Import Based on XML Schema</td>
<td>IDC_IMPORT_DB_SCHEMA</td>
<td>34081</td>
</tr>
<tr>
<td>Create DB Structure from XML Schema</td>
<td>IDC_CREATE_DB_BASED_ON_SCHEMA</td>
<td>34053</td>
</tr>
<tr>
<td>Export to Text Files...</td>
<td>IDC_EXPORT_TEXTFILE</td>
<td>34064</td>
</tr>
</tbody>
</table>
### Menu item | Command name | ID
--- | --- | ---
Export to a Database... | IDC_EXPORT_DB | 34003
Convert XML Instance to/from JSON... | IDC_JSON_CONVERT_TOFROM_XML | 34135
Convert XML Schema to/from JSON Schema... | IDC_JSON_CONVERT_TOFROM_XSD | 34350

#### 4.6.11 "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>Text View</td>
<td>IDC_VIEW_TEXT</td>
<td>34180</td>
</tr>
<tr>
<td>Enhanced Grid View</td>
<td>IDC_VIEW_GRID</td>
<td>34178</td>
</tr>
<tr>
<td>Schema Design View</td>
<td>IDC_VIEW_SCHEMA</td>
<td>34179</td>
</tr>
<tr>
<td>WSDL Design View</td>
<td>IDC_VIEW_WSDL</td>
<td>34117</td>
</tr>
<tr>
<td>XBRL Taxonomy View</td>
<td>IDC_VIEW_XBRL</td>
<td>34118</td>
</tr>
<tr>
<td>Authentic View</td>
<td>IDC_VIEW_CONTENT</td>
<td>34177</td>
</tr>
<tr>
<td>Browser View</td>
<td>IDC_VIEW_BROWSER</td>
<td>34176</td>
</tr>
<tr>
<td>Expand +</td>
<td>IDC_SEL_EXPAND</td>
<td>34152</td>
</tr>
<tr>
<td>Collapse -</td>
<td>IDC_SEL_COLLAPSE</td>
<td>34151</td>
</tr>
<tr>
<td>Expand Fully</td>
<td>IDC_SEL_EXPAND_ALL</td>
<td>33463</td>
</tr>
<tr>
<td>Collapse Unselected</td>
<td>IDC_COLLAPSE_UNSELECTED</td>
<td>33428</td>
</tr>
<tr>
<td>Optimal Widths</td>
<td>IDC_OPTIMAL_WIDTHS</td>
<td>34099</td>
</tr>
<tr>
<td>Word Wrap</td>
<td>IDC_OPTIMAL_WIDTHS</td>
<td>34181</td>
</tr>
<tr>
<td>Go to Line/Character</td>
<td>IDC_GOTO_LINE</td>
<td>34073</td>
</tr>
<tr>
<td>Go to File</td>
<td>IDC_GOTO_FILE</td>
<td>33448</td>
</tr>
<tr>
<td>Text View Settings</td>
<td>IDC_TEXTVIEW_SETTINGS</td>
<td>34119</td>
</tr>
</tbody>
</table>

#### 4.6.12 "Browser" Menu
The "Browser" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Back</td>
<td>IDC_STEP_BACK</td>
<td>32958</td>
</tr>
</tbody>
</table>
### 4.6.13 "WSDL" Menu

The "WSDL" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert Message</td>
<td>ID_WSDL_MESSAGES_ADDNEWMESSAGE</td>
<td>33715</td>
</tr>
<tr>
<td>Delete Message</td>
<td>ID_WSDL_MESSAGES_DELETESELECTEDMESSAGE</td>
<td>33717</td>
</tr>
<tr>
<td>Add Message Part (Parameter)</td>
<td>ID_WSDL_MESSAGES_ADDMESSAGEPART</td>
<td>33714</td>
</tr>
<tr>
<td>Delete Message Part (Parameter)</td>
<td>ID_WSDL_MESSAGES_DELETEMESSAGEPART</td>
<td>33716</td>
</tr>
<tr>
<td>request-response</td>
<td>IDC_WSDL_OPERATION_APPENDREQUESTRESPONSE</td>
<td>33734</td>
</tr>
<tr>
<td>solicit-response</td>
<td>IDC_WSDL_OPERATION_APPENDSOLICITRESPONSE</td>
<td>33737</td>
</tr>
<tr>
<td>one-way</td>
<td>IDC_WSDL_OPERATION_APPENDONeway</td>
<td>33735</td>
</tr>
<tr>
<td>notification</td>
<td>IDC_WSDL_OPERATION_APPENDNOTIFICATION</td>
<td>33736</td>
</tr>
<tr>
<td>Empty Operation</td>
<td>ID_WSDL_OPERATIONS_APPENDAOPERTITIONTOHISPORTTYPE</td>
<td>33722</td>
</tr>
<tr>
<td>Delete Operation</td>
<td>ID_WSDL_OPERATIONS_DELETEOPEPARTITION</td>
<td>33724</td>
</tr>
<tr>
<td>Add Input Element</td>
<td>ID_WSDL_OPERATIONS_ADDINPUTFUNCTION</td>
<td>33719</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>Add Output Element</td>
<td>ID_WSDL_OPERATIONS_ADDOUTPUTFUNCTION</td>
<td>33721</td>
</tr>
<tr>
<td>Add Fault Element</td>
<td>ID_WSDL_OPERATIONS_ADDFAULTFUNCTION</td>
<td>33718</td>
</tr>
<tr>
<td>Delete Input/Output/Fault Element</td>
<td>ID_WSDL_OPERATIONS_DELETEINPUTOUTPUTFUNCTION</td>
<td>33723</td>
</tr>
<tr>
<td>Add New Message to Input/Output/Fault Element</td>
<td>ID_WSDL_OPERATIONS_ADDNEWMESSAGESSAGETOTHISELEMENT</td>
<td>33720</td>
</tr>
<tr>
<td>Insert Port Type</td>
<td>ID_WSDL_PORTTYPE_INSERTAPORTYPE</td>
<td>33727</td>
</tr>
<tr>
<td>Delete Port Type</td>
<td>ID_WSDL_PORTTYPE_DELETETHISPORTYPE</td>
<td>33726</td>
</tr>
<tr>
<td>Insert Binding</td>
<td>ID_WSDL_BINDING_NEWBINDING</td>
<td>33713</td>
</tr>
<tr>
<td>Delete Binding</td>
<td>ID_WSDL_BINDING_DELETEBINDING</td>
<td>33711</td>
</tr>
<tr>
<td>soap:body</td>
<td>ID_WSDL_BINDING_APPENDEXTENSIBILITY_SOAPBODY</td>
<td>33706</td>
</tr>
<tr>
<td>soap:header</td>
<td>ID_WSDL_BINDING_APPENDEXTENSIBILITY_SOAPHEADER</td>
<td>33708</td>
</tr>
<tr>
<td>soap:headerfault</td>
<td>ID_WSDL_BINDING_APPENDEXTENSIBILITY_SOAPHEADERFAULT</td>
<td>33709</td>
</tr>
<tr>
<td>soap:fault</td>
<td>ID_WSDL_BINDING_APPENDEXTENSIBILITY_SOAPFAULT</td>
<td>33707</td>
</tr>
<tr>
<td>mime:content</td>
<td>ID_WSDL_BINDING_APPENDEXTENSIBILITY_MIMECONTENT</td>
<td>33702</td>
</tr>
<tr>
<td>mime:multipartrelated</td>
<td>ID_WSDL_BINDING_APPENDEXTENSIBILITY_MIMEMULTIPARTRELATED</td>
<td>33704</td>
</tr>
<tr>
<td>mime:part</td>
<td>ID_WSDL_BINDING_APPENDEXTENSIBILITY_MIMEPART</td>
<td>33705</td>
</tr>
<tr>
<td>mime:mimeXml</td>
<td>ID_WSDL_BINDING_APPENDEXTENSIBILITY_MIMEMIMEXML</td>
<td>33703</td>
</tr>
<tr>
<td>http:urlencoded</td>
<td>ID_WSDL_BINDING_APPENDEXTENSIBILITY_HTTPURLENCODED</td>
<td>33700</td>
</tr>
<tr>
<td>http:urireplacement</td>
<td>ID_WSDL_BINDING_APPENDEXTENSIBILITY_HTTPURLREPLACEMENT</td>
<td>33701</td>
</tr>
<tr>
<td>Delete Extensibility Element</td>
<td>ID_WSDL_BINDING_DELETEEXTEXTENSIBILITY</td>
<td>33712</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>Insert Service</td>
<td>ID_WSDL_SERVICE_INSERTSERVICE</td>
<td>33731</td>
</tr>
<tr>
<td>Delete Service</td>
<td>ID_WSDL_SERVICE_DELETETHISERVICE</td>
<td>33729</td>
</tr>
<tr>
<td>Insert Port</td>
<td>ID_WSDL_SERVICE_INSERTNEWPORT</td>
<td>33730</td>
</tr>
<tr>
<td>Delete Port</td>
<td>ID_WSDL_SERVICE_DELETETHISPORT</td>
<td>33728</td>
</tr>
<tr>
<td>Add New Interface</td>
<td>IDC_WSDL20_ADDINTERFACE</td>
<td>33794</td>
</tr>
<tr>
<td>Delete Interface</td>
<td>IDC_WSDL20_DELETEINTERFACE</td>
<td>33795</td>
</tr>
<tr>
<td>Add New Fault</td>
<td>IDC_WSDL20_ADDINTERFACEFAULT</td>
<td>33796</td>
</tr>
<tr>
<td>Delete Fault</td>
<td>IDC_WSDL20_DELETEINTERFACEFAULT</td>
<td>33808</td>
</tr>
<tr>
<td>In-only</td>
<td>IDC_WSDL20_ADDINTERFACEOPERATION_INONLY</td>
<td>33797</td>
</tr>
<tr>
<td>Robust-in-only</td>
<td>IDC_WSDL20_ADDINTERFACEOPERATION_ROBUSTINONLY</td>
<td>33798</td>
</tr>
<tr>
<td>In-out</td>
<td>IDC_WSDL20_ADDINTERFACEOPERATION_INOUT</td>
<td>33801</td>
</tr>
<tr>
<td>In-opt-out</td>
<td>IDC_WSDL20_ADDINTERFACEOPERATION_INOPTOUT</td>
<td>33802</td>
</tr>
<tr>
<td>Out-in</td>
<td>IDC_WSDL20_ADDINTERFACEOPERATION_OUTIN</td>
<td>33803</td>
</tr>
<tr>
<td>Out-opt-in</td>
<td>IDC_WSDL20_ADDINTERFACEOPERATION_OUTOPTIN</td>
<td>33804</td>
</tr>
<tr>
<td>Out-only</td>
<td>IDC_WSDL20_ADDINTERFACEOPERATION_OUTONLY</td>
<td>33800</td>
</tr>
<tr>
<td>Robust-out-only</td>
<td>IDC_WSDL20_ADDINTERFACEOPERATION_ROBUSTOUTONLY</td>
<td>33799</td>
</tr>
<tr>
<td>Empty Operation</td>
<td>IDC_WSDL20_ADDINTERFACEOPERATION_EMPTY</td>
<td>33805</td>
</tr>
<tr>
<td>Delete Operation</td>
<td>IDC_WSDL20_DELETEINTERFACEOPERATION</td>
<td>33809</td>
</tr>
<tr>
<td>Add New Binding</td>
<td>IDC_WSDL20_ADDBINDING</td>
<td>33820</td>
</tr>
<tr>
<td>Delete Binding</td>
<td>IDC_WSDL20_DELETEBINDING</td>
<td>33821</td>
</tr>
<tr>
<td>Add New Fault</td>
<td>IDC_WSDL20_ADDBINDINGFAULT</td>
<td>33822</td>
</tr>
<tr>
<td>Delete Fault</td>
<td>IDC_WSDL20_DELETEBINDINGFAULT</td>
<td>33826</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>Add New Operation</td>
<td>IDC_WSDL20_ADDBINDINGOPERATION</td>
<td>33823</td>
</tr>
<tr>
<td>Delete Operation</td>
<td>IDC_WSDL20_DELETEBINDINGOPERATION</td>
<td>33827</td>
</tr>
<tr>
<td>Add New Service</td>
<td>IDC_WSDL20_ADDSERVICE</td>
<td>33839</td>
</tr>
<tr>
<td>Delete Service</td>
<td>IDC_WSDL20_DELETESERVICE</td>
<td>33840</td>
</tr>
<tr>
<td>Add New Endpoint</td>
<td>IDC_WSDL20_ADDENDPOINT</td>
<td>33841</td>
</tr>
<tr>
<td>Delete Endpoint</td>
<td>IDC_WSDL20_DELETEENDPOINT</td>
<td>33842</td>
</tr>
<tr>
<td>New Schema</td>
<td>ID_WSDL_TYPES_NEWSCHEMA</td>
<td>33733</td>
</tr>
<tr>
<td>Embed Schema</td>
<td>ID_WSDL_TYPES_EMBEDSCHEMA</td>
<td>39456</td>
</tr>
<tr>
<td>Extract Schema(s)</td>
<td>ID_WSDL_TYPES_EXTRACTSCHEMAS</td>
<td>39459</td>
</tr>
<tr>
<td>Edit Schema(s) in Schema View</td>
<td>ID_WSDL_TYPES_EDITTHISSCHEMA</td>
<td>33732</td>
</tr>
<tr>
<td>Save Diagram...</td>
<td>IDC_WSDL_SAVE_DIAGRAM</td>
<td>39451</td>
</tr>
<tr>
<td>Generate Documentation...</td>
<td>ID_WSDL_GENERATEDDOCUMENTATION</td>
<td>39452</td>
</tr>
<tr>
<td>Reparse WSDL Document</td>
<td>IDC_WSDL_REPARSE</td>
<td>33774</td>
</tr>
<tr>
<td>Convert to WSDL 2.0</td>
<td>IDC_WSDL_CONVERT_TO_WSDL20</td>
<td>39453</td>
</tr>
<tr>
<td>Generate WSDL Program Code with MapForce...</td>
<td>IDC_WSDL_GENERATE_CODE_MAPFORCE</td>
<td>34122</td>
</tr>
</tbody>
</table>

4.6.14 "SOAP" Menu

The "SOAP" 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 New SOAP Request...</td>
<td>ID_SOAP_GENERATESOAPMESSAGE</td>
<td>34224</td>
</tr>
<tr>
<td>Send Request to Server...</td>
<td>ID_SOAP_SENDREQUESTTOSERVER</td>
<td>34225</td>
</tr>
<tr>
<td>SOAP Request Settings...</td>
<td>ID_SOAP_SOAPREQUESTSETTINGS</td>
<td>34227</td>
</tr>
<tr>
<td>Soap Debugger Session</td>
<td>ID_SOAP_SOAPDEBUGGER</td>
<td>34226</td>
</tr>
<tr>
<td>Go</td>
<td>ID_SOAPDEBUGGER_BUTTONPLAY</td>
<td>34221</td>
</tr>
<tr>
<td>Single Step</td>
<td>ID_SOAPDEBUGGER_SINGLESTEP</td>
<td>34222</td>
</tr>
<tr>
<td>Break on Next Request</td>
<td>ID_SOAPDEBUGGER_BREAKONNEXTREQUEST</td>
<td>34219</td>
</tr>
<tr>
<td>Break on Next Response</td>
<td>ID_SOAPDEBUGGER_BREAKONNEXT</td>
<td>34220</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>Stop the Proxy Server</td>
<td>ID_SOAPDEBUGGER_STOPSERVER</td>
<td>34223</td>
</tr>
<tr>
<td>Soap Debugger Options</td>
<td>ID_SOAPDEBUGGEROPTIONS</td>
<td>34218</td>
</tr>
</tbody>
</table>

### 4.6.15 "XBRL" Menu

The "XBRL" menu has the following commands:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Command name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arcroles…</td>
<td>IDC_XMLSPYXBREditor_ARCROLES</td>
<td>34114</td>
</tr>
<tr>
<td>Linkroles…</td>
<td>IDC_XMLSPYXBREditor_LINKROLES</td>
<td>34115</td>
</tr>
<tr>
<td>Namespace Prefixes…</td>
<td>IDC_XMLSPYXBREditorNAMESPACE</td>
<td>34116</td>
</tr>
<tr>
<td>Set Target Namespace…</td>
<td>IDC_XMLSPYXBREditor_SETTARGETNAMESPACE</td>
<td>34039</td>
</tr>
<tr>
<td>Parameter Values…</td>
<td>IDC_ICXBREditor_PARAMETER_VALUES</td>
<td>38913</td>
</tr>
<tr>
<td>Import/Reference…</td>
<td>IDC_XMLSPYXBREditor_IMPORT_REFERENCE</td>
<td>34137</td>
</tr>
<tr>
<td>Find Component By Id…</td>
<td>IDC_ICXBREditor_FIND_COMPONENTBYID</td>
<td>38893</td>
</tr>
<tr>
<td>Generate Documentation…</td>
<td>IDC_XMLSPYXBREditor_GENERATEDOCUMENTATION</td>
<td>34125</td>
</tr>
<tr>
<td>View Settings…</td>
<td>IDC_XMLSPYXBREditor_VIEWSETTINGS</td>
<td>34113</td>
</tr>
<tr>
<td>Generate XBRL from DB, Excel, CSV with MapForce…</td>
<td>IDC_XBRL_GENERATE_WITH_MAPFORCE</td>
<td>34045</td>
</tr>
<tr>
<td>Present XBRL as HTML/PDF/Word with StyleVision…</td>
<td>IDC_XBRL_PRESENT_WITH_STYLEVISION</td>
<td>34121</td>
</tr>
<tr>
<td>Execute Formula…</td>
<td>IDC_XBRL_EXECUTE_FORMULA</td>
<td>34305</td>
</tr>
<tr>
<td>Execute Formula on Server (high-performance)…</td>
<td>IDC_XBRL_EXECUTE_FORMULA_RAPTOR</td>
<td>34352</td>
</tr>
<tr>
<td>Generate Table…</td>
<td>IDC_XBRL_GENERATE_TABLE</td>
<td>34304</td>
</tr>
<tr>
<td>Generate Table on Server (high-performance)…</td>
<td>IDC_XBRL_GENERATE_TABLE_RAPTOR</td>
<td>34353</td>
</tr>
<tr>
<td>Transform Inline XBRL</td>
<td>IDC_IXBRL_TRANSFORM</td>
<td>34354</td>
</tr>
</tbody>
</table>
4.6.16 "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>Spelling...</td>
<td>IDC_SPELL_CHECK</td>
<td>34154</td>
</tr>
<tr>
<td>Spelling Options...</td>
<td>IDC_SPELL_OPTIONS</td>
<td>34155</td>
</tr>
<tr>
<td>Scripting Editor...</td>
<td>ID_SCRIPTFORMEDITOR_EDIT_PROJECT</td>
<td>39666</td>
</tr>
<tr>
<td>none</td>
<td>ID_SCRIPTFORMEDITOR_EXECUTE_MACRO_MENU_UPPDATE</td>
<td>39600</td>
</tr>
<tr>
<td>Compare Open File With...</td>
<td>ID_XMLDIFF_CHOOSE_FILES</td>
<td>34235</td>
</tr>
<tr>
<td>Compare Directories...</td>
<td>ID_XMLDIFF_CHOOSE_DIRECTORIES</td>
<td>34234</td>
</tr>
<tr>
<td>Compare Options...</td>
<td>ID_XMLDIFF_SETTINGS</td>
<td>34236</td>
</tr>
<tr>
<td>Global Resources</td>
<td>IDC_GLOBALRESOURCES</td>
<td>34292</td>
</tr>
<tr>
<td>Manage Raptor Servers ...</td>
<td>IDC_VALIDATE_RAPTOR_MANAGER</td>
<td>34311</td>
</tr>
<tr>
<td>none</td>
<td>IDC_VALIDATE_RAPTOR_NOCFG</td>
<td>34326</td>
</tr>
<tr>
<td>Customize...</td>
<td>IDC_APP_TOOLS_CUSTOMIZE</td>
<td>32959</td>
</tr>
<tr>
<td>Options...</td>
<td>IDC_SETTINGS</td>
<td>34133</td>
</tr>
</tbody>
</table>

4.6.17 "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 horizontally</td>
<td>ID_WINDOW_TILE_HORZ</td>
<td>57651</td>
</tr>
<tr>
<td>Tile vertically</td>
<td>ID_WINDOW_TILE_VERT</td>
<td>57652</td>
</tr>
<tr>
<td>Project window</td>
<td>IDC_PROJECT_WINDOW</td>
<td>34128</td>
</tr>
<tr>
<td>Info window</td>
<td>IDC_INFO_WINDOW</td>
<td>34085</td>
</tr>
</tbody>
</table>
### 4.6.18 "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>Keyboard Map...</td>
<td>IDC_HELP_KEYMAPDLG</td>
<td>32968</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>XMLSpy 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>Authentic on the Internet...</td>
<td>IDC_OPEN_HOME_PAGE</td>
<td>32964</td>
</tr>
<tr>
<td>Authentic Training...</td>
<td>IDC_OPEN_TRAINING_PAGE</td>
<td>32965</td>
</tr>
<tr>
<td>About XMLSpy...</td>
<td>ID_APP_ABOUT</td>
<td>57664</td>
</tr>
</tbody>
</table>
4.7 Object Reference

Objects:
XMLSpyCommand
XMLSpyCommands
XMLSpyControl
XMLSpyControlDocument
XMLSpyControlPlaceHolder

To give access to standard XMLSpy functionality, objects of the XMLSpy automation interface can be accessed as well. See XMLSpyControl.Application, XMLSpyControlDocument.Document and XMLSpyControlPlaceHolder.Project for more information.

4.7.1 XMLSpyCommand

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>
</table>
| An executable command   | • ID is greater than zero  
                          | • IsSeparator is false  
                          | • SubCommands is empty |
| A command container     | • ID is zero  
                          | • IsSeparator is true  
                          | • SubCommands contains a collection of Command objects |
| Separator               | • ID is zero  
                          | • IsSeparator is true |

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:

\[(\text{ALT+})(\text{CTRL+})(\text{SHIFT+})\text{key}\]

Where key is converted using the Windows Platform SDK function `GetKeyNameText`.

**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.

**IsSeparator**

*Property:* IsSeparator as boolean

*Description:* The property returns true if the command object is a menu separator; false otherwise. See also `Command`.

**Label**

*Property:* Label as string

*Description:* This property gets the text of the command as it is displayed in the graphical user interface of XMLSpy. 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.

**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 `<ApplicationFolder>\Examples\ActiveX\Images` of your XMLSpy installation.

**StatusText**

*Property:* StatusText as string

*Description:* The status text is the text shown in the status bar of XMLSpy 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.

**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`.

**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.

### 4.7.2 XMLSpyCommands

*Properties:*

- `Count`
- `Item`

*Description:*
Collection of `Command` objects to get access to command labels and IDs of the `XMLSpyControl`. Those commands can be executed with the `Exec` method and their status can be queried with `QueryStatus`.

*Count*  
*Property:* `Count` as `long`

*Description:*
Number of `Command` objects on this level of the collection.

*Item*  
*Property:* `Item (n as long)` as `Command`

*Description:*
Gets the command with the index `n` in this collection. Index is 1-based.

### 4.7.3 XMLSpyControl

*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 XMLSpy library is used in the Application Level mode.

CLSID: a258bba2-3835-4c16-8590-72b44f52c471
ProgID: Altova.XMLSpyControl

Properties
The following properties are defined:
- IntegrationLevel
- EnableUserPrompts
- Appearance
- BorderStyle

Command related properties:
- CommandsList
- MainMenu
- Toolbars
- CommandsStructure (deprecated)

Access to XMLSpyAPI:
- Application

*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.

*Application*

*Property*: Application as Application
**Dispatch Id:** 1

**Description:**
The `Application` property gives access to the `Application` object of the complete `XMLSpy` automation server API. The property is read-only.

**BorderStyle**

**Property:** `BorderStyle` as `short`

**Dispatch Id:** -504

**Description:**
A value of 1 displays the control with a thin border. Default value is 0.

**CommandsList**

**Property:** `CommandList` as `Commands` (read-only)

**Dispatch Id:** 1004

**Description:**
This property returns a flat list of all commands defined available with `XMLSpyControl`. To get commands organized according to their menu structure, use `MainMenu`. To get toolbar commands, use `Toolbars`.

```csharp
public void GetAllXmlSpyCommands()
{
    // Get all commands from the XMLSpy ActiveX control assigned to the
    // current form
    XMLSpyControlLib.XMLSpyCommands commands =
        this.axXMLSpyControl1.CommandList;
    // Iterate through all commands
    for (int i = 0; i < commands.Count; i++)
    {
        // Get each command by index and output it to the console
        XMLSpyControlLib.XMLSpyCommand cmd = axXMLSpyControl1.CommandList[i];
        Console.WriteLine("{0} {1} {2}", cmd.ID, cmd.Name, cmd.Label.Replace("&", "");
    }
}
```

**C# example**

**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.

IntegrationLevel

Property: IntegrationLevel as ICAcitiveXIntegrationLevel

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 XMLSpyControl object.

MainMenu

Property: MainMenu as Command (read-only)

Dispatch Id: 1003

Description:
This property provides information about the structure and commands available in the XMLSpyControl main menu, as a Command object. The Command object contains all available submenus of XMLSpy (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 GetXmlSpyMenus()
{
    // Get the main menu from the XMLSpy ActiveX control assigned to the current form
    XMLSpyControllLib.XMLSpyCommand mainMenu = this.axXMLSpyControl1.MainMenu;

    // Loop through entries of the main menu (e.g. File, Edit, etc.)
    for (int i = 0; i < mainMenu.SubCommands.Count; i++)
    {
        XMLSpyControllLib.XMLSpyCommand menu = mainMenu.SubCommands[i];
        Console.WriteLine("[0] menu has {1} children items (including separators)", menu.Label.Replace("&", "), menu.SubCommands.Count);
    }
}
```

C# example
Toolbars

Property: Toolbars as Commands (read-only)

Dispatch Id: 1005

Description:
This property provides information about the structure of XMLSpyControl toolbars, as a Command object. The Command object contains all available toolbars of XMLSpy. 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).

```csharp
public void GetXmlSpyToolbars()
{
    // Get the application toolbars from the StyleVision ActiveX control
    // assigned to the current form
    XMLSpyControlLib.XMLSpyCommands toolbars = this.axXMLSpyControl1.Toolbars;

    // Iterate through all toolbars
    for (int i = 0; i < toolbars.Count; i++)
    {
        XMLSpyControlLib.XMLSpyCommand 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++)
        {
            XMLSpyControlLib.XMLSpyCommand 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

Methods
The following methods are defined:

- Open
- Exec
- QueryStatus
**Exec**

**Method:** `Exec(nCmdID as long) as boolean`

**Dispatch Id:** 6

**Description:**
This method calls the XMLSpy 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 `XMLSpyControlDocument.Open` and `XMLSpyControlPlaceHolder.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 XMLSpy command. If `QueryStatus` returns a value of 1 or 5, the command is disabled.

**Events**
The XMLSpyControl ActiveX control provides the following connection point events:

- `OnUpdateCmdUI`
OnCloseEditingWindow

**Event:** OnCloseEditingWindow (i_strFilePath as String) as boolean

**Dispatch Id:** 1002

**Description:**
This event is triggered when XMLSpy 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 XMLSpyControl should try to close the editor and destroy the associated document control.

OnContextChanged

**Event:** OnContextChanged (i_strContextName as String, i_bActive as bool) as bool

**Dispatch Id:** 1004

**Description:**
This event is triggered when XMLSpy activates or de-actives one of the following operational contexts:

- XSLT Profiling - "XSLTProfiling" is passed as the context name
- XSLT / XQuery debugging - "DebuggingXSLT" is passed as the context name

OnDocumentOpen

**Event:** OnDocumentOpen (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 XMLSpy 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 XMLSpyControlDocument.OnDocumentOpen instead.

OnFileChangedAlert

**Event:** OnFileChangedAlert (i_strFilePath as String) as bool

**Dispatch Id:** 1001

**Description:**
This event is triggered when a file loaded with XMLSpyControl is changed on the hard disk by another application. Clients should return true, if they handled the event, or false, if XMLSpy should handle it in its customary way, i.e. prompting the user for reload.

**OnLicenseProblem**

*Event:* OnLicenseProblem(*i_strLicenseProblemText as String*)

*Dispatch Id:* 1005

*Description:* This event is triggered when XMLSpyControl 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).

**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 XMLSpy. 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 XMLSpyControl, since internal access is required. Otherwise, the file can be opened with different editors.

**OnToolWindowUpdated**

*Event:* OnToolWindowUpdated(*pToolWnd as long*)

*Dispatch Id:* 1006

*Description:* This event is triggered when the tool window is updated.

**OnUpdateCmdUI**

*Event:* OnUpdateCmdUI ()

*Dispatch Id:* 1003

*Description:* Called frequently to give integrators a good opportunity to check status of XMLSpy commands
using `XMLSpyControl.QueryStatus`. Do not perform long operations in this callback.

**OnValidationWindowUpdated Event**: `OnValidationWindowUpdated()`

**Dispatch Id**: 3

**Description**:
This event is triggered whenever the validation output window is updated with new information.

### 4.7.4 XMLSpyControlDocument

**Properties**:
- Appearance
- BorderStyle
- Document
- IsModified
- Path
- ReadOnly

**Methods**:
- Exec
- New
- Open
- QueryStatus
- Reload
- Save
- SaveAs

**Events**:
- OnDocumentOpened
- OnDocumentClosed
- OnModifiedFlagChanged
- OnContextChanged
- OnFileChangedAlert
- OnActivate

If the XMLSpyControl is integrated in the Document Level mode each document is displayed in an own object of type `XMLSpyControlDocument`. The `XMLSpyControlDocument` 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**: 52A552E6-2AB8-4e3e-B545-BE998233DDA0
**ProgID**: Altova.XMLSpyControlDocument

**Properties**
The following properties are defined:
- ReadOnly
- IsModified
Access to XMLSpyAPI:

**Document**

**Property:** Document as Document

**Dispatch Id:** 1

**Description:**
The Document property gives access to the Document object of the XMLSpy automation server API. This interface provides additional functionality which can be used with the document loaded in the control. The property is read-only.

**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.
Path
Property: Path as string

Dispatch Id: 1005

Description:
Sets or gets the full path name of the document loaded into the control.

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.

Methods
The following methods are defined:

Document handling:
New
Open
Reload
Save
SaveAs

Command Handling:
Exec
QueryStatus

Exec
Method: Exec (nCmdID as long) as boolean

Dispatch Id: 8

Description:
Exec calls the XMLSpy 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 XMLSpyControl. To get toolbar commands, use the Toolbars property of the XMLSpyControl.
**New**

*Method:* `New () as boolean`

*Dispatch Id:* 1000

*Description:*  
This method initializes a new document inside the control.

**Open**

*Method:* `Open (strFileName as string) as boolean`

*Dispatch Id:* 1001

*Description:*  
`Open` loads the file `strFileName` as the new document into the control.

**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>
<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 XMLSpy command. If `QueryStatus` returns a value of 1 or 5 the command is disabled. The client should call the `QueryStatus` method of the document control if there is currently an active document available in the application.

**Reload**

*Method:* `Reload () as boolean`

*Dispatch Id:* 1002

*Description:*  
`Reload` updates the document content from the file system.
Save
Method: Save() as boolean

Dispatch Id: 1003

Description:
Save saves the current document at the location Path.

SaveAs
Method: SaveAs(strFileName as string) as boolean

Dispatch Id: 1004

Description:
SaveAs sets Path to strFileName and then saves the document to this location.

Events
The XMLSpyControlDocument ActiveX control provides following connection point events:

OnDocumentOpened
OnDocumentClosed
OnModifiedFlagChanged
OnContextChanged
OnFileChangedAlert
OnActivate
OnSetTitle

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.

OnContextChanged
Event: OnContextChanged(i_strContextName as String, i_bActive as bool) as bool

Dispatch Id: 1004

Description:
This event is triggered when this document is shown in a different XMLSpy view. The following values are passed:

- Grid view - "View_0" is passed as the context name
- Text view - "View_1" is passed as the context name
Object Reference

1452

ActiveX Integration

Programmers’ Reference

Browser view - "View_2" is passed as the context name
Schema view - "View_3" is passed as the context name
Authentic view - "View_4" is passed as the context name
WSDL view - "View_5" is passed as the context name

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 XMLSpy automation interface and should be used with care.

OnDocumentOpened


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 XMLSpy automation interface, and can be used to query for more details about the document, or perform additional operations.

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.

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 XMLSpy should handle it in its customary way, i.e. prompting the user for reload.
**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.

**OnSetEditorTitle**

*Event:* OnSetEditorTitle()

*Dispatch Id:* 1006

*Description:* This event is being raised when the contained document is being internally renamed.

### 4.7.5 XMLSpyControlPlaceHolder

**Properties available for all kinds of placeholder windows:**

- PlaceholderWindowID

**Properties for project placeholder window:**

- Project

**Methods for project placeholder window:**

- OpenProject
- CloseProject

The XMLSpyControlPlaceHolder control is used to show the additional XMLSpy 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: 135DEEF4-6DF0-47c2-8F8C-F145F5F3F672

ProgID: Altova.XMLSpyControlPlaceHolder

**Properties**

The following properties are defined:

- PlaceholderWindowID

Access to XMLSpyAPI:

- Project
**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.

---

**PlaceholderWindowID**

*Property*: PlaceholderWindowID as XMLSpyControlPlaceholderWindow

*Dispatch Id*: 1

*Description*:  
This property specifies which XMLSpy 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 XMLSpyControlPlaceholderWindow enumeration. The control changes its state immediately and shows the new XMLSpy window.

---

**Project**

*Property*: Project as Project (read-only)

*Dispatch Id*: 2

*Description*:  
The Project property gives access to the Project object of the XMLSpy 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 XMLSpyXProjectWindow (=3). The property is read-only.

---

**Methods**

The following method is defined:

- **OpenProject**
- **CloseProject**

---

**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).

**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 XMLSpyXProjectWindow (=3).

**Events**  
The XMLSpyControlPlaceholder ActiveX control provides following connection point events:  
OnModifiedFlagChanged

**OnModifiedFlagChanged**  
**Event:** OnModifiedFlagChanged (i_bIsModified as boolean)  
**Dispatch Id:** 1  
**Description:** This event gets triggered only for placeholder controls with a PlaceholderWindowID of XMLSpyXProjectWindow (=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.

**OnSetLabel**  
**Event:** OnSetLabel (i_strNewLabel as string)  
**Dispatch Id:** 1000  
**Description:** Raised when the title of the placeholder window is changed.

### 4.7.6 Enumerations

The following enumerations are defined:  
ICActiveXIntegrationLevel  
XMLSpyControlPlaceholderWindow
ICActiveXIntegrationLevel

Possible values for the IntegrationLevel property of the XMLSpyControl.

ICActiveXIntegrationOnApplicationLevel = 0
ICActiveXIntegrationOnDocumentLevel = 1

XMLSpyControlPlaceholderWindow

This enumeration contains the list of the supported additional XMLSpy windows.

XMLSpyControlNoToolWnd = -1
XMLSpyControlEntryHelperTopToolWnd = 0
XMLSpyControlEntryHelperMiddleToolWnd = 1
XMLSpyControlEntryHelperBottomToolWnd = 2
XMLSpyControlValidatorOutputToolWnd = 3
XMLSpyControlProjectWindowToolWnd = 4
XMLSpyControlXSLTDebuggerContextToolWnd = 5
XMLSpyControlXSLTDebuggerCallstackToolWnd = 6
XMLSpyControlXSLTDebuggerVariableToolWnd = 7
XMLSpyControlXSLTDebuggerWatchToolWnd = 8
XMLSpyControlXSLTDebuggerTemplateToolWnd = 9
XMLSpyControlXSLTDebuggerInfoToolWnd = 10
XMLSpyControlXSLTDebuggerMessageToolWnd = 11
XMLSpyControlXSLTDebuggerTraceToolWnd = 12
XMLSpyControlSOAPDebuggerToolWnd = 13
XMLSpyControlXPathProfilerListToolWnd = 14
XMLSpyControlXPathProfilerTreeToolWnd = 15
XMLSpyControlXPathDialogToolWnd = 16
XMLSpyControlDBObjectQueryManagerToolWnd = 17
XMLSpyControlInfoToolWnd = 18
XMLSpyControlXSLOutlineToolWnd = 19
XMLSpyControlSchemaFindToolWnd = 20
XMLSpyControlXBRLFindToolWnd = 21
XMLSpyControlChartsToolWnd = 22
Appendices
Appendices

These appendices contain technical information about XMLSpy and important licensing information. Each appendix contains sub-sections as given below:

**Engine Information**
- XSLT and XQuery Engine Information
- XSLT and XQuery Extension Functions

**Datatype Conversions between DBs and XML Schemas**
- DBs to XML Schemas
- XML Schemas to DBs

**Technical Data**
- OS and memory requirements
- Altova XSLT and XQuery Engines
- Unicode support
- Internet usage

**License Information**
- Electronic software distribution
- Software activation and license metering
- End User License Agreement
XSLT and XQuery Engine Information

The XSLT and XQuery engines of XMLSpy 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 XMLSpy.

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 $E_1$ in a path expression $E_1/E_2$ 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
- XSLT 3.0
- XQuery 1.0
- XQuery 3.1
1.1 XSLT 1.0

The XSLT 1.0 Engine of XMLSpy 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 U+00A0 (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;.
1.2 XSLT 2.0

This section:

- Engine conformance
- Backward compatibility
- Namespaces
- Schema awareness
- Implementation-specific behavior

Conformance


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:

```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
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`. 
1.3 **XSLT 3.0**

The XSLT 3.0 Engine of XMLSpy conforms to the World Wide Web Consortium's (W3C's) [XSLT 3.0 Recommendation of 8 June 2017](https://www.w3.org/TR/xslt30/) and [XPath 3.1 Recommendation of 21 March 2017](https://www.w3.org/TR/xpath31/).

The XSLT 3.0 engine has the same implementation-specific characteristics as the XSLT 2.0 engine. Additionally, it includes support for a number of new XSLT 3.0 features: XPath/XQuery 3.1 functions and operators, and the [XPath 3.1 specification](https://www.w3.org/TR/xpath31/).

**Note:** The optional streaming feature is not supported currently. The entire document will be loaded into memory regardless of the value of the `streamable` attribute, and will be processed if enough memory is available. In 64-bit apps this should not be a problem. If memory does turn out to be an issue, a solution would be to add more memory to the system.
1.4 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 XMLSpy 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:

    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:
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.
1.5 XQuery 3.1

The XQuery 3.1 Engine of XMLSpy conforms to the World Wide Web Consortium's (W3C's) XQuery 3.1 Recommendation of 21 March 2017 and includes support for XPath and XQuery Functions 3.1. The XQuery 3.1 specification is a superset of the 3.0 specification. The XQuery 3.1 engine therefore supports XQuery 3.0 features.

Implementation-specific characteristics are the same as for XQuery 1.0.
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. 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 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>
</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.
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 \texttt{XP} symbol and call them XPath functions; those functions that can be used in the latter (XQuery) context are indicated with an \texttt{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 \texttt{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.

\begin{tabular}{|l|}
\hline
\textbf{XPath functions (used in XPath expressions in XSLT):} & XP1 XP2 XP3.1 \\
\hline
\textbf{XSLT functions (used in XPath expressions in XSLT):} & XSLT1 XSLT2 XSLT3 \\
\hline
\end{tabular}

\begin{tabular}{|l|}
\hline
\textbf{XQuery functions (used in XQuery expressions in XQuery):} & XQ1 XQ3.1 \\
\hline
\end{tabular}

**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
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):

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>XP1</td>
<td>XP2</td>
<td>XP3.1</td>
</tr>
</tbody>
</table>

### XSLT functions (used in XPath expressions in XSLT):

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLT1</td>
<td>XSLT2</td>
<td>XSLT3</td>
</tr>
</tbody>
</table>

### XQuery functions (used in XQuery expressions in XQuery):

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>XQ1</td>
<td>XQ3.1</td>
<td></td>
</tr>
</tbody>
</table>

#### Standard functions [altova:]

- **distinct-nodes**

  ```xml
  <xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
  ```

  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**

  ```xml
  <xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
  ```

  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*`.
<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 '//Name[1]"

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:**
Dynamic XPath expression with no dynamic variable:

```xml
<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:]

```xml
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.

### XBRL functions

Altova XBRL functions can be used only with editions of Altova products that have XBRL support.

#### xbrl-footnotes [altova:]

```xml
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:]

```xml
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.

### 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
altova:build-date
altova:build-duration
altova:build-time
altova:current-dateTime-no-TZ
altova:current-date-no-TZ
altova:current-time-no-TZ
altova:date-no-TZ
altova:dateTime-no-TZ
altova:days-in-month
altova:hours-from-dateTimeDuration-accumulated
altova:minutes-from-dateTimeDuration-accumulated
altova:seconds-from-dateTimeDuration-accumulated
Add a duration to xs:dateTime  
XP3.1 XQ3.1

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:]

  ```xml
  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:]

  ```xml
  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:]

  ```xml
  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"), -2)
  returns 2013-01-13T14:00:00
  ```
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(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  XP3.1 XQ3.1
These functions add a duration to xs:date and return xs:date. The xs:date 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  XP3.1 XQ3.1
  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  XP3.1 XQ3.1
  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  XP3.1 XQ3.1
  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  XP3.1 XQ3.1
These functions add a duration to xs:date and return xs:date. The xs:date type has a format of CCYY-MM-DD.

- format-duration [altova:]

  format-duration(Date as xs:date, Format as xs:string) as xs:string  XP3.1 XQ3.1
  Formats a date according to the specified format. The second argument is the format to use for the date. Examples include "dd/MM/yyyy" or "d/M/y".

  Examples
  - format-duration(xs:date("2014-01-15"), "dd/MM/yyyy") returns 15/01/2014
  - format-duration(xs:date("2014-01-15"), "d/M/y") returns 15/01/14

  [ Top ]
altova:format-duration(Duration as xs:duration, Picture as xs:string) as xs:string

Formats a duration, which is submitted as the first argument, according to a picture string submitted as the second argument. The output is a text string formatted according to the picture string.

**Examples**

- `altova:format-duration(xs:duration("P2DT2H53M11.7S"), "Days:[D01] Hours:[H01] Minutes:[m01] Seconds:[s01] Fractions:[f0]")` returns "Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7"
- `altova:format-duration(xs:duration("P3M2DT2H53M11.7S"), "Months:[M01] Days:[D01] Hours:[H01] Minutes:[m01]"") returns "Months:03 Days:02 Hours:02 Minutes:53"

parse-duration [altova:]

altova:parse-duration(InputString as xs:string, Picture as xs:string) as xs:duration

Takes a patterned string as the first argument, and a picture string as the second argument. The input string is parsed on the basis of the picture string, and an xs:duration is returned.

**Examples**

- `altova:parse-duration("Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7"), "Days:[D01] Hours:[H01] Minutes:[m01] Seconds:[s01] Fractions:[f0]")` returns "P2DT2H53M11.7S"

Add a duration to xs:time

These functions add a duration to xs:time and return xs:time. The xs:time type has a lexical form of hh:mm:ss.sss. An optional time zone may be suffixed. The letter Z indicates Coordinated Universal Time (UTC). All other time zones are represented by their difference from UTC in the format +hh:mm, or -hh:mm. If no time zone value is present, it is considered unknown; it is not assumed to be UTC.

add-hours-to-time [altova:]

altova:add-hours-to-time(Time as xs:time, Hours as xs:integer) as xs:time

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

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.
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:add-seconds-to-time(Time as xs:time, Minutes 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: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

**current-dateTime-no-TZ**

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 \texttt{2014-01-15T14:00:00+01:00}:

- \texttt{altova:current-dateTime-no-TZ()} returns \texttt{2014-01-15T14:00:00}

current-time-no-TZ [altova:]

\texttt{altova:current-time-no-TZ()} as \texttt{xs:time} \ XQ3.1

This function takes no argument. It removes the timezone part of \texttt{current-time()} (which is the current time according to the system clock) and returns an \texttt{xs:time} value.

Examples

If the current time is \texttt{14:00:00+01:00}:

- \texttt{altova:current-time-no-TZ()} returns \texttt{14:00:00}

date-no-TZ [altova:]

\texttt{altova:date-no-TZ(InputDate as xs:date) as xs:date} \ XQ3.1

This function takes an \texttt{xs:date} argument, removes the timezone part from it, and returns an \texttt{xs:time} value. Note that the date is not modified.

Examples

- \texttt{altova:date-no-TZ(xs:date("2014-01-15+01:00"))} returns \texttt{2014-01-15}

dateTime-no-TZ [altova:]

\texttt{altova:dateTime-no-TZ(InputDateTime as xs:dateTime) as xs:dateTime} \ XQ3.1

This function takes an \texttt{xs:dateTime} argument, removes the timezone part from it, and returns an \texttt{xs:dateTime} value. Note that neither the date nor the time is modified.

Examples

- \texttt{altova:dateTime-no-TZ(xs:date("2014-01-15T14:00:00+01:00"))} returns \texttt{2014-01-15T14:00:00}

time-no-TZ [altova:]

\texttt{altova:time-no-TZ(InputTime as xs:time) as xs:time} \ XQ3.1

This function takes an \texttt{xs:time} argument, removes the timezone part from it, and returns an \texttt{xs:time} value. Note that the time is not modified.

Examples

- \texttt{altova:time-no-TZ(xs:time("14:00:00+01:00"))} returns \texttt{14:00:00}
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**

```xml
altova:minutes-from-dayTimeDuration-accumulated(DayAndTime as xs:duration) as xs:integer
```

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`  

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 (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, Format as xs:integer)`  

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.
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.

[ Top ]

Return the week number from xs:dateTime or xs:date XP2 XQ1 XP3.1 XQ3.1
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).

**weeknumber-from-date [altova:]**

```xml
altova:weeknumber-from-date(Date as xs:date, Calendar as xs:integer) 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**

- `altova:weeknumber-from-date(xs:date("2014-03-23"), 0)` returns 13
- `altova:weeknumber-from-date(xs:date("2014-03-23"), 1)` returns 12
- `altova:weeknumber-from-date(xs:date("2014-03-23"), 2)` returns 13
- `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.

**weeknumber-from-dateTime [altova:]**

```xml
altova:weeknumber-from-dateTime(DateTime as xs:dateTime, Calendar as xs:integer) 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:]**
  
  \[
  \text{altova:build-date(Year as xs:integer, Month as xs:integer, Date as xs:integer) as xs:date XP3.1 XQ3.1}
  \]
  
  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**
  
  - \text{altova:build-date(2014, 2, 03)} \text{ returns 2014-02-03}

- **build-time [altova:]**
  
  \[
  \text{altova:build-time(Hours as xs:integer, Minutes as xs:integer, Seconds as xs:integer) as xs:time XP3.1 XQ3.1}
  \]
  
  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**
  
  - \text{altova:build-time(23, 4, 57)} \text{ returns 23:04:57}

- **build-time [altova:]**
  
  \[
  \text{altova:build-time(Hours as xs:integer, Minutes as xs:integer, Seconds as xs:integer, TimeZone as xs:string) as xs:time XP3.1 XQ3.1}
  \]
  
  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**
  
  - \text{altova:build-time(23, 4, 57, '+1')} \text{ returns 23:04:57+01:00}

- **build-duration [altova:]**
  
  \[
  \text{altova:build-duration(Years as xs:integer, Months as xs:integer) as}
  \]
  
  The first and second arguments are, respectively, the number of years and months. The two arguments are combined to build a value of xs:duration type. The values of the integers must be within the correct range of that particular duration part. For example, the second (Months) argument should not be greater than 12.
xs:yearMonthDuration  XP3.1  XQ3.1
Takes two arguments to build a value of type xs:yearMonthDuration. The first argument 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

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

Construct date, dateTime, and time datatypes from string input  XP2  XQ1  XP3.1  XQ3.1
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:]

  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.

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Date</td>
</tr>
<tr>
<td>M</td>
<td>Month</td>
</tr>
<tr>
<td>Y</td>
<td>Year</td>
</tr>
</tbody>
</table>

  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

- `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

ParseDateTime [altova:]

```
altova:parse-dateTime(Date 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

- `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

ParseTime [altova:]

```
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: Hour
- m: minutes
- s: seconds

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=\([H]h\)\(m\)\(s\)\)\) returns 13:56:24`
  - `altova:parse-time("time=24s56m13h", ",\(time=\([s]\)\(m\)\(H\)\)\) 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.

- **age [altova:]**

  `altova:age(StartDate as xs:date) as xs:integer`  
  ![XP3.1](#)  ![XQ3.1](#)

  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**
    - If the current date is **2014-01-15**:
      - `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`

  `altova:age(StartDate as xs:date, EndDate as xs:date) as xs:integer`  
  ![XP3.1](#)  ![XQ3.1](#)

  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)`

## Path/Path Functions: Geolocation

The following geolocation XPath/XQuery extension functions are supported in the current version of XMLSpy 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):

<table>
<thead>
<tr>
<th>Function</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>XP1</td>
<td></td>
</tr>
<tr>
<td>XP2</td>
<td></td>
</tr>
<tr>
<td>XP3.1</td>
<td></td>
</tr>
</tbody>
</table>
XSLT and XPath/XQuery Functions

| 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
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" version="1.0">
  <xs:element name="altova:format-geolocation">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Latitude" type="xs:decimal"/>
        <xs:element name="Longitude" type="xs:decimal"/>
        <xs:element name="GeolocationOutputStringFormat" type="xs:integer"/>
      </xs:sequence>
    </xs:complexType>
    <xs:attribute name="as" type="xs:string"/>
  </xs:element>
</xs:schema>
```

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"`
- `altova:format-geolocation(33.33, -22.22, 2)` returns the `xs:string "33.33\N 22.22\W"`
- `altova:format-geolocation(-33.33, 22.22, 2)` returns the `xs:string "33.33\S 22.22\E"`
- `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>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Degrees, minutes, decimal seconds, with suffixed orientation (N/S, E/W)</td>
<td>33°55′11.11″N 22°44′66.66″W</td>
</tr>
<tr>
<td>2</td>
<td>Decimal degrees, with suffixed orientation (N/S, E/W)</td>
<td>33.33N 22.22W</td>
</tr>
<tr>
<td>3</td>
<td>Degrees, minutes, decimal seconds, with prefixed sign (+/-); plus sign for (N/E) is optional</td>
<td>+/-33°55′11.11″ -22°44′66.66″</td>
</tr>
</tbody>
</table>
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+

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°51'29.6"N 24°17'40.2"E") returns the sequence of two xs:decimals (48.8582222222222, 24.2945)
- altova:parse-geolocation("48°51'29.6"N 24°17'40.2"E") 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)**
  \[D°M’S.SS”N/S  D°M’S.SS”W/E\]
  Example: 33°55.11"N  22°44.55.25"W

- **Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (N/W) is optional**
  \[+/-D°M’S.SS”  +/-D°M’S.SS”\]
  Example: 33°55.11"  -22°44.55.25"W

- **Degrees, decimal minutes, with suffixed orientation (N/S, W/E)**
  \[D°M.MM’S’N/S  D°M.MM’S’W/E\]
  Example: 33°55.55'S  22°44.44'W

- **Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (N/W) is optional**
  \[+/-D°M.MM’S’  +/-D°M.MM’S’\]
  Example: +33°55.55'S  -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"W
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>
<tr>
<td>21.91</td>
<td></td>
<td>11.73</td>
<td></td>
<td></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'5.55"S/N    D°M'S'5.55"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'5.55"      +/-D°M'S'5.55"
  ```

  **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)

Example: 33.33N  22.22W

Decimal degrees, with prefixed sign (+/-); the plus sign for (N/W) is optional

Example: 33.33  -22.22

Examples of format-combinations:

<table>
<thead>
<tr>
<th>Latitude Value Format</th>
<th>Direction</th>
<th>Longitude Value Format</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>33.33N</td>
<td>S</td>
<td>151 13 11.73E</td>
<td>E</td>
</tr>
<tr>
<td>33.33 22.45</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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 21.91</td>
<td>S</td>
<td>151 13 11.73E</td>
<td>E</td>
<td>33°51'21.91&quot;S 151°13'11.73&quot;E</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 XQ3.1 XP3.1`

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’N 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’N 22°44.44’W

- Decimal degrees, with suffixed orientation (N/S, W/E)
  \[D.DD/N/S \quad D.DD/W/E\]
  **Example:** 33.33N 22.22W

- Decimal degrees, with prefixed sign (+/-); the plus sign for (N/W) is optional
  \[+/-D.DD \quad +/-D.DD\]
  **Example:** 33.33N 22.22W

**Examples of format-combinations:**
- 33.33 22°44’55.25”W
- 33.33 22°44’55.25”W
- 33.33 22.22W
- 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°</td>
</tr>
<tr>
<td>21.91</td>
<td></td>
<td>11.73</td>
<td></td>
<td>13’11.73”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)**
  
  D°M’S.SS”N/S  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”  +/-D°M’S.SS’

  **Example:** 33°55’11.11”  -22°44’55.25”

- **Degrees, decimal minutes, with suffixed orientation (N/S, 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
  
  \(+/-D^\circ M.MM'\) \(+/-D^\circ M.MM'\)

  Example: +33°55.55′ −22°44.44′

- Decimal degrees, with suffixed orientation (N/S, W/E)

  \(D.DD/D.DDW/E\)

  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>GPSLatitudeRef</th>
<th>GPSLongitude</th>
<th>GPSLongitudeRef</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″S 151°13′11.73″E</td>
</tr>
</tbody>
</table>

geolocation-within-rectangle [altova:]

**altova:geolocation-within-rectangle(Geolocation as xs:string, RectCorner-1 as xs:string, RectCorner-2 as xs:string) as xs:boolean** XP3.1 XQ3.1

Determines whether 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 ("").

1. Degrees, minutes, decimal seconds, with suffixed orientation (N/S, W/E)
   D°M'S.SS"N/S  D°M'S.SS"W/E
   Example: 33°55'11.11"N  22°44'55.25"W

2. Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (N/W) is optional
   +/-D°M'S.SS"  +/-D°M'S.SS"
   Example: 33°55'11.11"  -22°44'55.25"

3. 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

4. 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'

5. Decimal degrees, with suffixed orientation (N/S, W/E)
   D.DDN/S  D.DDW/E
   Example: 33.33N  22.22W

6. 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"W
33.33  22°44'55.25"W
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>

2.1.4 XPath/XQuery Functions: Image-Related

The following image-related XPath/XQuery extension functions are supported in the current version of XMLSpy 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.

- **XP1**
- **XP2**
- **XP3.1**

- **XSLT1**
- **XSLT2**
- **XSLT3**

- **XQ1**
- **XQ3.1**

- **suggested-image-file-extension [altova:]**

  **altova:suggested-image-file-extension**(Base64String as string) as string?

  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, 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**
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 `jpg` as the file's type and extension. In the second example, the submitted Base64 encoding does not provide usable file extension information.

**Examples**

To access any one attribute, use the function like this:

```xml
image-exif-data(//MyImages/Image20141130.01)/@GPSLatitude
image-exif-data(//MyImages/Image20141130.01)/@Geolocation
```

To access all the attributes, use the function like this:

```xml
image-exif-data(//MyImages/Image20141130.01)/@*
```

To access the names of all the attributes, use the following expression:

```xml
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 `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</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>

**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).

![Exif Orientation to Altova OrientationDegree diagram](image)

- **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
- GPSSLatitudeRef
- GPSSLatitude
- GPSSLongitudeRef
- GPSSLongitude
- GPSSAltitudeRef
- GPSSAltitude
- GPSTimeStamp
- GPSSatellites
- GPSStatus
- GPSMeasureMode
- GPSDOP
- GPSSpeedRef
- GPSSpeed
- GPSTrackRef
- GPSTrack
- GPSImgDirectionRef
- GPSImgDirection
- GPSMapDatum
- GPSDestLatitudeRef
- GPSDestLatitude
- GPSDestLongitudeRef
- GPSDestLongitude
- GPSDestBearingRef
- GPSDestBearing
- GPSDestDistanceRef
- GPSDestDistance
- GPSProcessingMethod
- GPSAreaInformation
- GPSDateStamp
- GPSDifferential

[ Top ]
### 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:]**

  ```xml
  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:]**

  ```xml
  altova:reset-auto-number(ID 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:]

  ```xml
  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:]

  ```xml
  altova:integer-to-hex-string(Integer as xs:integer) as xs:string
  ```

  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
```

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").`
attributes [altova:]

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()*

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()*
- altova:attributes("MyAttribute", "Rip") returns an unrecognized-flag error.
- altova:attributes("MyAttribute", ) returns a missing-second-argument error.

elements [altova:]

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()*

altova:elements(ElementName as xs:string, SearchOptions as xs:string) as


\textbf{element(*) \hspace{1em} XP3.1 \hspace{1em} XQ3.1}

Returns all elements that have a local name which is the same as the name supplied in the input argument, \textit{ElementName}. The search is case-sensitive and conducted along the \textit{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:
- \texttt{r} = switches to a regular-expression search; \textit{ElementName} must then be a regular-expression search string;
- \texttt{f} = If this option is specified, then \textit{ElementName} provides a full match; otherwise \textit{ElementName} need only partially match an element name to return that element. For example: if \texttt{f} is not specified, then \texttt{MyElem} will return \texttt{MyElement}.
- \texttt{i} = switches to a case-insensitive search;
- \texttt{p} = includes the namespace prefix in the search; \textit{ElementName} should then contain the namespace prefix, for example: \texttt{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.

\textbf{Examples}

- \texttt{altova:elements("MyElement", "rip")} returns \texttt{MyElement(*)}
- \texttt{altova:elements("MyElement", "pri")} returns \texttt{MyElement(*)}
- \texttt{altova:elements("MyElement", ")}) returns \texttt{MyElement(*)}
- \texttt{altova:attributes("MyElem", "rip")} returns \texttt{MyElement(*)}
- \texttt{altova:attributes("MyElements", "rfip")} returns no match
- \texttt{altova:elements("MyElement", "Rip")} returns an unrecognized-flag error.
- \texttt{altova:elements("MyElement", )} returns a missing-second-argument error.

\textbf{find-first [altova:]}\hspace{1em}

\texttt{altova:find-first((Sequence as item()), (Condition( Sequence-Item as xs:boolean)) as item()) \hspace{1em} XP3.1 \hspace{1em} XQ3.1}

This function takes two arguments. The first argument is a sequence of one or more items of any datatype. The second argument, \textit{Condition}, is a reference to an XPath function that takes one argument (has an arity of 1) and returns a boolean. Each item of \textit{Sequence} is submitted, in turn, to the function referenced in \textit{Condition}. (\textit{Remember:} This function takes a single argument.) The first \textit{Sequence} item that causes the function in \textit{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 \texttt{xs:integer 6}
  
  The \textit{Condition} argument references the XPath 3.0 inline function, \texttt{function()}, which declares an inline function named \$a and then defines it. Each item in the \textit{Sequence} argument of \texttt{altova:find-first} 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 first input value to satisfy this condition is returned as the result of \texttt{altova:find-first} (in this case 6).

- \texttt{altova:find-first((1 to 10), (function($a) {($a+3=7)})} returns \texttt{xs:integer 4}
Further examples
If the file C:\Temp\Customers.xml exists:

- \texttt{altova:find-first( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) ) returns xs:string C:.Temp\Customers.xml}

If the file C:\Temp\Customers.xml does not exist, and http://www.altova.com/index.html exists:


If the file C:\Temp\Customers.xml does not exist, and 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}

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 \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 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 \texttt{doc-available()}, which uses the string as a URI and tests whether a document node exists at the URI. If one does, the \texttt{doc-available()} evaluates to \texttt{true()} and that string is returned as the result of the \texttt{altova:find-first} function. \textit{Note about the \texttt{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.}

\begin{verbatim}
find-first-combination [altova:]
\end{verbatim}
\begin{verbatim}
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 XP3.1
\end{verbatim}
This function takes three arguments:
- The first two arguments, \texttt{Seq-01} and \texttt{Seq-02}, are sequences of one or more items of any datatype.
- The third argument, \texttt{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) ... (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:]

altova:find-first-pair((Seq-01 as item()*) as item()*, (Seq-02 as item()*) as item()*, (Condition( Seq-01-Item, Seq-02-Item as xs:boolean)) as item()*)

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+$b = 32}) returns the sequence of xs:integers (11, 21)
• **altova:find-first-pair**(11 to 20, 21 to 30, function($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: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

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

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

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 no result

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 current base URI, which is by default the URI of the XML document from which the function is loaded.

**for-each-attribute-pair [altova:]**

```
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: \((1+1, (1)\times 2, 3 +3, 4+())\). 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:]

\[
\text{altova:for-each-combination(FirstSequence as item()*, SecondSequence as item()*, Function($i,$j){$i || $j}) as item()}* \quad \text{XP3.1 XQ3.1}
\]

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:]

\[
\text{altova:for-each-matching-attribute-pair(Seq1 as element()?, Seq2 as element()?, Function as function()) as item()}* \quad \text{XP3.1 XQ3.1}
\]

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
  
  <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" />

- altova:for-each-matching-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, 33) if
  
  <Test-A att4="4" att1="1" att3="3" />
  <Test-B att3="3" att2="2" att1="1" />
substitute-empty [altova:]

```
altova:substitute-empty(FirstSequence as item(), SecondSequence as item())
```

XP3.1 XQ3.1

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)`

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.

<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>

camel-case [altova:]

```
altova:camel-case(InputString as xs:string) as xs:string
```

XP3.1 XQ3.1

Contracts 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 sequence of 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

\texttt{altova:camel-case(\texttt{InputString} \texttt{as xs:string}, \texttt{SplitChars} \texttt{as xs:string}, \texttt{IsRegex } \texttt{as xs:boolean} ) \texttt{as xs:string} \ XQ3.1}

Converts the input string \texttt{InputString} to camel case by using \texttt{SplitChars} to determine the character/s that trigger the next capitalization. \texttt{SplitChars} is used as a regular expression when \texttt{IsRegex = true()}, or as plain characters when \texttt{IsRegex = false()}. The first character in the output string is capitalized.

\textbf{Examples}

- altova:camel-case("setname getname", "set|get", true()) returns setName
  getName
- altova:camel-case("altova\documents\testcases", ",", false()) returns Altova\Documents\Testcases

\texttt{char [altova:]}\n
\texttt{altova:char(\texttt{Position} \texttt{as xs:integer} ) \texttt{as xs:string} \ XQ3.1}

Returns a string containing the character at the position specified by the \texttt{Position} argument, in the string obtained by converting the value of the context item to \texttt{xs:string}. The result string will be empty if no character exists at the index submitted by the \texttt{Position} argument.

\textbf{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.

\texttt{altova:char(\texttt{InputString} \texttt{as xs:string}, \texttt{Position} \texttt{as xs:integer}) \texttt{as xs:string} \ XQ3.1}

Returns a string containing the character at the position specified by the \texttt{Position} argument, in the string submitted as the \texttt{InputString} argument. The result string will be empty if no character exists at the index submitted by the \texttt{Position} argument.

\textbf{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.

\texttt{create-hash-from-string[altova:]}\n
\texttt{altova:create-hash-from-string(\texttt{InputString} \texttt{as xs:string}) \texttt{as xs:string} \ XQ1 XQ3.1}

\texttt{altova:create-hash-from-string(\texttt{InputString} \texttt{as xs:string}, \texttt{HashAlgo} \texttt{as xs:string}) \texttt{as xs:string} \ XQ1 XQ3.1 XQ3.1}
Generates a hash string from \texttt{InputString} by using the hashing algorithm specified by the \texttt{HashAlgo} argument. The following hashing algorithms may be specified (in upper or lower case): \texttt{MD5}, \texttt{SHA-1}, \texttt{SHA-224}, \texttt{SHA-256}, \texttt{SHA-384}, \texttt{SHA-512}. If the second argument is not specified (see the first signature above), then the \texttt{SHA-256} hashing algorithm is used.

\textbf{Examples}

- \texttt{altova:create-hash-from-string('abc')} returns a hash string generated by using the \texttt{SHA-256} hashing algorithm.
- \texttt{altova:create-hash-from-string('abc', 'md5')} returns a hash string generated by using the \texttt{MD5} hashing algorithm.
- \texttt{altova:create-hash-from-string('abc', 'MD5')} returns a hash string generated by using the \texttt{MD5} hashing algorithm.

\textbf{first-chars [altova:]}

\texttt{altova:first-chars(X-Number as xs:integer) as xs:string} \texttt{XP3.1 XQ3.1}

Returns a string containing the first \texttt{X-Number} of characters of the string obtained by converting the value of the context item to \texttt{xs:string}.

\textbf{Examples}

If the context item is 1234ABCD:

- \texttt{altova:first-chars(2)} returns 12
- \texttt{altova:first-chars(5)} returns 1234A
- \texttt{altova:first-chars(9)} returns 1234ABCD

\texttt{altova:first-chars(InputString as xs:string, X-Number as xs:integer) as xs:string} \texttt{XP3.1 XQ3.1}

Returns a string containing the first \texttt{X-Number} of characters of the string submitted as the \texttt{InputString} argument.

\textbf{Examples}

- \texttt{altova:first-chars("2014-01-15", 5)} returns 2014-
- \texttt{altova:first-chars("USA", 1)} returns U

\textbf{format-string [altova:]}

\texttt{altova:format-string(InputString as xs:string, FormatSequence as item()) as xs:string} \texttt{XP3.1 XQ3.1}

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.

\textbf{Examples}

- \texttt{altova:format-string('Hello \%1, \%2, \%3', ('Jane', 'John', 'Joe'))} returns "Hello Jane, John, Joe"
- \texttt{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:]`

#### `altova:last-chars(X-Number as xs:integer) as xs:string XP3.1 XQ3.1`

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

- `altova:last-chars(2)` returns CD
- `altova:last-chars(5)` returns 4ABCD
- `altova:last-chars(9)` returns 1234ABCD

#### `altova:last-chars(InputString as xs:string, X-Number as xs:integer) as xs:string XP3.1 XQ3.1`

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:]`

#### `altova:pad-string-left(StringToPad as xs:string, StringLength as xs:integer, PadCharacter as xs:string) as xs:string XP3.1 XQ3.1`

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:]`

#### `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:]

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:]

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 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('ABCD-ABCD', 'B')` returns 'CD'
- `altova:substring-after-last('ABCD-ABCD-ABCD', 'BCD')` returns ''

substring-before-last [altova:]

altova:substring-before-last(MainString as xs:string, CheckString as xs:string) as 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('ABCD-ABCD', 'B')` returns `'ABCD-A'`
- `altova:substring-before-last('ABCD-ABCD-ABCD', 'ABCD')` returns `'ABCD-

### substring-pos [altova:]

```
altova:substring-pos(StringToCheck as xs:string, StringToFind as xs:string) as 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', 'tov')` returns `3`
- `altova:substring-pos('Altova', 'tv')` returns `0`
- `altova:substring-pos('AltovaAltova', 'to', 1)` returns `3`
- `altova:substring-pos('Altova-Altova', 'to', 4)` returns `10`

### substring-pos [altova:]

#### 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
```

Returns the string without any leading or trailing whitespace. This function is useful for trimming whitespace from strings that are expected to have leading or trailing whitespace.
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:]**

```xml
altova:trim-string-left(InputString as xs:string) as xs:string XP3.1 XQ3.1
```

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:]**

```xml
altova:trim-string-right(InputString as xs:string) as xs:string XP3.1 XQ3.1
```

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"
- `altova:trim-string-right(" Hello World")` returns " Hello World"
- `altova:trim-string-right("Hello World")` returns "Hello World"
- `altova:trim-string-right("Hello World")` returns "Hello World"

### 2.1.8 XPath/XQuery Functions: Miscellaneous

The following general purpose XPath/XQuery extension functions are supported in the current version of XMLSpy 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>
<tr>
<td>XQuery functions (used in XQuery expressions in XQuery):</td>
<td>XQ1 XQ3.1</td>
</tr>
</tbody>
</table>

**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

\[ \text{sleep[altova:]} \]

\texttt{altova:sleep(Millisecs \texttt{as xs:integer}) \texttt{as empty-sequence()}}

Suspends execution of the current operation for the number of milliseconds given by the Millisecs argument.

\textbf{Examples}

- \texttt{altova:sleep(1000)} suspends execution of the current operation for 1000 milliseconds.
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.

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
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.

```xml
    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:

```xml
<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.

**User-Defined Class Files**

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`.

```
<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:template match="/"
   <a>
      <xsl:value-of select="car:getVehicleType()"/>
   </a>
</xsl:template>
</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`.

```
<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.

```
<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')" />
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:

```
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.

```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?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.

**User-Defined Jar Files**

If access is via a JAR file, the URI of the JAR file must be specified using the following syntax:

```
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:

xmlns:ns1="java:docx.layout.pages?path=jar:file:///c:/projects/docs/docx.jar!/
ns1:main()

xmlns:ns2="java?path=jar:file:///c:/projects/docs/docx.jar!/
ns2:docx.layout.pages.main()

Here is a complete XSLT example that uses a JAR file to call a Java extension function:

<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.

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:
  `<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):
  `<xsl:value-of select="date:toString(date:new())"
  xmlns:date="java:java.util.Date" />

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 \( e \) and \( \pi \), 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)" />
  select="jMath:cos( jMath:PI() )" />
  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>
```
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 stylesheet 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.

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>Datatype</th>
<th>Corresponding Java Types</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 (primitive), java.lang.Boolean</code></td>
</tr>
<tr>
<td><code>xs:integer</code></td>
<td><code>int, long, short, byte, float, double, and the wrapper classes of these, such as java.lang.Integer</code></td>
</tr>
<tr>
<td><code>xs:float</code></td>
<td><code>float (primitive), java.lang.Float, double (primitive)</code></td>
</tr>
<tr>
<td><code>xs:double</code></td>
<td><code>double (primitive), java.lang.Double</code></td>
</tr>
<tr>
<td><code>xs:decimal</code></td>
<td><code>float (primitive), java.lang.Float, double (primitive), 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).

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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asm</td>
<td>The name of the assembly to be loaded.</td>
</tr>
<tr>
<td>ver</td>
<td>The version number (maximum of four integers separated by periods).</td>
</tr>
<tr>
<td>sn</td>
<td>The key token of the assembly's strong name (16 hex digits).</td>
</tr>
<tr>
<td>from</td>
<td>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.</td>
</tr>
<tr>
<td>partialname</td>
<td>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.</td>
</tr>
<tr>
<td>loc</td>
<td>The locale, for example, en-US. The default is neutral.</td>
</tr>
</tbody>
</table>
If the assembly is to be loaded from a DLL, use the \texttt{from} parameter and omit the \texttt{sn} parameter. If the assembly is to be loaded from the Global Assembly Cache (GAC), use the \texttt{sn} parameter and omit the \texttt{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 \texttt{System.Environment}:

```xml
xmlns:myns="clitype:System.Environment"
```

An example of a namespace declaration in XSLT that identifies the class to be loaded as \texttt{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 \texttt{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=b9f091b72dcdcfba8";
   ```

2. When the assembly is loaded from the DLL (complete and partial references below):
   ```xml
   declare namespace cs="clitype:MyManagedDLL.testClass?fom=file:///C:/Altova/Projects/extFunctions/MyManagedDLL.dll;
   declare namespace cs="clitype:MyManagedDLL.testClass?fom=MyManagedDLL.dll;
   ```

**XSLT example**

Here is a complete XSLT example that calls functions in system class \texttt{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">
            <sqrt><xsl:value-of select="math:Sqrt(9)"/></sqrt>
            <pi><xsl:value-of select="math:PI()"/></pi>
            <e><xsl:value-of select="math:E()"/></e>
            <pow><xsl:value-of select="math:Pow(math:PI(), math:E())"/></pow>
        </math>
    </xsl:template>
</xsl:stylesheet>
```
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:

```
<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.

**.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.ToDateTime()`, 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" />
  ```
.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>

.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:

```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="/">
    <xsl:variable name="releasedate"
      select="date:new(2008, 4, 29)"
      xmlns:date="clitype:System.DateTime"/>
    <doc>
      <xsl:value-of select="date:ToString(date:new(2008, 4, 29))"
        xmlns:date="clitype:System.DateTime"/>
    </doc>
  </xsl:template>
</xsl:stylesheet>
```
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.

**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.
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.

### 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 datatype 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).

#### 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`.
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.

```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: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: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.

```xml
<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.
3 Datatypes in DB-Generated XML Schemas

When an XML Schema is generated from a database (DB), the datatypes specific to that DB are converted to XML Schema datatypes. The mappings of DB datatypes to XML Schema datatypes for commonly used DBs are given in this Appendix. Select from the list below.

- ADO
- MS Access
- MS SQL Server
- MySQL
- ODBC
- Oracle
- Sybase
### 3.1 ADO

When an XML Schema is generated from an ADO database (DB), the ADO DB datatypes are converted to XML Schema datatypes as listed in the table below.

<table>
<thead>
<tr>
<th>ADO Datatype</th>
<th>XML Schema Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>adGUID</td>
<td>xs:ID</td>
</tr>
<tr>
<td>adChar</td>
<td>xs:string</td>
</tr>
<tr>
<td>adWChar</td>
<td>xs:string</td>
</tr>
<tr>
<td>adVarChar</td>
<td>xs:string</td>
</tr>
<tr>
<td>adWVarChar</td>
<td>xs:string</td>
</tr>
<tr>
<td>adLongVarChar</td>
<td>xs:string</td>
</tr>
<tr>
<td>adWLongVarChar</td>
<td>xs:string</td>
</tr>
<tr>
<td>adVarChar</td>
<td>xs:string</td>
</tr>
<tr>
<td>adBoolean</td>
<td>xs:boolean</td>
</tr>
<tr>
<td>adSingle</td>
<td>xs:float</td>
</tr>
<tr>
<td>adDouble</td>
<td>xs:double</td>
</tr>
<tr>
<td>adNumeric</td>
<td>xs:decimal</td>
</tr>
<tr>
<td>adCurrency</td>
<td>xs:decimal</td>
</tr>
<tr>
<td>adDBTimeStamp</td>
<td>xs:dateTime</td>
</tr>
<tr>
<td>adDate</td>
<td>xs:date</td>
</tr>
<tr>
<td>adBinary</td>
<td>xs:base64Binary</td>
</tr>
<tr>
<td>adVarBinary</td>
<td>xs:base64Binary</td>
</tr>
<tr>
<td>adLongVarBinary</td>
<td>xs:base64Binary</td>
</tr>
<tr>
<td>adInteger</td>
<td>xs:Integer</td>
</tr>
<tr>
<td>adUnsignedInt</td>
<td>xs:unsignedInt</td>
</tr>
<tr>
<td>adSmallInt</td>
<td>xs:short</td>
</tr>
<tr>
<td>adUnsignedSmallInt</td>
<td>xs:unsignedShort</td>
</tr>
<tr>
<td>adBigInt</td>
<td>xs:long</td>
</tr>
<tr>
<td>adUnsignedBigInt</td>
<td>xs:unsignedLong</td>
</tr>
<tr>
<td>adTinyInt</td>
<td>xs:byte</td>
</tr>
<tr>
<td>adUnsignedTinyInt</td>
<td>xs:unsignedByte</td>
</tr>
</tbody>
</table>
### 3.2 MS Access

When an XML Schema is generated from an MS Access database (DB), the MS Access DB datatypes are converted to XML Schema datatypes as listed in the table below.

<table>
<thead>
<tr>
<th>MS Access Datatype</th>
<th>XML Schema Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUID</td>
<td>xs:ID</td>
</tr>
<tr>
<td>char</td>
<td>xs:string</td>
</tr>
<tr>
<td>varchar</td>
<td>xs:string</td>
</tr>
<tr>
<td>memo</td>
<td>xs:string</td>
</tr>
<tr>
<td>bit</td>
<td>xs:boolean</td>
</tr>
<tr>
<td>Number(single)</td>
<td>xs:float</td>
</tr>
<tr>
<td>Number(double)</td>
<td>xs:double</td>
</tr>
<tr>
<td>Decimal</td>
<td>xs:decimal</td>
</tr>
<tr>
<td>Currency</td>
<td>xs:decimal</td>
</tr>
<tr>
<td>Date/Time</td>
<td>xs:dateTime</td>
</tr>
<tr>
<td>Number(Long Integer)</td>
<td>xs:integer</td>
</tr>
<tr>
<td>Number(Integer)</td>
<td>xs:short</td>
</tr>
<tr>
<td>Number(Byte)</td>
<td>xs:byte</td>
</tr>
<tr>
<td>OLE Object</td>
<td>xs:base64Binary</td>
</tr>
</tbody>
</table>
### 3.3 MS SQL Server

When an XML Schema is generated from an MS SQL Server database (DB), the MS SQL Server DB datatypes are converted to XML Schema datatypes as listed in the table below.

<table>
<thead>
<tr>
<th>MS SQL Server Datatype</th>
<th>XML Schema Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniqueidentifier</td>
<td>xs:ID</td>
</tr>
<tr>
<td>char</td>
<td>xs:string</td>
</tr>
<tr>
<td>nchar</td>
<td>xs:string</td>
</tr>
<tr>
<td>varchar</td>
<td>xs:string</td>
</tr>
<tr>
<td>nvarchar</td>
<td>xs:string</td>
</tr>
<tr>
<td>text</td>
<td>xs:string</td>
</tr>
<tr>
<td>ntext</td>
<td>xs:string</td>
</tr>
<tr>
<td>sysname</td>
<td>xs:string</td>
</tr>
<tr>
<td>bit</td>
<td>xs:boolean</td>
</tr>
<tr>
<td>real</td>
<td>xs:float</td>
</tr>
<tr>
<td>float</td>
<td>xs:double</td>
</tr>
<tr>
<td>decimal</td>
<td>xs:decimal</td>
</tr>
<tr>
<td>money</td>
<td>xs:decimal</td>
</tr>
<tr>
<td>smallmoney</td>
<td>xs:decimal</td>
</tr>
<tr>
<td>datetime</td>
<td>xs:dateTime</td>
</tr>
<tr>
<td>smalldatetime</td>
<td>xs:dateTime</td>
</tr>
<tr>
<td>binary</td>
<td>xs:base64Binary</td>
</tr>
<tr>
<td>varbinary</td>
<td>xs:base64Binary</td>
</tr>
<tr>
<td>image</td>
<td>xs:base64Binary</td>
</tr>
<tr>
<td>integer</td>
<td>xs:integer</td>
</tr>
<tr>
<td>smallint</td>
<td>xs:short</td>
</tr>
<tr>
<td>bigint</td>
<td>xs:long</td>
</tr>
<tr>
<td>tinyint</td>
<td>xs:byte</td>
</tr>
</tbody>
</table>
## 3.4 MySQL

When an XML Schema is generated from a MySQL database (DB), the MySQL DB datatypes are converted to XML Schema datatypes as listed in the table below.

<table>
<thead>
<tr>
<th>MySQL Datatype</th>
<th>XML Schema Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>xs:string</td>
</tr>
<tr>
<td>varchar</td>
<td>xs:string</td>
</tr>
<tr>
<td>text</td>
<td>xs:string</td>
</tr>
<tr>
<td>tinytext</td>
<td>xs:string</td>
</tr>
<tr>
<td>mediumtext</td>
<td>xs:string</td>
</tr>
<tr>
<td>longtext</td>
<td>xs:string</td>
</tr>
<tr>
<td>tinyint(1)</td>
<td>xs:boolean</td>
</tr>
<tr>
<td>float</td>
<td>xs:float</td>
</tr>
<tr>
<td>double</td>
<td>xs:double</td>
</tr>
<tr>
<td>decimal</td>
<td>xs:decimal</td>
</tr>
<tr>
<td>datetime</td>
<td>xs:dateTime</td>
</tr>
<tr>
<td>blob</td>
<td>xs:base64Binary</td>
</tr>
<tr>
<td>tinyblob</td>
<td>xs:base64Binary</td>
</tr>
<tr>
<td>mediumblob</td>
<td>xs:base64Binary</td>
</tr>
<tr>
<td>longblob</td>
<td>xs:base64Binary</td>
</tr>
<tr>
<td>smallint</td>
<td>xs:short</td>
</tr>
<tr>
<td>bigint</td>
<td>xs:long</td>
</tr>
<tr>
<td>tinyint</td>
<td>xs:byte</td>
</tr>
</tbody>
</table>
When an XML Schema is generated from an ODBC database (DB), the ODBC DB datatypes are converted to XML Schema datatypes as listed in the table below.

<table>
<thead>
<tr>
<th>ODBC Datatype</th>
<th>XML Schema Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_GUID</td>
<td>xs:ID</td>
</tr>
<tr>
<td>SQL_CHAR</td>
<td>xs:string</td>
</tr>
<tr>
<td>SQL_VARCHAR</td>
<td>xs:string</td>
</tr>
<tr>
<td>SQL_LONGVARCHAR</td>
<td>xs:string</td>
</tr>
<tr>
<td>SQL_BIT</td>
<td>xs:boolean</td>
</tr>
<tr>
<td>SQL_REAL</td>
<td>xs:float</td>
</tr>
<tr>
<td>SQL_DOUBLE</td>
<td>xs:double</td>
</tr>
<tr>
<td>SQL_DECIMAL</td>
<td>xs:decimal</td>
</tr>
<tr>
<td>SQL_TIMESTAMP</td>
<td>xs:dateTime</td>
</tr>
<tr>
<td>SQL_DATE</td>
<td>xs:date</td>
</tr>
<tr>
<td>SQL_BINARY</td>
<td>xs:base64Binary</td>
</tr>
<tr>
<td>SQL_VARBINARY</td>
<td>xs:base64Binary</td>
</tr>
<tr>
<td>SQL_LONGVARBINARY</td>
<td>xs:base64Binary</td>
</tr>
<tr>
<td>SQL_INTEGER</td>
<td>xs:integer</td>
</tr>
<tr>
<td>SQL_SMALLINT</td>
<td>xs:short</td>
</tr>
<tr>
<td>SQL_BIGINT</td>
<td>xs:long</td>
</tr>
<tr>
<td>SQL_TINYINT</td>
<td>xs:byte</td>
</tr>
</tbody>
</table>
3.6 Oracle

When an XML Schema is generated from an Oracle database (DB), the Oracle DB datatypes are converted to XML Schema datatypes as listed in the table below.

<table>
<thead>
<tr>
<th>Oracle Datatype</th>
<th>XML Schema Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROWID</td>
<td>xs:ID</td>
</tr>
<tr>
<td>CHAR</td>
<td>xs:string</td>
</tr>
<tr>
<td>NCHAR</td>
<td>xs:string</td>
</tr>
<tr>
<td>VARCHAR2</td>
<td>xs:string</td>
</tr>
<tr>
<td>NVARCHAR2</td>
<td>xs:string</td>
</tr>
<tr>
<td>CLOB</td>
<td>xs:string</td>
</tr>
<tr>
<td>NCLOB</td>
<td>xs:string</td>
</tr>
<tr>
<td>NUMBER (with check constraint applied)*</td>
<td>xs:boolean</td>
</tr>
<tr>
<td>NUMBER</td>
<td>xs:decimal</td>
</tr>
<tr>
<td>FLOAT</td>
<td>xs:double</td>
</tr>
<tr>
<td>DATE</td>
<td>xs:dateTime</td>
</tr>
<tr>
<td>INTERVAL YEAR TO MONTH</td>
<td>xs:gYearMonth</td>
</tr>
<tr>
<td>BLOB</td>
<td>xs:base64Binary</td>
</tr>
</tbody>
</table>

* If a check constraint is applied to a column of datatype NUMBER, and the check constraint checks for the values 0 or 1, then the NUMBER datatype for this column will be converted to an XML Schema datatype of xs:boolean. This mechanism is useful for generating an xs:boolean datatype in the generated XML Schema.
3.7 **Sybase**

When an XML Schema is generated from a Sybase database (DB), the Sybase DB datatypes are converted to XML Schema datatypes as listed in the table below.

<table>
<thead>
<tr>
<th>Sybase Datatype</th>
<th>XML Schema Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>xs:string</td>
</tr>
<tr>
<td>nchar</td>
<td>xs:string</td>
</tr>
<tr>
<td>varchar</td>
<td>xs:string</td>
</tr>
<tr>
<td>nvarchar</td>
<td>xs:string</td>
</tr>
<tr>
<td>text</td>
<td>xs:string</td>
</tr>
<tr>
<td>sysname-varchar(30)</td>
<td>xs:string</td>
</tr>
<tr>
<td>bit</td>
<td>xs:boolean</td>
</tr>
<tr>
<td>real</td>
<td>xs:float</td>
</tr>
<tr>
<td>float</td>
<td>xs:float</td>
</tr>
<tr>
<td>double</td>
<td>xs:double</td>
</tr>
<tr>
<td>decimal</td>
<td>xs:decimal</td>
</tr>
<tr>
<td>money</td>
<td>xs:decimal</td>
</tr>
<tr>
<td>smallmoney</td>
<td>xs:decimal</td>
</tr>
<tr>
<td>datetime</td>
<td>xs:dateTime</td>
</tr>
<tr>
<td>smalldatetime</td>
<td>xs:dateTime</td>
</tr>
<tr>
<td>timestamp</td>
<td>xs:dateTime</td>
</tr>
<tr>
<td>binary&lt;=255</td>
<td>xs:base64Binary</td>
</tr>
<tr>
<td>varbinary&lt;=255</td>
<td>xs:base64Binary</td>
</tr>
<tr>
<td>image</td>
<td>xs:base64Binary</td>
</tr>
<tr>
<td>integer</td>
<td>xs:integer</td>
</tr>
<tr>
<td>smallint</td>
<td>xs:short</td>
</tr>
<tr>
<td>tinyint</td>
<td>xs:byte</td>
</tr>
</tbody>
</table>
4 Datatypes in DBs Generated from XML Schemas

When a DB structure is created from an XML Schema, the datatypes specific to that DB are generated from XML Schema datatypes. The mappings of XML Schema datatypes to DB datatypes for commonly used DBs are given in this Appendix. Select from the list below.

- MS Access
- MS SQL Server
- MySQL
- Oracle
### 4.1 MS Access

When an MS Access database (DB) is created from an XML Schema, the XML Schema datatypes are converted to MS Access datatypes as listed in the table below.

<table>
<thead>
<tr>
<th>XML Schema Datatype</th>
<th>MS Access Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>xs:ID</td>
<td>GUID</td>
</tr>
<tr>
<td>xs:string</td>
<td>If no facets</td>
</tr>
<tr>
<td></td>
<td>Size = either</td>
</tr>
<tr>
<td></td>
<td>If Size &lt;= 255</td>
</tr>
<tr>
<td></td>
<td>else memo</td>
</tr>
<tr>
<td>xs:normalizedString</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:token</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:Name</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:NCName</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:anyURI</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:QName</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:NOTATION</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:boolean</td>
<td>bit</td>
</tr>
<tr>
<td>xs:float</td>
<td>Number (single)</td>
</tr>
<tr>
<td>xs:double</td>
<td>Number (double)</td>
</tr>
<tr>
<td>xs:decimal</td>
<td>Decimal</td>
</tr>
<tr>
<td>xs:duration</td>
<td>Date/Time</td>
</tr>
<tr>
<td>xs:dateTime</td>
<td>Date/Time</td>
</tr>
<tr>
<td>xs:time</td>
<td>Date/Time</td>
</tr>
<tr>
<td>xs:date</td>
<td>Date/Time</td>
</tr>
<tr>
<td>xs:gYearMonth</td>
<td>Date/Time</td>
</tr>
<tr>
<td>xs:gYear</td>
<td>Date/Time</td>
</tr>
<tr>
<td>xs:gMonthDay</td>
<td>Date/Time</td>
</tr>
<tr>
<td>xs:gDay</td>
<td>Date/Time</td>
</tr>
<tr>
<td>xs:gMonth</td>
<td>Date/Time</td>
</tr>
<tr>
<td>xs:hexBinary</td>
<td>If no facets</td>
</tr>
<tr>
<td></td>
<td>Size = either</td>
</tr>
<tr>
<td></td>
<td>If Size &lt;= 8000</td>
</tr>
<tr>
<td></td>
<td>else image (OLE Object)</td>
</tr>
<tr>
<td>Datatype</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>xs:base64Binary</td>
<td>Same as xs:hexBinary</td>
</tr>
<tr>
<td>xs:integer</td>
<td>Number (Long Integer)</td>
</tr>
<tr>
<td>xs:int</td>
<td>Number (Long Integer)</td>
</tr>
<tr>
<td>xs:negativeInteger</td>
<td>Number (Long Integer); value constraint</td>
</tr>
<tr>
<td>xs:positiveInteger</td>
<td>Number (Long Integer); value constraint</td>
</tr>
<tr>
<td>xs:nonNegativeInteger</td>
<td>Number (Long Integer); value constraint</td>
</tr>
<tr>
<td>xs:nonPositiveInteger</td>
<td>Number (Long Integer); value constraint</td>
</tr>
<tr>
<td>xs:unsignedInt</td>
<td>Number (Long Integer)</td>
</tr>
<tr>
<td>xs:short</td>
<td>-- no equivalent --</td>
</tr>
<tr>
<td>xs:unsignedShort</td>
<td>-- no equivalent --</td>
</tr>
<tr>
<td>xs:long</td>
<td>-- no equivalent --</td>
</tr>
<tr>
<td>xs:unsignedLong</td>
<td>-- no equivalent --</td>
</tr>
<tr>
<td>xs:byte</td>
<td>Number (Byte)</td>
</tr>
<tr>
<td>xs:unsignedByte</td>
<td>Number (Byte)</td>
</tr>
</tbody>
</table>
## 4.2 MS SQL Server

When an XML Schema is generated from an MS SQL Server database (DB), the MS SQL Server DB datatypes are converted to XML Schema datatypes as listed in the table below.

<table>
<thead>
<tr>
<th>XML Schema Datatype</th>
<th>MS SQL Server Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>uniqueidentifier</td>
</tr>
<tr>
<td>xs:string</td>
<td>If no facets</td>
</tr>
<tr>
<td></td>
<td>{ if UNICODE nvarchar (255)</td>
</tr>
<tr>
<td></td>
<td>else varchar (255)</td>
</tr>
<tr>
<td></td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>{ if UNICODE</td>
</tr>
<tr>
<td></td>
<td>(Size = either length or maxLength)</td>
</tr>
<tr>
<td></td>
<td>if Size &lt;= 4000</td>
</tr>
<tr>
<td></td>
<td>if FacetLengthIsSet then nChar</td>
</tr>
<tr>
<td></td>
<td>else nVarChar</td>
</tr>
<tr>
<td></td>
<td>if Size &lt;= 1073741823 then nText</td>
</tr>
<tr>
<td></td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>{ if NON-UNICODE</td>
</tr>
<tr>
<td></td>
<td>(Size = either length or maxLength)</td>
</tr>
<tr>
<td></td>
<td>if Size &lt;= 8000</td>
</tr>
<tr>
<td></td>
<td>if FacetLengthIsSet then char</td>
</tr>
<tr>
<td></td>
<td>else varchar</td>
</tr>
<tr>
<td></td>
<td>if Size &lt;= 2147483647 then text</td>
</tr>
<tr>
<td>xs:normalizedString</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:token</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:Name</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:NCName</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:anyURI</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:QName</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:NOTATION</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:boolean</td>
<td>bit</td>
</tr>
<tr>
<td>xs:float</td>
<td>real</td>
</tr>
<tr>
<td>xs:double</td>
<td>float</td>
</tr>
<tr>
<td>xs:decimal</td>
<td>decimal</td>
</tr>
</tbody>
</table>
### Datatypes in DBs Generated from XML Schemas

<table>
<thead>
<tr>
<th>XML Datatype</th>
<th>SQL Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>xs:duration</td>
<td>datetime</td>
</tr>
<tr>
<td>xs:dateTime</td>
<td>datetime</td>
</tr>
<tr>
<td>xs:time</td>
<td>datetime</td>
</tr>
<tr>
<td>xs:date</td>
<td>datetime</td>
</tr>
<tr>
<td>xs:gYearMonth</td>
<td>datetime</td>
</tr>
<tr>
<td>xs:gYear</td>
<td>datetime</td>
</tr>
<tr>
<td>xs:gMonthDay</td>
<td>datetime</td>
</tr>
<tr>
<td>xs:gDay</td>
<td>datetime</td>
</tr>
<tr>
<td>xs:gMonth</td>
<td>datetime</td>
</tr>
<tr>
<td>xs:hexBinary</td>
<td>If no facets varbinary (255)</td>
</tr>
<tr>
<td></td>
<td>(Size = either length or maxLength)</td>
</tr>
<tr>
<td></td>
<td>If Size &lt;= 8000</td>
</tr>
<tr>
<td></td>
<td>if FacetLengthIsSet then binary</td>
</tr>
<tr>
<td></td>
<td>else varbinary</td>
</tr>
<tr>
<td></td>
<td>if Size &lt;= 2147483647 then image</td>
</tr>
<tr>
<td>xs:base64Binary</td>
<td>Same as xs:hexBinary</td>
</tr>
<tr>
<td>xs:integer</td>
<td>int</td>
</tr>
<tr>
<td>xs:int</td>
<td>int</td>
</tr>
<tr>
<td>xs:negativeInteger</td>
<td>Int (constrained to {...,-2,-1})</td>
</tr>
<tr>
<td>xs:positiveInteger</td>
<td>Int (constrained to {1,2,...})</td>
</tr>
<tr>
<td>xs:nonNegativeInteger</td>
<td>int (constrained to {0,1,2,...})</td>
</tr>
<tr>
<td>xs:nonPositiveInteger</td>
<td>int (constrained to {...,-2,-1,0})</td>
</tr>
<tr>
<td>xs:unsignedInt</td>
<td>int (additional constraints)</td>
</tr>
<tr>
<td>xs:short</td>
<td>smallint</td>
</tr>
<tr>
<td>xs:unsignedShort</td>
<td>smallint (additional constraints)</td>
</tr>
<tr>
<td>xs:long</td>
<td>bigint</td>
</tr>
<tr>
<td>xs:unsignedLong</td>
<td>bigint (additional constraints)</td>
</tr>
<tr>
<td>xs:byte</td>
<td>tinyint</td>
</tr>
<tr>
<td>xs:unsignedByte</td>
<td>tinyint (additional constraints)</td>
</tr>
</tbody>
</table>
### 4.3 MySQL

When an XML Schema is generated from a MySQL database (DB), the MySQL DB datatypes are converted to XML Schema datatypes as listed in the table below.

<table>
<thead>
<tr>
<th>XML Schema Datatype</th>
<th>MySQL Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xs:ID</code></td>
<td><code>varchar(255)</code></td>
</tr>
<tr>
<td><code>xs:string</code></td>
<td>If no facets then <code>varchar (255)</code></td>
</tr>
<tr>
<td></td>
<td>else if facet length is set and &lt;= 255 then <code>char</code></td>
</tr>
<tr>
<td></td>
<td>else if facet maxLength set and &lt;= 255 then <code>varchar</code></td>
</tr>
<tr>
<td></td>
<td>else if maxLength is set and &lt;= 65545 then <code>text</code></td>
</tr>
<tr>
<td></td>
<td>else if maxlength is set and &lt;= 16777215 then <code>mediumtext</code></td>
</tr>
<tr>
<td></td>
<td>else if maxlength is set and &lt;= 429496295 then <code>longtext</code></td>
</tr>
<tr>
<td><code>xs:normalizedString</code></td>
<td>Same as <code>xs:string</code></td>
</tr>
<tr>
<td><code>xs:token</code></td>
<td>Same as <code>xs:string</code></td>
</tr>
<tr>
<td><code>xs:Name</code></td>
<td>Same as <code>xs:string</code></td>
</tr>
<tr>
<td><code>xs:NCName</code></td>
<td>Same as <code>xs:string</code></td>
</tr>
<tr>
<td><code>xs:anyURI</code></td>
<td>Same as <code>xs:string</code></td>
</tr>
<tr>
<td><code>xs:QName</code></td>
<td>Same as <code>xs:string</code></td>
</tr>
<tr>
<td><code>xs:NOTATION</code></td>
<td>Same as <code>xs:string</code></td>
</tr>
<tr>
<td><code>xs:boolean</code></td>
<td><code>tinyint(1)</code></td>
</tr>
<tr>
<td><code>xs:float</code></td>
<td><code>float</code></td>
</tr>
<tr>
<td><code>xs:double</code></td>
<td><code>double</code></td>
</tr>
<tr>
<td><code>xs:decimal</code></td>
<td><code>decimal</code></td>
</tr>
<tr>
<td><code>xs:duration</code></td>
<td><code>timestamp</code></td>
</tr>
<tr>
<td><code>xs:dateTime</code></td>
<td><code>datetime</code></td>
</tr>
<tr>
<td><code>xs:time</code></td>
<td><code>time</code></td>
</tr>
<tr>
<td><code>xs:date</code></td>
<td><code>date</code></td>
</tr>
<tr>
<td><code>xs:gYearMonth</code></td>
<td><code>timestamp(4)</code></td>
</tr>
<tr>
<td><code>xs:gYear</code></td>
<td><code>year(4)</code></td>
</tr>
<tr>
<td><code>xs:gMonthDay</code></td>
<td><code>timestamp(8); constraints to check month, day</code></td>
</tr>
<tr>
<td><code>xs:gDay</code></td>
<td><code>timestamp(8); constraints to check day</code></td>
</tr>
<tr>
<td>XSD Datatype</td>
<td>MySQL Datatype</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>xs:gMonth</td>
<td>timestamp(8)</td>
</tr>
<tr>
<td>xs:hexBinary</td>
<td>If no facets then blob (255)</td>
</tr>
<tr>
<td></td>
<td>else if facet length is set and &lt;= 255 then blob</td>
</tr>
<tr>
<td></td>
<td>else if facet maxLength is set and &lt;= 255 then tinyblob</td>
</tr>
<tr>
<td></td>
<td>else if maxLength is set and &lt;= 65545 then blob</td>
</tr>
<tr>
<td></td>
<td>else if maxLength is set and &lt;= 16777215 then mediumblob</td>
</tr>
<tr>
<td></td>
<td>else if maxLength is set and &lt;= 429496295 then longblob</td>
</tr>
<tr>
<td>xs:base64Binary</td>
<td>Same as xs:hexBinary</td>
</tr>
<tr>
<td>xs:integer</td>
<td>Integer</td>
</tr>
<tr>
<td>xs:int</td>
<td>int</td>
</tr>
<tr>
<td>xs:negativeInteger</td>
<td>Integer (constrained to {...,-2,-1})</td>
</tr>
<tr>
<td>xs:positiveInteger</td>
<td>Integer (constrained to {1,2,...})</td>
</tr>
<tr>
<td>xs:nonNegativeInteger</td>
<td>Integer (constrained to {0,1,2,...})</td>
</tr>
<tr>
<td>xs:nonPositiveInteger</td>
<td>Integer (constrained to {...,-2,-1,0})</td>
</tr>
<tr>
<td>xs:unsignedInt</td>
<td>Int (additional constraints)</td>
</tr>
<tr>
<td>xs:short</td>
<td>Smallint</td>
</tr>
<tr>
<td>xs:unsignedShort</td>
<td>Smallint (additional constraints)</td>
</tr>
<tr>
<td>xs:long</td>
<td>Bignum</td>
</tr>
<tr>
<td>xs:unsignedLong</td>
<td>Bignum (additional constraints)</td>
</tr>
<tr>
<td>xs:byte</td>
<td>Tinyint</td>
</tr>
<tr>
<td>xs:unsignedByte</td>
<td>Tinyint (additional constraints)</td>
</tr>
</tbody>
</table>
4.4 **Oracle**

When an XML Schema is generated from an Oracle database (DB), the Oracle DB datatypes are converted to XML Schema datatypes as listed in the table below.

<table>
<thead>
<tr>
<th>XML Schema Datatype</th>
<th>Oracle Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>xs:ID</td>
<td>ROWID</td>
</tr>
<tr>
<td>xs:string</td>
<td>If no facets</td>
</tr>
<tr>
<td></td>
<td>if UNICODE then NVARCHAR2 (255)</td>
</tr>
<tr>
<td></td>
<td>else VARCHAR2 (255)</td>
</tr>
<tr>
<td></td>
<td>else if UNICODE</td>
</tr>
<tr>
<td></td>
<td>(Size = either length or maxLength)</td>
</tr>
<tr>
<td></td>
<td>If Size &lt;= 2000 then NCHAR</td>
</tr>
<tr>
<td></td>
<td>if Size &lt;= 4000 then NVARCHAR2</td>
</tr>
<tr>
<td></td>
<td>if Size &lt;= 4 Gigabytes then NCLOB</td>
</tr>
<tr>
<td></td>
<td>else if NON-UNICODE</td>
</tr>
<tr>
<td></td>
<td>(Size = either length or maxLength)</td>
</tr>
<tr>
<td></td>
<td>If Size &lt;= 2000 then CHAR</td>
</tr>
<tr>
<td></td>
<td>if Size &lt;= 4000 then VARCHAR2</td>
</tr>
<tr>
<td></td>
<td>if Size &lt;= 4 Gigabytes then CLOB</td>
</tr>
<tr>
<td>xs:normalizedString</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:token</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:Name</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:NCName</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:anyURI</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:QName</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:NOTATION</td>
<td>Same as xs:string</td>
</tr>
<tr>
<td>xs:boolean</td>
<td>NUMBER with constraint Boolean</td>
</tr>
<tr>
<td>xs:float</td>
<td>FLOAT</td>
</tr>
<tr>
<td>xs:double</td>
<td>FLOAT</td>
</tr>
<tr>
<td>xs:decimal</td>
<td>NUMBER</td>
</tr>
<tr>
<td>xs:duration</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>xs:dateTime</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>xs:time</td>
<td>DATE</td>
</tr>
<tr>
<td>xs:date</td>
<td>DATE</td>
</tr>
</tbody>
</table>
### Datatypes in DBs Generated from XML Schemas

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xs:gYearMonth</td>
<td>INTERVAL YEAR TO MONTH</td>
</tr>
<tr>
<td>xs:gYear</td>
<td>DATE</td>
</tr>
<tr>
<td>xs:gMonthDay</td>
<td>DATE</td>
</tr>
<tr>
<td>xs:gDay</td>
<td>DATE</td>
</tr>
<tr>
<td>xs:gMonth</td>
<td>DATE</td>
</tr>
<tr>
<td>xs:hexBinary</td>
<td>if no facets then RAW (255)</td>
</tr>
<tr>
<td></td>
<td>(Size = either length or maxLength)</td>
</tr>
<tr>
<td></td>
<td>If Size &lt;= 2000 then RAW (X)</td>
</tr>
<tr>
<td></td>
<td>else Size &lt;= 2 Gigabytes then LONG RAW (X)</td>
</tr>
<tr>
<td></td>
<td>if Size &lt;= 4 Gigabytes then BLOB (X)</td>
</tr>
<tr>
<td>xs:base64Binary</td>
<td>BLOB</td>
</tr>
<tr>
<td>xs:integer</td>
<td>NUMBER</td>
</tr>
<tr>
<td>xs:int</td>
<td>NUMBER</td>
</tr>
<tr>
<td>xs:negativeInteger</td>
<td>NUMBER (constrained to {...,-2,-1})</td>
</tr>
<tr>
<td>xs:positiveInteger</td>
<td>NUMBER (constrained to {1,2,...})</td>
</tr>
<tr>
<td>xs:nonNegativeInteger</td>
<td>NUMBER (constrained to {0,1,2,...})</td>
</tr>
<tr>
<td>xs:nonPositiveInteger</td>
<td>NUMBER (constrained to {...,-2,-1,0})</td>
</tr>
<tr>
<td>xs:unsignedInt</td>
<td>NUMBER (additional constraints)</td>
</tr>
<tr>
<td>xs:short</td>
<td>NUMBER</td>
</tr>
<tr>
<td>xs:unsignedShort</td>
<td>NUMBER (additional constraints)</td>
</tr>
<tr>
<td>xs:long</td>
<td>NUMBER</td>
</tr>
<tr>
<td>xs:unsignedLong</td>
<td>NUMBER (additional constraints)</td>
</tr>
<tr>
<td>xs:byte</td>
<td>BLOB</td>
</tr>
<tr>
<td>xs:unsignedByte</td>
<td>BLOB (additional constraints)</td>
</tr>
</tbody>
</table>
5 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
5.1 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.
5.2 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.
5.3 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.
5.4 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.
6 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.
6.1 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.
6.2 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.
6.3 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
Index

...
Application,
  Quit, 1110
  ReloadSettings, 1110
  RemoveXSLT_XQParameter, 1110
  RunMacro, 1111
  ScriptingEnvironment, 1111
  ShowApplication, 1111
  ShowForm, 1112
  URLDelete, 1113
  URLMakeDirectory, 1113
  WarningNumber, 1114
  WarningText, 1114
Application Events, 1016
Apply, 954
Archive View, 244, 501
  and EPUB files, 509
  and OOXML files, 503
  and ZIP files, 507
ASPRJ files, 998
Assertion messages, 230
Assertions in Schema View, 204, 208
Assertions of simple types, 225
Assign,
  shortcut to a command, 941
Assigning StyleVision Power Stylesheet to XML file, 841
ATL,
  plug-in sample files, 1044
ATTLIST declaration,
  add as child in Grid View, 774
  appending in Grid View, 767
  convert to in Grid View, 776
  inserting in Grid View, 759
Attribute, 46
  add as child in Grid View, 769
  appending in Grid View, 761
  convert to in Grid View, 775
  in schema definitions, 46
  inserting in Grid View, 753
  toggle in Content model view, 46
Attribute groups in Schema View, 205
Attribute preview, 963
Attribute value templates,
  XPath intelligent editing in, 306
Attribute values,
  entering in Authentic View, 378
AttributeFormDefault,
  settings in Schema Design View, 804
Attributes, 175
Attributes entry helper,
  in Authentic View, 387
Attributes in Schema View, 204, 205
Authentic menu, 839
  dynamic table editing, 382
  markup display, 382
Authentic Scripting,
  security settings, 848
  trusted locations, 848
Authentic View, 401
  adding nodes, 373
  applying elements, 373
  CDATA sections in, 376
  clearing elements, 373
  context menu, 370
  context menus, 392
  data entry devices in, 376
  displaying markup tags, 370
  document display, 385
  editing data in an XML DB, 842
  editing DB data in, 840
  editing XML in, 256
  entering attribute values, 378
  entering data in, 376
  entities in, 376
  entry helpers, 370
  entry helpers in, 387
  formatting text in, 382
  generating output documents from PXF file, 848
  inserting entities in, 378
  inserting nodes, 373
  main window in, 385
  markup display in, 382, 385
  opening an XML document in, 369
  opening new XML file in, 840
  overview of GUI, 381
  paste as XML/Text, 392
  printing an XML document from, 379
  removing nodes, 373
  special characters in, 376
  SPS Tables, 400
  switching to, 905
  tables (SPS and XML), 400
  tables in, 373
  toolbar icons, 382
  usage of important features, 395
  usage of XML tables, 401
  XML table icons, 405

© 2019 Altova GmbH
Index

Authentic View, 401
  XML tables, 401
Authentic View Events, 1016
Authentic View template, 369
Authentic XML, 366
AuthenticDataTransfer,
  dropEffect, 1116
  getData, 1116
  ownDrag, 1117
  type, 1117
AuthenticRange,
  AppendRow, 1121
  Application, 1122
  CanPerformAction, 1122
  CanPerformActionWith, 1123
  Close, 1123
  CollapsToBegin, 1123
  CollapsToEnd, 1124
  Copy, 1124
  Cut, 1124
  Delete, 1125
  DeleteRow, 1125
  DuplicateRow, 1125
  ExpandTo, 1126
  FirstTextPosition, 1127
  FirstXMLData, 1128
  FirstXMLDataOffset, 1128
  GetElementAttributeNames, 1129
  GetElementAttributeValue, 1130
  GetElementHierarchy, 1130
  GetEntityNames, 1131
  Goto, 1132
  GotoNext, 1132
  GotoNextCursorPosition, 1133
  GotoPrevious, 1133
  GotoPreviousCursorPosition, 1134
  HasElementAttribute, 1134
  InsertEntity, 1135
  InsertRow, 1135
  IsCopyEnabled, 1136
  IsCutEnabled, 1136
  IsDeleteEnabled, 1136
  IsEmpty, 1137
  IsEqual, 1137
  IsFirstRow, 1137
  IsInDynamicTable, 1137
  IsLastRow, 1138
  IsPasteEnabled, 1138
  IsTextStateApplied, 1138
  LastTextPosition, 1139
  LastXMLData, 1140
  LastXMLDataOffset, 1140
  MoveBegin, 1141
  MoveEnd, 1142
  MoveRowDown, 1142
  MoveRowUp, 1142
  Parent, 1143
  Paste, 1143
  PerformAction, 1143
  Select, 1144
  SelectNext, 1144
  SelectPrevious, 1145
  SetElementAttributeValue, 1146
  SetFromRange, 1147
  Text, 1148
AuthenticView, 1166
  Application, 1158
  AsXMLString, 1159
  DocumentBegin, 1160
  DocumentEnd, 1160
  Event, 1161
  Goto, 1162
  IsRedoEnabled, 1163
  IsUndoEnabled, 1163
  MarkupVisibility, 1164
  OnBeforeCopy, 1149
  OnBeforeCut, 1150
  OnBeforeDelete, 1150
  OnBeforeDrop, 1151
  OnBeforePaste, 1152
  OnDragOver, 1153
  OnKeyboardEvent, 1154
  OnMouseEvent, 1155
  OnSelectionChanged, 1156
  Parent, 1164
  Print, 1164
  Redo, 1164
  Selection, 1165
  Undo, 1166
  WholeDocument, 1167
  XMLDataRoot, 1167
Auto-complete,
  text view enable/disable, 962
Auto-completion in SQL scripts, 858
Auto-hiding windows, 100
Auto-Macro setting, 1020
Automatic validation, 958
Avro Schema, 431

B
Back,
in Schema View, 240
Background Information, 1559
Base type,
modifying, 233
Big-endian, 961
Bookmark, 985
Bookmark margin, 908
Bookmarks,
inserting and removing, 721
navigating, 721
Bookmarks in SQL scripts, 858
Bookmarks in Text View, 133
Breakpoint,
dialog box, 837
Breakpoints,
using in XSLT/XQuery Debugger, 356
Breakpoints dialog, 344
Browse,
Oracle XML Db, 875
Browser, 963
View, 906
Browser menu, 911
Browser pane,
in Database Query window, 855
Browser View, 911
back, 911
font size, 912
forward, 911
refresh content, 912
separate window, 912
stop loading page, 911

C
C#,
integration of XMLSpy, 1398
Call Stack Window,
in XSLT/XQuery Debugger, 352
Calling named templates, 311
Carriage return key,
see Enter key, 420
Cascading,
Window, 982
Catalog,
Oasis XML, 783
Catalogs, 275
CDATA,
add as child in Grid View, 769
appending in Grid View, 762
convert to in Grid View, 776
inserting in Grid View, 754
CDATA sections,
inserting in Authentic View, 395
Changing view,
to Authentic View, 382
Chapters, 985
Character,
position, 907
Character-Set,
encoding, 961
Check,
spelling checker, 913
Class,
JAVA, 1344
Class ID,
in XMLSpy integration, 1401
ClassPath statement, 1344
Code page, 967
CodeGenDlg,
Application, 1168
CPPSettings_DOMType, 1169
CPPSettings_LibraryType, 1170
CPPSettings_UseMFC, 1170
CSharpSettings_ProjectType, 1170
OutputPath, 1171
OutputPathDialogAction, 1171
OutputResultDialogAction, 1171
Parent, 1171
ProgrammingLanguage, 1172
PropertySheetDialogAction, 1172
TemplateFileName, 1172
Collapse,
unselected, 907
Color, 967
COM-API,
documentation, 1056
<table>
<thead>
<tr>
<th>Command, 945</th>
<th>Connecting to SchemaAgent Server, 281, 821</th>
</tr>
</thead>
<tbody>
<tr>
<td>add to toolbar/menu, 937</td>
<td>Content Model,</td>
</tr>
<tr>
<td>context menu, 945</td>
<td>creating a basic model, 22</td>
</tr>
<tr>
<td>delete from menu, 945</td>
<td>save diagram, 807</td>
</tr>
<tr>
<td>reset menu, 945</td>
<td>toggle attributes, 46</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Commands,</th>
<th>Content Model View, 18</th>
</tr>
</thead>
<tbody>
<tr>
<td>listing in key map, 986</td>
<td>assigning conditional types, 197</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Comment,</th>
<th>Configurations,</th>
</tr>
</thead>
<tbody>
<tr>
<td>add as child in Grid View, 769</td>
<td>of a global resource, 594</td>
</tr>
<tr>
<td>appending in Grid View, 762</td>
<td>Configurations in global resources, 609</td>
</tr>
<tr>
<td>convert to in Grid View, 776</td>
<td>Configure,</td>
</tr>
<tr>
<td>inserting in Grid View, 754</td>
<td>XMLSPY UI, 1041</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Commenting in and out,</th>
<th>Configure view,</th>
</tr>
</thead>
<tbody>
<tr>
<td>in XML documents in Text View., 251</td>
<td>dialog for Content Model View, 813</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Commenting XML text in and out, 722</th>
<th>Copyright information, 1564</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Comments, 172</th>
<th>CR&amp;LF, 955</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Comments in Schema View, 175</th>
<th>Create,</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comments in SQL scripts, 858</td>
<td>DB based on schema, 890</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Comparing directories, 926</th>
<th>Copyright information, 1564</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparing files, 922</td>
<td>CR&amp;LF, 955</td>
</tr>
<tr>
<td>Comparisons, 929</td>
<td>Create,</td>
</tr>
<tr>
<td>Comparisons, 638</td>
<td>Document outline, 426</td>
</tr>
<tr>
<td>of directories, 638</td>
<td>CSS, 423</td>
</tr>
<tr>
<td>of files, 637</td>
<td>auto-completion, 426</td>
</tr>
<tr>
<td>of files and directories, 636</td>
<td>document outline, 426</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Complex type, 35</th>
<th>Copy,</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex types, 35</td>
<td>as XML text, 706</td>
</tr>
<tr>
<td>of a global resource, 594</td>
<td>XML as structured text, 706</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component definition, 35</th>
<th>Copy command, 705</th>
</tr>
</thead>
<tbody>
<tr>
<td>reusing, 35</td>
<td>CSS, 423</td>
</tr>
<tr>
<td>Components entry helper, 220, 225</td>
<td>auto-completion, 426</td>
</tr>
<tr>
<td>Compositor, 22</td>
<td>document outline, 426</td>
</tr>
<tr>
<td>for sequences, 22</td>
<td>CSS, 423</td>
</tr>
<tr>
<td>Compositors in Schema View, 185</td>
<td>auto-completion, 426</td>
</tr>
<tr>
<td>Conditional type alternatives, 197</td>
<td>Configure,</td>
</tr>
<tr>
<td>Conditional type assignments, 197</td>
<td>XML as structured text, 706</td>
</tr>
<tr>
<td>Configurations, 197</td>
<td>Configure view,</td>
</tr>
<tr>
<td>of a global resource, 594</td>
<td>dialog for Content Model View, 813</td>
</tr>
<tr>
<td>Configurations in global resources, 609</td>
<td>Configure view,</td>
</tr>
<tr>
<td>Configure, 1041</td>
<td>dialog for Content Model View, 813</td>
</tr>
<tr>
<td>Configure view, 1041</td>
<td>Configure view,</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Connect to SchemaAgent Server, 281, 821</th>
<th>Configure view,</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure view, dialog for Content Model View, 813</td>
<td>Configure view,</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Convert, 877</th>
<th>Configure view, dialog for Content Model View, 813</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convert menu, 877</td>
<td>Configure view, dialog for Content Model View, 813</td>
</tr>
<tr>
<td>Convert To command, 877</td>
<td>Configure view, dialog for Content Model View, 813</td>
</tr>
<tr>
<td>Create, 351</td>
<td>Configure view, dialog for Content Model View, 813</td>
</tr>
<tr>
<td>CSS, 423</td>
<td>Configure view, dialog for Content Model View, 813</td>
</tr>
<tr>
<td>Copyright information, 1564</td>
<td>Configure view, dialog for Content Model View, 813</td>
</tr>
<tr>
<td>CR&amp;LF, 955</td>
<td>Configure view, dialog for Content Model View, 813</td>
</tr>
<tr>
<td>Create, 351</td>
<td>Configure view, dialog for Content Model View, 813</td>
</tr>
<tr>
<td>CSS, 423</td>
<td>Configure view, dialog for Content Model View, 813</td>
</tr>
<tr>
<td>Copyright information, 1564</td>
<td>Configure view, dialog for Content Model View, 813</td>
</tr>
<tr>
<td>CR&amp;LF, 955</td>
<td>Configure view, dialog for Content Model View, 813</td>
</tr>
</tbody>
</table>
CSS Info window, 426
CSV file,
  import as XML, 877
Custom dictionary, 913
CustomCatalog, 783
Customization, 125
Customize, 945
  context menu, 945
  Customize context menu, 950
  macros, 947
  menu, 945
  toolbar/menu commands, 937
Cut command, 705

D

Database,
  create DB based on schema, 890
  editing records of, 862
  export of XML data to, 896
  import data as XML, 880
  import structure as XML Schema, 885
  Oracle XML Db, 872
Database connection,
  reusing from Global Resources, 545
  setting up, 515
  setup examples, 546
  starting the wizard, 516
Database drivers,
  overview, 518
Database Query,
  Browser pane in DB Query window, 855
  Connecting to DB for query, 852
  creating the query, 861
  Editing results, 862
  Messages pane, 862
  Results of, 862
Database Query window, 850
Database/Table View,
  how to use, 73
DatabaseConnection,
  ADOConnection, 1173
  AsAttributes, 1174
  CreateMissingTables, 1174
  CreateNew, 1175
  DatabaseKind, 1175
  ExcludeKeys, 1175
  File, 1176
  IncludeEmptyElements, 1177
  NumberDateTimeFormat, 1177
  ODBCConnection, 1177
  SQLSelect, 1178
  TextFieldLen, 1179
Databases,
  and global resources, 608
  editing in Authentic View, 840
  see also DB, 408
  support in XMLSpy, 592
Databases in XMLSpy, 513
Date Picker,
  using in Authentic View, 415
Dates,
  changing manually, 416
DB, 408, 409
  creating queries, 409
  editing in Authentic View, 408, 413
  filtering display in Authentic View, 409
  navigating tables in Authentic View, 408
  parameters in DB queries, 409
  queries in Authentic View, 408
DB XML,
  assigning XML Schemas to, for IBM DB2, 868
  managing XML Schemas, for IBM DB2, 865
db2-fn:sqlquery, 336
db2-fn:xmlcolumn, 336
Debugger,
  breakpoints/tracepoints dialog box, 837
  debug windows, 838
  enable/disable breakpoint, 836
  enable/disable tracepoint, 837
  end session, 835
  insert/remove breakpoint, 836
  insert/remove tracepoint, 836
  restart XSLT Debugger, 835
  settings, 838
  show curr. exec. nodes, 836
  start XSLT Debugger, 834
  step into, 835
  step out, 835
  step over, 835
  stop XSLT Debugger, 834
Debugger windows,
  arrangement, 355
Debugging macros, 1024
Default,

coding, 961
menu, 945
Default editor, 958
Default open content models, 201
Default view,
setting in Main Window, 958
defaultOpenContent, 175
Definitions Overview Grid, 459
Delete, 937
  Application.URLDelete, 1113
  command from context menu, 945
  command from toolbar, 937
  icon from toolbar, 937
  row (in Authentic View), 847
  shortcut, 941
  toolbar, 938
Delete command, 705
Derived types,
  modifying base type of, 233
Deriving a schema type, 234
Details entry helper, 22, 223
Dialogs,
  Application, 1180
  CodeGeneratorDlg, 1180
  DTDSchemaGeneratorDlg, 1181
  FileSelectionDlg, 1180
  GenerateSampleXMLDlg, 1181
  Parent, 1180
  SchemaDocumentationDlg, 1181
Dictionary, 913
  adding custom, 913
  modifying existing, 913
  spelling checker, 913
directories,
  comparing two, 926
  creating with Application.URLMakeDirectory, 1113
Directory comparisons, 636, 638
Disable,
  breakpoint - XSLT debugger, 836
  tracepoint - XSLT debugger, 837
Disconnect XMLSpy from SchemaAgent, 821
Display all globals, 817
Display diagram, 817
Distribution,
  of Altova's software products, 1564, 1565
DocEditEvent (obsolete),
  altKey (obsolete), 1285
  altLeft (obsolete), 1286
  button (obsolete), 1287
  cancelBubble (obsolete), 1288
  clientX (obsolete), 1289
  clientY (obsolete), 1290
  ctrlKey (obsolete), 1291
  ctrlLeft (obsolete), 1292
  dataTransfer (obsolete), 1293
  fromElement (obsolete), 1294
  keyCode (obsolete), 1294
  propertyName (obsolete), 1295
  repeat (obsolete), 1295
  returnValue (obsolete), 1295
  shiftKey (obsolete), 1296
  shiftLeft (obsolete), 1297
  srcElement (obsolete), 1298
  type (obsolete), 1299
DocEditView (obsolete),
  ApplyTextState (obsolete), 1305
  CurrentSelection (obsolete), 1306
  EditClear (obsolete), 1306
  EditCopy (obsolete), 1307
  EditCut (obsolete), 1307
  EditPaste (obsolete), 1308
  EditRedo (obsolete), 1308
  EditSelectAll (obsolete), 1309
  EditUndo (obsolete), 1309
  event (obsolete), 1310
  GetAllowedElements (obsolete), 1310
  GetNextVisible (obsolete), 1313
  GetPreviousVisible (obsolete), 1314
  IsEditClearEnabled (obsolete), 1315
  IsEditCopyEnabled (obsolete), 1315
  IsEditCutEnabled (obsolete), 1317
  IsEditPasteEnabled (obsolete), 1317
  IsEditRedoEnabled (obsolete), 1318
  IsEditUndoEnabled (obsolete), 1318
  IsRowAppendEnabled (obsolete), 1319
  IsRowDeleteEnabled (obsolete), 1319
  IsRowDuplicateEnabled (obsolete), 1320
  IsRowInsertEnabled (obsolete), 1320
  IsRowMoveDownEnabled (obsolete), 1321
  IsRowMoveUpEnabled (obsolete), 1321
  IsTextStateApplied (obsolete), 1322
  IsTextStateEnabled (obsolete), 1322
  LoadXML (obsolete), 1323
  RowAppend (obsolete), 1325
  RowDelete (obsolete), 1325
DocEditView (obsolete),
  RowDuplicate (obsolete), 1326
  RowInsert (obsolete), 1326
  RowMoveDown (obsolete), 1327
  RowMoveUp (obsolete), 1327
  SaveXML (obsolete), 1328
  SelectionMoveTabOrder (obsolete), 1329
  SelectionSet (obsolete), 1330
  XMLRoot (obsolete), 1331

Dockable window, 982, 983
Docking windows, 100

DOCTYPE declaration,
  add as child in Grid View, 773
  appending in Grid View, 765
  convert to in Grid View, 776
  inserting in Grid View, 757

Document, 1194
  Application, 1187
  AssignDTD, 1187
  AssignSchema, 1187
  AssignXSL, 1187
  AssignXSLFO, 1188
  AuthenticView, 1188
  browse Oracle XML Db, 875
  Close, 1189
  ConvertDTDOrSchema, 1189
  CreateChild, 1191
  CreateSchemaDiagram, 1192
  CurrentViewMode, 1192
  DataRoot, 1192
  DocEditView, 1193
  Encoding, 1193
  EndChanges, 1194
  ExecuteXQuery, 1194
  ExportToDatabase, 1194
  ExportToText, 1195
  FullName, 1197
  GenerateDTDOrSchema, 1197, 1198
  GenerateProgramCode, 1198
  GenerateSampleXML, 1198
  GenerateSchemaDocumentation, 1199
  GetExportElementList, 1199
  GetPathName, 1200
  GridView, 1200
  IsModified, 1201
  IsValid, 1201
  IsWellFormed, 1202
  Name, 1202
  OnBeforeCloseDocument, 1184
  OnBeforeSaveDocument, 1184
  OnBeforeValidate, 1185
  OnCloseDocument, 1186
  OnViewActivation, 1186
  Path, 1203
  RootElement, 1203
  Save, 1203
  SaveAs, 1204
  Saved, 1204
  SaveInString, 1204
  SaveToURL, 1205
  SetActiveDocument, 1205
  SetEncoding, 1205
  SetExternalValid, 1207
  SetPathName, 1207
  Spelling checker, 913
  StartChanges, 1207
  SwitchViewMode, 1208
  Title, 1208
  TransformXSL, 1209
  TransformXSLFO, 1209
  UpdateViews, 1210
  UpdateXMLData, 1210
  XQuery, 1194

Document Events, 1016

Documentation,
  for schema, 53
  of XML Schema files, 807

Document-level,
  examples of integration of XMLSpy, 1398

Documents,
  Count, 1211
  Item, 1211
  NewAuthenticFile, 1212
  NewFile, 1212
  NewFileFromText, 1212
  OpenAuthenticFile, 1213
  OpenFile, 1213
  OpenURL, 1214
  OpenURLDialog, 1214

Documents in Main Window, 101

DTD,
  assigning to XML document, 790
  Attlist declaration in, 759, 767, 774, 776
  converting to UML, 797
  converting to XML Schema, 794
  Element declaration in, 758, 766, 773, 776

Dockable window, 982, 983
Docking windows, 100

DOCTYPE declaration,
  add as child in Grid View, 773
  appending in Grid View, 765
  convert to in Grid View, 776
  inserting in Grid View, 757

Document, 1194
  Application, 1187
  AssignDTD, 1187
  AssignSchema, 1187
  AssignXSL, 1187
  AssignXSLFO, 1188
  AuthenticView, 1188
  browse Oracle XML Db, 875
  Close, 1189
  ConvertDTDOrSchema, 1189
  CreateChild, 1191
  CreateSchemaDiagram, 1192
  CurrentViewMode, 1192
  DataRoot, 1192
  DocEditView, 1193
  Encoding, 1193
  EndChanges, 1194
  ExecuteXQuery, 1194
  ExportToDatabase, 1194
  ExportToText, 1195
  FullName, 1197
  GenerateDTDOrSchema, 1197, 1198
  GenerateProgramCode, 1198
  GenerateSampleXML, 1198
  GenerateSchemaDocumentation, 1199
  GetExportElementList, 1199
  GetPathName, 1200
  GridView, 1200
  IsModified, 1201
  IsValid, 1201
  IsWellFormed, 1202
  Name, 1202
  OnBeforeCloseDocument, 1184
  OnBeforeSaveDocument, 1184
  OnBeforeValidate, 1185
  OnCloseDocument, 1186
  OnViewActivation, 1186
  Path, 1203
  RootElement, 1203
  Save, 1203
  SaveAs, 1204
  Saved, 1204
  SaveInString, 1204
  SaveToURL, 1205
  SetActiveDocument, 1205
  SetEncoding, 1205
  SetExternalValid, 1207
  SetPathName, 1207
  Spelling checker, 913
  StartChanges, 1207
  SwitchViewMode, 1208
  Title, 1208
  TransformXSL, 1209
  TransformXSLFO, 1209
  UpdateViews, 1210
  UpdateXMLData, 1210
  XQuery, 1194

Document Events, 1016

Documentation,
  for schema, 53
  of XML Schema files, 807

Document-level,
  examples of integration of XMLSpy, 1398

Documents,
  Count, 1211
  Item, 1211
  NewAuthenticFile, 1212
  NewFile, 1212
  NewFileFromText, 1212
  OpenAuthenticFile, 1213
  OpenFile, 1213
  OpenURL, 1214
  OpenURLDialog, 1214

Documents in Main Window, 101

DTD,
  assigning to XML document, 790
  Attlist declaration in, 759, 767, 774, 776
  converting to UML, 797
  converting to XML Schema, 794
  Element declaration in, 758, 766, 773, 776
DTD,
    Entity declaration in, 759, 767, 774, 777
    generate outline XML file from, 799
    generating from XML document, 792
    generating from XML Schema (Enterprise and Professional editions), 270
    go to definition in from XML document, 792
    go to from XML document, 791
    including entities, 791
    menu commands related to, 790
    Notation declaration in, 759, 767, 774, 777
    reference from XML to external DTD, 758

DTD/Schema menu, 790

DTDs, 267, 955, 958
    converting to XML Schemas (Enterprise and Professional editions), 268
    editing in Grid View (Enterprise and Professional editions), 268
    editing in Text View, 268
    generating XML document from, 268

DTDSchemaGeneratorDlg,
    Application, 1215
    AttributeTypeDefinition, 1215
    DTDSchemaFormat, 1216
    FrequentElements, 1216
    GlobalAttributes, 1216
    MaxEnumLength, 1216
    MergeAllEqualNamed, 1217
    OnlyStringEnums, 1217
    OutputPath, 1217
    OutputPathDialogAction, 1217
    Parent, 1218
    ResolveEntities, 1218
    TypeDetection, 1218
    ValueList, 1218

Duplicate,
    row (in Authentic View), 847

Dynamic (SPS) tables in Authentic View,
    usage of, 400

Dynamic tables,
    editing, 382

E

Eclipse platform,
    and XMLSpy, 671
    and XMLSpy Integration Package, 672

XMLSpy Debugger perspectives, 682
XMLSpy perspective in, 678

Edit,
    macro button, 950

Edit menu, 704

Edited with XMLSpy, 955

Editing database records, 862

Editing in Text View, 136

Editing views, 130

Element, 31
    add as child in Grid View, 769
    appending in Grid View, 761
    convert to in Grid View, 775
    inserting in Grid View, 753
    making optional, 31
    restricting content, 31

ELEMENT declaration,
    add as child in Grid View, 773
    appending in Grid View, 766
    convert to in Grid View, 776

ELEMENT declaration in DTD,
    inserting in Grid View, 758

element type,
    specifying in XML document, 60

ElementFormDefault,
    settings in Schema Design View, 804

ElementList,
    Count, 1219
    Item, 1219
    RemoveElement, 1219

ElementListItem,
    ElementKind, 1220
    FieldCount, 1220
    Name, 1220
    RecordCount, 1220

Elements entry helper,
    in Authentic View, 387

E-mail,
    sending files with, 699

Empty elements, 958

Empty lines,
    in XML documents in Text View, 251

Enable breakpoint - XSLT debugger, 836

Enable tracepoint - XSLT debugger, 837

Enclose in Element command, 780

Encoding,
    default, 961
    of files, 693
End, debugger session, 835
End User License Agreement, 1564, 1567
End-of-line markers, 908
Enhanced Grid View, 904
see Grid View, 62
Enter key, effects of using, 420
Entities, defining in Authentic View, 395, 416
in XML Schema-based XML, 249
inserting in Authentic View, 378, 395
Entities entry helper, in Authentic View, 387
ENTITY declaration, add as child in Grid View, 774
appending in Grid View, 767
convert to in Grid View, 777
inserting in Grid View, 759
Entry Helper, Details, 22
in Grid View, 71
Entry helpers, 104
for XML documents, 258
for XQuery, 321
toggling display on and off, 983
updating, 788
Entry helpers in Schema View, 219
Entry helpers in Text View, 149
Entry-Helper, 983
Enumeration, defining for attributes, 46
Enumerations, in XMLSpyControl, 1455
SPYAttributeTypeDefinition, 1332
SPYAuthenticActions, 1332
SPYAuthenticDocumentPosition, 1332
SpyAuthenticElementActions, 1333
SPYAuthenticElementKind, 1333
SPYAuthenticMarkupVisibility, 1333
SPYDatabaseKind, 1334
SPYDialogAction, 1334
SPYDOMType, 1334
SPYDTDSchemaFormat, 1335
SPYEncodingByteOrder, 1335
SPYExportNamespace, 1335
SPYFrequentElements, 1335
SPYKeyEvent, 1336
SPYLibType, 1336
SPYLoading, 1336
SPYMouseEvent, 1337
SPYNumberDateTimeFormat, 1337
SPYProgrammingLanguage, 1338
SPYProjectItemTypes, 1338
SPYProjectType, 1338
SPYSampleXMLGenerationOptimization, 1339
SPYSampleXMLGenerationSchemaOrDTDAssignment, 1339
SPYSchemaDefKind, 1339
SPYSchemaDocumentationFormat, 1340
SPYTextDelimiters, 1341
SPYTextEnclosing, 1341
SPYTypeDetection, 1341
SPYURLTapes, 1341
SPYViewModes, 1342
SPYVirtualKeyMask, 1342
SPYXMLDataKind, 1343
Enumerations of simple types, 225
EPUB files, 509
Evaluating XPath, 107
Evaluation key, for your Altova software, 987
Evaluation period, of Altova's software products, 1564, 1565
Event, 1096, 1097, 1098, 1149, 1150, 1151, 1152, 1153, 1154, 1155, 1156, 1184, 1185, 1186, 1240, 1241, 1242
Event handlers, in Scripting Project, 1016
overview, 1003
Events, 1016, 1063
and event handlers, 1006
Excel 2007, 244, 501
Expand, fully, 906
Explorer, 958
Exporting XML data to database, 896
Exporting XML data to text files, 893
ExportSettings, CreateKeys, 1221
ElementList, 1221
EntitiesToText, 1222
ExportAllElements, 1222
FromAttributes, 1222
FromSingleSubElements, 1222
FromTextValues, 1223
IndependentPrimaryKey, 1223
ExportSettings,
  Namespace, 1223
  SubLevelLimit, 1223
Extended validation, 289
  in SchemaAgent, 822
Extension functions for XSLT and XQuery, 1524
Extension Functions in .NET for XSLT and XQuery,
  see under .NET extension functions, 1533
Extension Functions in Java for XSLT and XQuery,
  see under Java extension functions, 1524
Extension Functions in MSXSL scripts, 1538
External applications,
  opening files in, 940
External ID declaration,
  add as child in Grid View, 773
  appending in Grid View, 766
  convert to in Grid View, 776
  inserting in Grid View, 758
External parsed entities, 958
External XSL processor, 973

File types, 958
Files,
  adding to source control, 733
  comparing two, 922
  comparison options, 929
  most recently used, 703
FileSelectionDlg,
  Application, 1224
  DialogAction, 1224
  FullName, 1225
  Parent, 1225
Find,
  and replace text in document, 718
  text in document, 715
  using regular expressions, 141
Find in Files command, 719
Find in Files Window, 121
Find in Schemas, 291
Find in Schemas Window, 122
Firebird,
  Connecting through JDBC, 549
  Connecting through ODBC, 546
Floating windows, 100
Folding margin, 908
Font, 970
  schema, 970
  Schema Documentation, 970
Font size,
  in Browser View, 912
Fonts in Text View, 132
FOP,
  fonts, 262
Foreign keys,
  disable in SQLite, 544
Form Object Palette, 999
Form Object properties, 1011
Formatting in Text View, 132
Forms,
Forms,
   and built-in commands, 1026
   and event handling, 1013
   and Form Objects, 1011
   creating new, 1010
   in Scripting Projects, 1010
   invocation of, 1006
   naming, 1010
   overview, 1003
   properties of, 1010
   setting tab sequence of objects, 1011
Forward,
   in Schema View, 240

G
Generate,
   DB structure based on schema, 890
Generate Sample XML, 1332, 1339
GenerateSampleXMLDlg,
   Application, 1236
   FillWithSampleData, 1237
   NonMandatoryAttributes, 1238
   NonMandatoryElements, 1238
   Parent, 1239
   RepeatCount, 1239
   TakeFirstChoice, 1240
Global,
   settings, 954
Global attribute groups, 175
Global attributes, 175
Global comments,
   line display of, 172
Global declarations, 1008
   overview, 1003
Global element,
   using in XML Schema, 43
Global elements, 175
Global resources, 593
   active configuration for, 932
   changing configurations, 609
   defining, 594, 931
   defining database-type, 602
   defining file-type, 596
   defining folder-type, 601
   toolbar activation, 938
using, 605, 608, 609
   using file-type and folder-type, 605
Global Resources XML File, 594
Global schema components,
   finding and renaming, 303
Global scripting project,
   of XMLSpy, 998
Go to File, 908
Go to line/char, 907
Grammar, 958
Graphics formats,
   in Authentic View, 419
Gray bar, 906
Grid fonts, 967
Grid view, 71, 155, 904, 906
   and data import/export, 158
   and Table View, 73
   appending elements and attributes, 71
   data-entry in, 62
   editing, 780
   editing in, 156
   editing XML documents in (Enterprise and Professional editions), 253
   entry helpers in, 161
   switching to Table View, 777
   tables, 158
   using Entry Helpers, 71
GridView Events, 1016
GridView,
   CurrentFocus, 1243
   Deselect, 1243
   IsVisible, 1243
   OnBeforeDrag, 1240
   OnBeforeDrop, 1241
   OnBeforeStartEditing, 1241
   OnEditingFinished, 1242
   OnFocusChanged, 1242
   Select, 1243
   SetFocus, 1244
GUI description, 100

H
Help,
   contents, 985
   key map, 986
Help menu, 985
Help system, 985
Hide, 982, 983
Hide markup, 382, 385
Hide markup (in Authentic View), 846
Hotkey, 941
HTML, 423
- integration of XMLSpy, 1403
HTML documents,
  - editing, 424
  - Info window, 424
HTML example,
  - of XMLSpyControl integration, 1401, 1402, 1403
HTML Info window, 424
HTML output,
  - generating in Authentic View from PXF file, 848
HTTP, 490
HTTP message window,
  - and Accept header, 498
  - and HTTP status codes, 498
  - and WADL, 496
  - HTTP methods, 491
  - importing a request into, 496
  - receiving a response, 498
  - sending the request, 491
IBM DB2,
  - assigning XML Schemas to XML file, 868
  - connecting through ODBC, 550
  - managing XML Schemas, 865
  - schema management and assignment, 864
IBM DB2 for i,
  - connecting through ODBC, 555
IBM Informix,
  - connecting through JDBC, 558
Icon,
  - add to toolbar/menu, 937
  - show large, 950
Identity constraint,
  - toggle in Content model view, 46
Identity constraints in Schema View, 204, 212
Image formats,
  - in Authentic View, 419
Import,
  - database data as XML, 880
  - database data based on XML Schema, 889
  - database structure as XML Schema, 885
  - MS Word document as XML, 885
  - text file as XML, 877
Import/export of data,
  - and Grid View, 158
Indentation,
  - in Text View, 714
Indentation guides, 908
Indentation in Text View, 132
Info Window, 104, 983
  - for CSS documents, 426
  - for HTML documents, 424
  - in XSLT/XQuery Debugger, 354
Info window, XSLT tab,
  - and creating Zip folders, 314
  - and Projects, 314
  - and XSLT documents, 314
  - description of, 314
  - see also XSL Outline, 314
Information Windows,
  - arranging, 355
Insert,
  - breakpoint - XSLT debugger, 836
  - row (in Authentic View), 847
  - tracepoint - XSLT debugger, 836
Insert command,
  - in Grid View, 752
Integrating,
  - XMLSpy in applications, 1386
Intelligent Editing, 962
Internet, 990
Internet usage,
  - in Altova products, 1563

J
Java, 1407
  - API, 1344
  - ClassPath, 1344
Java extension functions,
  - constructors, 1529
  - datatype conversions, Java to Xpath/XQuery, 1532
  - datatype conversions, XPath/XQuery to Java, 1531
  - for XSLT and XQuery, 1524
Java extension functions,
instance methods, instance fields, 1531
overview, 1524
static methods, static fields, 1530
user-defined class files, 1526
user-defined JAR files, 1528

Java settings, 977

Java virtual machine,
path setting, 977

JDBC,
as data connection interface, 515
connect to Teradata, 589
setting up a connection (Windows), 537

JRE,
for XMLSpy Plugin for Eclipse, 672

JSON,
and XPath, 449
and XPath/XQuery Output Window, 449
and XQuery, 449
convert JSON instance to/from XML instance, 899
convert JSON schema to/from XML schema, 902

JSON data,
arrays, 433
example, 433
objects, 433
types, 433

JSON documents,
converting to and from XML, 438, 443, 458
creating new, 431
editing in Grid View, 443
editing in Text View, 438
opening in XMLSpy, 431
validating, 447

JSON options, 976

JSON Schema,
adding global definitions, 461
allOf, 483
any, 481
anyOf, 483
arrays, 478
atomic types, 480
conditionals, 485
description, 436
forbidden, 481
generating from JSON instance, 454
if-then-else, 485
multiple, 481
not, 483
numeric definitions, 480
object properties, 469
objects, 469, 472
objects and dependencies, 475
oneOf, 483
operators, 483
primitive types, 480
simple types, 480
string definitions, 480
terminology, 436
type selectors (any, multiple), 481
type selectors (unconstrained, forbidden), 481
unconstrained, 481
unspecified properties, 472

JSON Schema version, 460
default for new documents, 976

JSON Schema View, 459
adding external schemas, 463
configuring, 486
Constraints entry helper, 463
Design View, 467
Details entry helper, 463
type helpers, 463
global and local definitions, 466

JSON transformations with XSLT/XQuery, 449

JSON validation using format, 976

K

Key map, 986

Keyboard shortcut, 941

Key-codes,
for your Altova software, 987

L

Language,
scripting language - changing, 1006

Large markup (in Authentic View), 846

Legal information, 1564

License, 1567
information about, 1564

License metering,
in Altova products, 1566
Licenses, for your Altova software, 987
Line, go to, 907
Line length, word wrap in text view, 907
Line margin, 908
Line numbering in Text View, 133
Line-breaks, 955
Links, following in Authentic View, 395
Little-endian, 961
loading, 1214

Macro, add to menu/toolbar, 947
edit button, 950
Macros, creating with Scripting Editor, 1020
debugging, 1024
editing with Scripting Editor, 1020
execution of, 1006
functions for, in Global Declarations, 1008
how to use in Scripting Project, 1020
overview, 1003
running, 1022
running application macros, 922
setting as Auto-Macro in Scripting Editor, 1020
Main Window, 101
MainCatalog, 783
MapForce, 799
MariaDB, connect through ODBC, 559
Markup, in Authentic View, 382, 385
Markup (in Authentic View), hide, 846
show small/large/mixed, 846
Maximum cell width, 963
Memory, storage of schema information, 803
Memory requirements, 1560
Menu, 945
add macro to, 947
add/delete command, 937
Authentic, 839
Convert, 877
customize, 945
Default/XMLSpy, 945
delete commands from, 945
DTD/Schema, 790
Edit, 704
Help, 985
Project, 723
Schema Design, 804
Tools, 913
View, 904
Window, 982
XML, 752
XSL/XQuery, 826
Menu Bar, 123
Menu Browser, 911
Messages Window, 105
in XSLT/XQuery Debugger, 353
Microsoft Access, connecting through ADO, 521, 561
Microsoft Office 2007, 244, 501
Microsoft SQL Server, connecting through ADO, 563
connecting through ODBC, 566
Microsoft® SharePoint® Server, 744
MIME, 958
Mixed markup (in Authentic View), 846
Model groups, 175
Mostly recently used files, list of, 703
Move Left command, 780
Move Right command, 780
Move up/down, row (Authentic View, 848
MS SQL Server, schema extensions, 819
schema settings, 819, 820
MSXML, 973
msxsl:script, 1538
Multi-user, 955
MySQL, connecting through ODBC, 569
N

Named schema relationships, 819
Named templates, 311
Namespace, 20
Namespace Prefix, 788
Namespaces, 804
Navigation, 50
Navigation history, 240
New features, 91
New file, 684
New XML document, 58
Node, 836
Non-XML files, 958
NOTATION declaration, 774
ODBC, 534
Output formatting, 955
Oracle, 817
Overview, 1058
Parameters, 829
Parent, 1203

Office Open XML, 244, 501
OLE DB, 515
OOXML, see under Office Open XML, 244, 501
Open, 688
Open Office XML, 244, 501
Optional element, 31
Oracle, 817
Oracle database, 575
Oracle XML Db, 872
Ordering Altova software, 987
OS, 1560
Output windows, 983
Overrides, 175
Overview, 1058
Parameters, 829
Parent, 1203
Parser,
  built into Altova products, 1561
  XSLT, 973
Paste,
  as Text, 395
  as XML, 395
Paste As,
  Text, 392
  XML, 392
Paste command, 705
Patterns of simple types, 225
PDF,
  transforming to in XMLSpy, 308
PDF fonts, 262
PDF output,
  generating in Authentic View from PXF file, 848
Pending Update List (PUL), 330
Platforms,
  for Altova products, 1560
Plug-in,
  ATL sample files, 1044
  registration, 1039
  User interface configuration, 1041
  XMLSPY, 1038
Position, 907
  Character, 907
  Line, 907
PostgreSQL,
  connecting natively, 541
  connecting through ODBC, 577
PowerPoint 2007, 244, 501
Presentation, 963
Pretty-print,
  in Text View, 714
Print setup, 702
Printing,
  from Authentic View, 379
Printing options, 700
Processing Instruction,
  add as child in Grid View, 770
  appending in Grid View, 763
  convert to in Grid View, 776
  inserting in Grid View, 755
Processing Instructions in Schema View, 175
Program settings, 954
Programmers' Reference, 994
Programming points,
  in Scripting Project, 1025
Progress OpenEdge database,
  connecting through JDBC, 582
  connecting through ODBC, 579
Project,
  properties, 749
Project management in XMLSpy, 84
Project menu, 723
Project Window, 102, 982, 983
  toggling display on and off, 983
Projects, 741
  adding active files to, 741
  adding external folders to, 742
  adding external Web folders to, 744
  adding files to, 740
  adding folders to, 741
  adding global resources to, 740
  adding related files to, 741
  adding to source control, 733
  adding URL to, 741
  batch processing with, 616
  benefits of using, 616
  closing, 726
  creating new, 725
  how to create and edit, 612
  most recently used, 751
  naming, 612
  opening, 726
  overview, 723
  overview of, 611
  properties of, 612
  reloading, 726
  saving, 612, 726
  using, 616
Projects in XMLSpy,
  benefits of, 85
  how to create, 86
Properties and Events pane, 999
PUBLIC,
  identifier - catalog, 783
PUL in XQuery Update, 330
PXF file,
  generating output documents from Authentic View, 848
Q
Queries,
Queries,
  for DB display in Authentic View, 409
Query,
  see under Database Query window, 850
  see under Query Database, 850
  see under XQuery, 850
Query Database command, 850
Query pane,
  in Database Query window, 858

R
Redefines, 175
Redo command, 705
Regions in SQL scripts, 858
Register,
  plug-in, 1039
Registering your Altova software, 987
Registry,
  settings, 954
Regular expressions, 719
  find and replace using, 141
  in search string, 715
Reload, 955
Reloading,
  changed files, 693
Remove,
  breakpoint - XSLT debugger, 836
  tracepoint - XSLT debugger, 836
Repeated elements, 962
Replace, 121
  text, 715
  text in document, 718
  text in multiple files, 719
  using regular expressions, 141
Reset,
  menu commands, 945
  shortcut, 941
  toolbar & menu commands, 938
Restart,
  XSLT debugger, 835
Return key,
  see Enter key, 420
RichEdit, 847
Row,
  append (in Authentic View), 847
  delete (in Authentic View), 847
  duplicate (in Authentic View), 847
  insert (in Authentic View), 847
  move up/down, 848
Row in Grid View,
  appending, 778
  inserting, 778
RTF output,
  generating in Authentic View from PXF file, 848

S
Sample values of simple types, 225
save, 1205
Saving files,
  encoding of, 693
schema, 1106
  also see XML Schema, 790
  assigning to DB XML, 868
  converting to UML, 797
  create DB based on schema, 890
  Design view, 905
  documentation, 53
  Documentation font, 970
  management and assignment in IBM DB2 databases, 864
  see XML Schema, 17
  settings, 955
Schema Design menu, 804
Schema Design View,
  Display all globals, 817
  Display diagram, 817
  zoom feature, 816
Schema editing,
  content models, 183
Schema fonts, 970
Schema Overview, 18
  and Content Model View, 171, 172
  and global comments, 172
  and line display of global comments, 172
  editing in, 172
  icons in, 172
  sorting components in, 172
Schema Subsets, 271, 822, 824
Schema View, 165
  Components entry helper, 220, 225
  configuring the view, 29
**Schema View, 165**  
Details entry helper, 223  
entry helpers, 219  
moving back and forward, 240

**Schema View, searching in,**  
see Find in Schemas, 291

**SchemaAgent,**  
connect to server from XMLSpy, 821  
disconnect from server, 821  
display schemas in, 822  
extended validation, 822  
on opening schemas from XMLSpy, 289  
working with, 283, 284, 289

**SchemaAgent in XMLSpy, 280**

**SchemaAgent Server,**  
connecting to, 281

**SchemaDocumentationDlg,**  
AllDetails, 1245  
Application, 1245  
IncludeAll, 1247  
IncludeAttributeGroups, 1247  
IncludeComplexTypes, 1248  
IncludeGlobalElements, 1248  
IncludeGroups, 1248  
IncludeIndex, 1249  
IncludeLocalElements, 1249  
IncludeRedefines, 1250  
IncludeSimpleTypes, 1250  
OptionsDialogAction, 1251  
OutputFile, 1251  
OutputFileDialogAction, 1252  
OutputFormat, 1252  
Parent, 1252  
ShowAnnotations, 1252  
ShowAttributes, 1253  
ShowChildren, 1253  
ShowConstraints, 1254  
ShowDiagram, 1253  
ShowEnumerations, 1254  
ShowNamespace, 1254  
ShowPatterns, 1255  
ShowProgressBar, 1255  
ShowProperties, 1255  
ShowResult, 1255  
ShowSingleFacets, 1256  
ShowSourceCode, 1256  
ShowType, 1256  
ShowUsedBy, 1257

**Schemas,**  
in memory, 803  
managing for IBM DB2, 865

**Schemas, finding in,**  
see Find in Schemas, 291

**Script language, 978**

**Scripting, 978**

**Scripting Editor,**  
GUI description, 999  
Main Window, 999  
starting, 921

**Scripting Environment, 996**  
usage overview, 998

**Scripting language, 1006**

**Scripting Project,**  
and Events, 1016  
application event handlers, 1016  
Event Handlers, 1003  
Forms, 1003  
Forms in, 1010  
Global Declarations, 1003  
Global Declarations in, 1008  
Macros, 1003  
Macros in, 1020  
programming points, 1025  
steps for creating, 1006

**Scripting Project Tree pane, 999**

**Scripting Projects,**  
for XMLSpy, 998  
for XMLSpy Project, 998

**Scripts in XSLT/XQuery,**  
see under Extension functions, 1524

**Search,**  
see Find, 718

**Searching in schemas,**  
see Find in Schemas, 291

**Select All command, 715**

**Sequence compositor,**  
using, 22

**Settings, 125, 954**  
scripting, 978  
XSLT Debugger, 838

**Settings for file comparison, 929**

**SharePoint® Server, 744**

**Shortcut, 941**  
assigning/deleting, 941  
show in tooltip, 950

**Show, 982, 983**
Show curr. exec. nodes, XSLT debugger, 836
Show large markup, 382, 385
Show mixed markup, 382, 385
Show small markup, 385
Show small markup, 382
Side-by-side, 963
Simple type,
  assertions on, 225
  defining facets of, 225
  enumerations of, 225
  in schema definitions, 35
  patterns of, 225
  sample values of, 225
Simple type derivations, 223
Simple types,
  anonymous, 175
  global, 175
  named, 175
Small markup (in Authentic View), 846
Smart Fix for XML Schemas, 229
Smart Restrictions, 234
SOAP Debugger,
  in Visual Studio .NET, 670
Software product license, 1567
Source control, 978
  add to source control, 733
  changing provider, 739
  checking out, 730
  enabling, disabling, 728
  get latest version, 729
  getting files, 729
  installing a source-control plug-in, 640
  open project, 727
  properties, 738
  refresh status, 739
  removing from, 734
  sharing from, 734
  show differences, 737
  show history, 736
  supported providers, 726
  undo check out, 732
Source control manager, 739
Source folding in Text View, 133
Spelling checker, 913
  custom dictionary, 913
Spelling options, 917
Splash screen, 963
Split View in Text View, 150
SPP file locations, 723
SPS,
  assigning to new XML file, 684
SPS file,
  assigning to XML file, 841
SPS tables,
  editing dynamic tables, 382
SPS tables in Authentic View,
  usage of, 400
SpyProject,
  CloseProject, 1258
  ProjectFile, 1258
  RootItems, 1258
  SaveProject, 1259
  SaveProjectAs, 1259
SpyProjectItem,
  ChildItems, 1259
  FileExtensions, 1260
  ItemType, 1260
  Name, 1260
  Open, 1260
  ParentItem, 1260
  Path, 1261
  ValidateWith, 1261
  XMLForXSLTransformation, 1261
  XSLForXMLTransformation, 1261
  XSLTransformationFileExtension, 1261
  XSLTransformationFolder, 1261
SpyProjectItems,
  AddFile, 1262
  AddFolder, 1262
  AddURL, 1262
  Count, 1263
  Item, 1263
  RemoveItem, 1263
SQL Editor,
  creating query in, 861
  description of, 858
  in Database Query window, 858
SQL Server,
  connecting through ADO, 521
  connecting through ADO.NET, 527
  manage XML Schemas, 870
SQLite,
  disable foreign keys, 544
  setting up a connection (Windows), 544
Start,
Start, XSLT debugger, 834
Start group,
ad (context menu), 950
Static (SPS) tables in Authentic View,
usage of, 400
Status Bar, 123
Step into,
XSLT debugger, 835
Step out,
XSLT debugger, 835
Step over,
XSLT debugger, 835
Stop,
XSLT debugger, 834
Strip whitespace, 715
Structured text, 962
Style, 967
Stylesheet PI, 833
StyleVision, 799
for editing StyleVision Power Stylesheet, 841
StyleVision Power Stylesheet,
assigning to XML file, 841
editing in StyleVision, 841
Support Center, 990
Support options, 128
Sybase,
connecting through JDBC, 584
Syntax checking of JSON documents, 438, 443
Syntax coloring,
for XQuery, 321
Syntax-coloring, 958, 963
System DSN,
setting up, 534

T
Tab characters, 955
Tab size,
and pretty-printing, 908
setting, 908
Table,
build automatically, 958
Table command,
in Grid View, 777
Table of contents, 985

Table View, 962
and switching to Grid View, 777
append row, 778
how to use, 73
insert row, 778
sorting columns, 779
Tables,
editing dynamic (SPS) tables, 382
in Authentic View, 373
in Grid View, 158
Tables in Authentic View,
icons for editing XML tables, 405
usage of, 400
using SPS (static and dynamic) tables, 400
using XML tables, 401
Technical Information, 1559
Technical Support, 990
Template files,
for new documents, 684
Template XML File,
in Authentic View, 369
Templates,
of XML documents in Authentic View, 840
Templates Window,
in XSLT/XQuery Debugger, 353
Teradata,
connect through JDBC, 589
connect through ODBC, 585
terminate, 1110
Tex View,
Split View in, 150
Text,
add as child in Grid View, 769
appending in Grid View, 762
convert to in Grid View, 775
editing in Authentic View, 395
find and replace, 718
finding in document, 715
formatting in Authentic View, 395
inserting in Grid View, 754
pretty-printing, 714
Text file,
export of XML data to, 893
import as XML, 877
Text view, 131, 904
and commenting in XML documents, 251
and empty lines in XML documents, 251
auto-complete enable/disable, 962
Text view, 131, 904
  bookmarks in, 133
  editing in, 63
  Entry helpers in, 149
  font properties, 132
  formatting of text, 132
  indentation, 132
  indentation in, 133
  intelligent editing features, 136
  keyboard shortcuts, 152
  line numbering in, 133
  schema fonts, 970
  source folding in, 133
  special editing features for XML documents, 251
  word-wrapping, 132
Text View Events, 1016
Text View Settings dialog, 908
TextImportExportSettings,
  DestinationFolder, 1264
  EnclosingCharacter, 1264
  Encoding, 1264
  EncodingByteOrder, 1265
  FieldDelimiter, 1265
  FileExtension, 1265
  HeaderRow, 1265
  ImportFile, 1265
Tile,
  horizontally, 982
  vertically, 982
Toggle, 982, 983
Toolbar, 123, 938
  activate/deactivate, 938
  add command to, 937
  add macro to, 947
  create new, 938
  reset toolbar & menu commands, 938
  show large icons, 950
Tools,
  see also External applications, 940
Tools menu, 913
Tooltip, 950
  show, 950
  show shortcuts in, 950
Topic,
  view on TOC, 985
Trace window, 359
  in XSLT/XQuery Debugger, 354
Tracepoint,
  dialog box, 837
Tracepoints,
  using in XSLT/XQuery Debugger, 359
Transformation,
  see XSLT transformation, 827
Trusted locations for Authentic scripts, 848
Turn off automatic validation, 958
type,
  extension in XML document, 60

U
UCS-2, 961
UML,
  converting schemas to, 797
Undo command, 705
Unicode support,
  in Altova products, 1562
Unnamed element relationships,
  MS SQL Server schema settings, 820
Unselected, 907
Update Entry Helpers command, 788
URL, 1113, 1205, 1214
  sending by e-mail, 699
User DSN,
  setting up, 534
User interface,
  configure using plug-in, 1041
User interface description, 100
User Manual, 2, 90
User Reference, 683
UTF-16, 961

V
Validating,
  XML documents, 67
Validating XML documents, 249
Validation, 125, 783
  assigning DTD to XML document, 790
  assigning XML Schema to XML document, 791
  of related schemas using SchemaAgent, 289
Validation messages, 105
Validation of XML Schemas, 229
Index

Validator,
in Altova products, 1561

Value templates (XSLT 3.0),
XPath intelligent editing in, 306

Variables Window,
in XSLT/XQuery Debugger, 351

View,
Browser view, 906
Collapse, 906, 907
Enhance Grid view, 904
Expand, 906
Go to File, 908
Go to line/char, 907
Optimal widths, 907
Schema Design view, 905
Text View, 904

View menu, 904

Visual Studio,
adding the XMLSpy ActiveX Controls to the toolbox, 1389

Visual Studio .Net,
and XMLSpy, 665
and XMLSpy Debuggers, 670
and XMLSpy differences, 668

VS .NET,
and XMLSpy Integration Package, 666

W

WADL,
using for HTTP requests, 496

Watch for changes, 955

Web Server, 990

Well-formed test of JSON documents, 438, 443

Well-formedness check, 781
for XML document, 67

Well-formedness of XML documents, 249

Whitespace,
removing, 715

Whitespace markers, 908

Window, 983
Cascade, 982
Entry-Helper, 983
Info, 983
Open, 984
Project, 982, 983
Tile horizontally, 982

Tile vertically, 982

Window menu, 982

Windows,
auto-hiding, 100
floating, docking, tabbing, 100
managing display of, 100
support for Altova products, 1560

Word 2007, 244, 501

Word 2007+ output,
generating in Authentic View from PXF file, 848

Word document,
import as XML, 885

Word wrap,
enable/disable, 907

Word-wrapping in Text View, 132

Wrap,
word wrap enable/disable, 907

WSDL,
using for HTTP requests, 496

X

XInclude,
adding as child in Grid View, 770
appending in Grid View, 763
inserting in Grid View, 710, 755
inserting in Text View, 710
inserting in XML document, 251

XML,
copying as structured text, 706
Oasis catalog, 783
spelling checker, 913

XML data,
exporting to database, 896
exporting to text file, 893

XML DB,
loading new data row into Authentic View, 842
loading new XML data row, 408

XML Diff,
comparing directories, 926
comparing files, 922, 929

XML document,
assigning to XSLT stylesheet, 833
browse Oracle XML Db, 875
creating new, 58
editing in Text View, 63

© 2019 Altova GmbH
Index

**XML document,**
- generating from DTD, 268
- generating from XML Schema (Enterprise and Professional editions), 270
- opening in Authentic View, 369

**XML document creation,**
- tutorial, 57

**XML documents, 246**
- and commenting in Text View, 251
- and empty lines in Text View, 251
- and XPath expression of a node, 251
- and XQuery, 260
- assigning schemas (incl. DTDs), 249
- automatic validation, 247
- automating XQuery executions of, 260
- automating XSLT transformations of, 260
- checking validity of, 67
- checking well-formedness, 249
- default views of, 247
- editing in Authentic View, 256
- editing in Grid View (Enterprise and Professional editions), 253
- encoding of, 265
- evaluating XPath expressions on, 265
- generating schemas from, 265
- importing and exporting text, 265
- inserting file paths in, 251
- inserting XInclude, 251
- opening, 247
- saving, 247
- searching and replacing in, 265
- Text View editing features for, 251
- transforming with XSLT, 260
- validating, 249

**XML file,**
- generate from DTD or XML Schema, 799

**XML Import,**
- based on schema, 889

**XML menu, 752**

**XML Parser,**
- about, 1561

**XML prolog,**
- add as child in Grid View, 770
- appending in Grid View, 762
- convert to in Grid View, 776
- inserting in Grid View, 754

**XML Schema, 790, 804**
- `<alternative>` element, 197
- adding components, 22
- adding elements with, 27
- also see Schema, 790
- assigning to DB XML, 868
- assigning to XML document, 791
- configuring Content Model View, 813
- configuring the view, 29
- content model diagram, 807
- converting to DTD, 794
- creating a basic schema, 17
- creating a new file, 18
- defining namespaces in, 20
- editing content models, 183
- generate outline XML file from, 799
- generating documentation of, 807
- generating from DTD (Enterprise and Professional editions), 268
- generating from XML document, 792
- global components, 175
- go to definition in from XML document, 792
- go to from XML document, 792
- management and assignment in IBM DB2 databases, 864
- managing for IBM DB2, 865
- menu commands related to, 790
- modifying while editing XML document, 77
- MS SQL Server extensions, 819
- MS SQL Server schema settings, 819, 820
- namespaces settings in Schema Design View, 804
- navigation in design view, 50
- Oracle extensions, 817
- Oracle schema settings, 818
- settings in Schema Design View, 804
- Smart Fix, 229
- tutorial, 17
- validation, 229

**XML schema definitions,**
- advanced, 34

**XML Schemas, 267**
- and global resources, 249
- converting to DTD (Enterprise and Professional editions), 270
- editing in DTD (Enterprise and Professional editions), 270
- editing in Grid View (Enterprise and Professional editions), 270
- editing in Schema View (Enterprise and Professional editions), 270
- editing in Text View, 270
- generating XML document from, 270
- plus DTDs, 249

**XML Signature, 843**
XML signatures, 418
XML tables in Authentic View, 405
icons for editing, 405
usage of, 401
XML text, 706
copying
xml:base, 239
xml:id, 239
xml:lang, 239
xml:space, 239
XML-Conformance, 958
XMLData,
  AppendChild, 1273
  EraseAllChildren, 1274
  EraseCurrentChild, 1274
  GetChild, 1275
  GetChildKind, 1276
  GetCurrentChild, 1277
  GetFirstChild, 1277
  GetNextChild, 1278
  HasChildren, 1279
  HasChildrenKind, 1279
  InsertChild, 1280
  IsSameNode, 1281
  Kind, 1281
  MayHaveChildren, 1281
  Name, 1282
  Parent, 1282
  TextValue, 1282
XMLSchemas,
  flattening included schemas, 271, 824
  including other schemas in, 271
  splitting into subsets, 822
XMLSpy, 683
  features, 128
  help, 128
  integration, 1386
  plug-in registration, 1039
XMLSpy API,
  documentation, 1056
  overview, 1058
XMLSpy command table, 1418
XMLSpy Debugger perspectives in Eclipse, 682
XMLSpy Enterprise Edition,
  user manual, 2
XMLSpy in Eclipse, 671
XMLSpy integration,
  example of, 1401, 1402, 1403
XMLSpy Integration Package, 666, 672
XMLSpy perspective in Eclipse, 678
XMLSpy plug-in, 1038
XMLSpy Plugin for Eclipse,
  installing, 672
XMLSpy Plugin for VS .NET,
  installing, 666
XMLSpyCommand,
  in XMLSpyControl, 1437
XMLSpyCommands,
  in XMLSpyControl, 1439
XMLSpyControl, 1439
documentation of, 1386
  example of integration at application level, 1401, 1402, 1403
  examples of integration at document level, 1398
  integration using C#, 1398
  integration using HTML, 1403
  object reference, 1437
XMLSpyControlDocument, 1447
XMLSpyControlPlaceHolder, 1453
XMLSpyDocumentEditor,
  MarkUpView, 1323
XMLSpyLib, 1056, 1058
  Application, 1094
  AuthenticDataTransfer, 1115
  AuthenticRange, 1120
  AuthenticSelection (obsolete), 1300
  AuthenticView, 1148
  CodeGeneratorDlg, 1167
  DatabaseConnection, 1173
  Dialogs, 1179
  DocEditEvent (obsolete), 1284
  DocEditView (obsolete), 1303
  Document, 1182
  Documents, 1210
  DTDSchemaGeneratorDlg, 1215
  ElementList, 1218
  ElementListItem, 1219
  ExportSettings, 1221
  FileSelectionDlg, 1224
  GenerateSampleXMLDlg, 1235
  GridView, 1240
  ProjectItem, 1259
  SchemaDocumentationDlg, 1244
  SpyProject, 1257
  SpyProjectItems, 1262
  TextImportExportSettings, 1263
  XMLData, 1271
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML-Text, 962</td>
</tr>
<tr>
<td>XPath,</td>
</tr>
<tr>
<td>evaluating, 107</td>
</tr>
<tr>
<td>generating of a node in an XML document, 251</td>
</tr>
<tr>
<td>XPath 1.0,</td>
</tr>
<tr>
<td>in XPath Evaluator, 781</td>
</tr>
<tr>
<td>XPath 2.0,</td>
</tr>
<tr>
<td>in XPath Evaluator, 781</td>
</tr>
<tr>
<td>XPath Evaluator,</td>
</tr>
<tr>
<td>usage, 781</td>
</tr>
<tr>
<td>XPath intelligent editing, 306</td>
</tr>
<tr>
<td>XPath of selected node in XML document,</td>
</tr>
<tr>
<td>copying to the clipboard, 710</td>
</tr>
<tr>
<td>XPath to selected node, 381</td>
</tr>
<tr>
<td>XPath Window, 107</td>
</tr>
<tr>
<td>XPaths,</td>
</tr>
<tr>
<td>setting for tracepoints, 359</td>
</tr>
<tr>
<td>XPath-Watch Window,</td>
</tr>
<tr>
<td>in XSLT/XQuery Debugger, 352</td>
</tr>
<tr>
<td>XPointer,</td>
</tr>
<tr>
<td>generating of a node in an XML document, 251</td>
</tr>
<tr>
<td>XPointer of selected node in XML document,</td>
</tr>
<tr>
<td>copying to the clipboard, 710</td>
</tr>
<tr>
<td>XQuery, 327</td>
</tr>
<tr>
<td>DB support, 336</td>
</tr>
<tr>
<td>document validation, 327</td>
</tr>
<tr>
<td>editing in Text View, 317</td>
</tr>
<tr>
<td>entry helpers, 321</td>
</tr>
<tr>
<td>execution, 327</td>
</tr>
<tr>
<td>Extension functions, 1524</td>
</tr>
<tr>
<td>for querying XML databases, 336</td>
</tr>
<tr>
<td>functions for IBM DB2, 336</td>
</tr>
<tr>
<td>intelligent editing features, 323</td>
</tr>
<tr>
<td>opening file, 320</td>
</tr>
<tr>
<td>passing variables to the XQuery document, 829</td>
</tr>
<tr>
<td>syntax coloring, 321</td>
</tr>
<tr>
<td>XQuery Debugger,</td>
</tr>
<tr>
<td>see XSLT/XQuery Debugger, 341</td>
</tr>
<tr>
<td>XQuery Execution, 328, 832</td>
</tr>
<tr>
<td>XQuery files,</td>
</tr>
<tr>
<td>setting file extensions in XMLSpy, 320</td>
</tr>
<tr>
<td>XQuery options, 975</td>
</tr>
<tr>
<td>XQuery Update, 328</td>
</tr>
<tr>
<td>previewing, 330</td>
</tr>
<tr>
<td>PUL, 330</td>
</tr>
<tr>
<td>XQuery Update Facility, 330</td>
</tr>
<tr>
<td>XQuery Update in XMLSpy, 330</td>
</tr>
<tr>
<td>XQuery Update options, 975</td>
</tr>
<tr>
<td>XSD 1.0 and 1.1,</td>
</tr>
<tr>
<td>editing modes, 167</td>
</tr>
<tr>
<td>XSD mode, 167</td>
</tr>
<tr>
<td>XSD validation, 229</td>
</tr>
<tr>
<td>xsi:type,</td>
</tr>
<tr>
<td>usage, 60</td>
</tr>
<tr>
<td>XSL,</td>
</tr>
<tr>
<td>see XSLT, 834</td>
</tr>
<tr>
<td>XSL Outline, 310</td>
</tr>
<tr>
<td>XSL Outline window, 119, 311</td>
</tr>
<tr>
<td>XSL transformation,</td>
</tr>
<tr>
<td>see XSLT, 79</td>
</tr>
<tr>
<td>XSL/XQuery menu, 826</td>
</tr>
<tr>
<td>xsl:call-template, 311</td>
</tr>
<tr>
<td>XSL-FO,</td>
</tr>
<tr>
<td>and XSLT transformations, 308</td>
</tr>
<tr>
<td>xsl:param, 311</td>
</tr>
<tr>
<td>xsl:with-param, 311</td>
</tr>
<tr>
<td>XSLT, 907</td>
</tr>
<tr>
<td>and batch transformations, 308</td>
</tr>
<tr>
<td>auto-completion in Text View, 306</td>
</tr>
<tr>
<td>documents, 306</td>
</tr>
<tr>
<td>entry helpers for, 306</td>
</tr>
<tr>
<td>Extension functions, 1524</td>
</tr>
<tr>
<td>functionality in XMLSpy, 305</td>
</tr>
<tr>
<td>modifying in XMLSpy, 82</td>
</tr>
<tr>
<td>processor, 973</td>
</tr>
<tr>
<td>transformations in XMLSpy, 308</td>
</tr>
<tr>
<td>validating, 306</td>
</tr>
<tr>
<td>XSLT Debugger,</td>
</tr>
<tr>
<td>breakpoints/tracepoints dialog box, 837</td>
</tr>
<tr>
<td>debug windows, 838</td>
</tr>
<tr>
<td>enable/disable breakpoint, 836</td>
</tr>
<tr>
<td>enable/disable tracepoint, 837</td>
</tr>
<tr>
<td>end debugger session, 835</td>
</tr>
<tr>
<td>in Visual Studio .NET, 670</td>
</tr>
<tr>
<td>insert/remove breakpoint, 836</td>
</tr>
<tr>
<td>insert/remove tracepoint, 836</td>
</tr>
<tr>
<td>restart debugger, 835</td>
</tr>
<tr>
<td>settings, 838</td>
</tr>
<tr>
<td>show curr. exec. nodes, 836</td>
</tr>
<tr>
<td>start debugger, 834</td>
</tr>
<tr>
<td>step into, 835</td>
</tr>
<tr>
<td>step out, 835</td>
</tr>
<tr>
<td>step over, 835</td>
</tr>
<tr>
<td>stop debugger, 834</td>
</tr>
<tr>
<td>XSLT document structure, 119</td>
</tr>
</tbody>
</table>
| XSLT documents,
XSLT documents,
and Info window, 314
ing XSL Outline, 310
editing and managing with XSL Outline, 310

XSLT functions,
in XSL Outline window, 311

XSLT parameters,
passing to stylesheet via interface, 829

XSLT stylesheet,
assigning to XML document, 833
assigning XML document to, 833
opening, 834

XSLT stylesheet for FO,
assigning to XML document, 833

XSLT templates,
in XSL Outline window, 311

XSLT transformation, 827
assigning XSLT file, 80
in XMLSpy, 81
to FO, 827
to PDF, 827
tutorial, 79

XSLT/XQuery Debugger, 342
breakpoints usage, 356
Call Stack Window, 352
Context window, 351
description of interface, 342
description of mechanism, 342
features and usage, 341
Info Window, 354
information windows, 349
keyboard shortcuts, 364
Messages Window, 353
settings, 347
starting a session, 348
Templates Window, 353
toolbar icons, 344
Trace Window, 354
tracepoints usage, 359
Variables Window, 351
XPath-Watch Window, 352

XSLT/XQuery debugging,
files used, 341

creating in Archive View, 507
editing in Archive View, 507

Zoom feature,
in Schema Design View, 816
Zooming in Text View, 133

ZIP files, 244, 501