User and Reference Manual
# Table of Contents

1 Altova StyleVision 2015 Enterprise Edition 3

2 About this Documentation 6

3 New Features: Version 2015 10
   3.1 Version 2014 ................................................................. 11
   3.2 Version 2013 ................................................................. 12
   3.3 Version 2012 ................................................................. 13
   3.4 Version 2011 ................................................................. 14
   3.5 Version 2010 ................................................................. 16

4 Introduction 20
   4.1 What Is an SPS? ............................................................... 21
   4.2 Product Features ............................................................... 24
   4.3 Terminology ................................................................. 29
   4.4 Setting up StyleVision ....................................................... 33
   4.5 Authentic View in Altova Products .................................. 36

5 User Interface 38
   5.1 Main Window ............................................................... 40
      5.1.1 Design View .......................................................... 42
      5.1.2 Authentic View ....................................................... 44
      5.1.3 Output Views ....................................................... 46
   5.2 Sidebars ........................................................................ 48
      5.2.1 Design Overview ..................................................... 52
      5.2.2 Schema Tree .......................................................... 56
      5.2.3 Design Tree .......................................................... 60
      5.2.4 Style Repository ..................................................... 64
      5.2.5 Styles ................................................................. 67
      5.2.6 Properties ............................................................. 69
### 6 Quick Start Tutorial

6.1 Creating and Setting Up a New SPS ................................................................. 79
6.2 Inserting Dynamic Content (from XML Source) ........................................... 83
6.3 Inserting Static Content .................................................................................. 90
6.4 Formatting the Content .................................................................................. 95
6.5 Using Auto-Calculations ............................................................................... 101
6.6 Using Conditions ......................................................................................... 105
6.7 Using Global Templates and Rest-of-Contents ............................................ 112
6.8 That's It! ........................................................................................................ 117

### 7 Usage Overview

7.1 SPS and Sources ......................................................................................... 122
7.2 Creating the Design ..................................................................................... 124
7.3 XSLT and XPath Versions .......................................................................... 126
7.4 Internet Explorer Compatibility .................................................................... 127
7.5 SPS and Authentic View ............................................................................. 129
7.6 Synchronizing StyleVision and Authentic .................................................. 131
7.7 Generated Files ......................................................................................... 132
7.8 Projects in StyleVision .............................................................................. 135
7.9 Catalogs in StyleVision ............................................................................. 141

### 8 SPS File: Content

8.1 Inserting XML Content as Text .................................................................. 149
8.1.1 Inserting Content with a Predefined Format .......................................... 152
8.1.2 Adding Elements in Authentic View ..................................................... 153
8.1.3 Rest-of-Contents .................................................................................. 156
8.2 User-Defined Templates ........................................................................... 157
8.3 User-Defined Elements, XML Text Blocks .............................................. 160
8.3.1 User-Defined Elements ....................................................................... 161
8.3.2 User-Defined XML Text Blocks ......................................................... 163
8.4 Tables ......................................................................................................... 165
8.4.1 Static Tables ....................................................................................... 168
8.4.2 Dynamic Tables .................................................................................. 171
8.4.3 Conditional Processing in Tables ......................................................... 177
8.4.4 Tables in Design View ................................................................. 179
8.4.5 Table Formatting ...................................................................... 181
8.4.6 Row and Column Display .......................................................... 186
8.4.7 CALS/HTML Tables ................................................................... 188
8.5 Lists ............................................................................................. 193
  8.5.1 Static Lists .............................................................................. 194
  8.5.2 Dynamic Lists ........................................................................ 196
8.6 Graphics ....................................................................................... 199
  8.6.1 Images: URIs and Inline Data .................................................. 200
  8.6.2 Image Types and Output ......................................................... 203
  8.6.3 Example: A Template for Images ........................................... 206
8.7 Charts ........................................................................................... 207
  8.7.1 Chart Basics ........................................................................... 210
  8.7.2 Chart Settings: Quick Reference ............................................. 214
  8.7.3 Chart Settings and Appearance ............................................. 220
    Basic Chart Settings ..................................................................... 221
    Advanced Chart Settings .............................................................. 228
  8.7.4 Chart Data Selection: Simple ................................................ 245
  8.7.5 Chart Data Selection: Flexible ............................................... 247
  8.7.6 Example: Pie Charts ............................................................... 252
  8.7.7 Example: Bar Charts ............................................................... 265
  8.7.8 Example: Line Charts .............................................................. 271
  8.7.9 Example: Value Line Charts .................................................. 274
  8.7.10 Example: Area Charts ............................................................ 278
  8.7.11 Example: Candlestick Charts ............................................... 283
  8.7.12 Example: Gauge Charts ....................................................... 289
  8.7.13 Example: Overlay Charts ..................................................... 292
8.8 Form Controls .............................................................................. 295
  8.8.1 Input Fields, Multiline Input Fields ......................................... 297
  8.8.2 Check Boxes .......................................................................... 298
  8.8.3 Combo Boxes .......................................................................... 300
  8.8.4 Radio Buttons, Buttons ......................................................... 303
8.9 Links ............................................................................................. 304
8.10 Barcodes ...................................................................................... 305
8.11 Layout Modules .......................................................................... 310
  8.11.1 Layout Containers ................................................................. 311
  8.11.2 Layout Boxes ......................................................................... 315
  8.11.3 Lines ...................................................................................... 320
8.12 The Change-To Feature ............................................................... 323
9 SPS File: Structure

9.1 Schema Sources ................................................................. 330
  9.1.1 DTDs and XML Schemas .............................................. 333
  9.1.2 DB Schemas ................................................................. 339
  9.1.3 XBRL Taxonomy ........................................................... 340
  9.1.4 User-Defined Schemas ............................................... 346
  9.1.5 Multiple Schema Sources ......................................... 349
9.2 Merging XML Data from Multiple Sources ......................... 352
9.3 Modular SPSs .................................................................. 355
  9.3.1 Available Module Objects ............................................ 358
  9.3.2 Creating a Modular SPS ............................................... 362
  9.3.3 Example: An Address Book ....................................... 366
9.4 Templates and Design Fragments ....................................... 371
  9.4.1 Main Template .............................................................. 372
  9.4.2 Global Templates ........................................................ 373
  9.4.3 User-Defined Templates .............................................. 379
  9.4.4 Variable Templates ..................................................... 382
  9.4.5 Node-Template Operations .......................................... 383
  9.4.6 Design Fragments ....................................................... 387
9.5 XSLT Templates ................................................................ 391
9.6 Multiple Document Output ................................................. 393
  9.6.1 Inserting a New Document Template ............................ 395
  9.6.2 New Document Templates and Design Structure ............ 397
  9.6.3 URLs of New Document Templates ............................... 399
  9.6.4 Preview Files and Output Document Files ...................... 402
  9.6.5 Document Properties and Styles .................................. 405

10 SPS File: Advanced Features .............................................. 408

10.1 Auto-Calculations ............................................................. 409
  10.1.1 Editing and Moving Auto-Calculations ......................... 410
  10.1.2 Updating Nodes with Auto-Calculations ....................... 413
  10.1.3 Auto-Calculations Based on Updated Nodes .................. 416
  10.1.4 Example: An Invoice ................................................ 418
10.2 Conditions .................................................................. 422
  10.2.1 Setting Up the Conditions .......................................... 423
  10.2.2 Editing Conditions ..................................................... 426
  10.2.3 Output-Based Conditions ........................................... 427
10.2.4 Conditions and Auto-Calculations ................................................................. 429
10.3 Conditional Presence ..................................................................................... 430
10.4 Grouping ......................................................................................................... 432
10.4.1 Example: Group-By (Persons.sps) ................................................................. 435
10.4.2 Example: Group-By (Scores.sps) ................................................................. 437
10.5 Sorting .............................................................................................................. 441
10.5.1 The Sorting Mechanism .............................................................................. 442
10.5.2 Example: Sorting on Multiple Sort-Keys .................................................... 444
10.6 Parameters and Variables .............................................................................. 447
10.6.1 User-Declared Parameters ........................................................................... 448
10.6.2 Parameters for Design Fragments ............................................................... 450
10.6.3 SPS Parameters for Sources ....................................................................... 453
10.6.4 Variables ..................................................................................................... 454
10.6.5 Editable Variables in Authentic ................................................................. 457
10.7 Table of Contents, Referencing, Bookmarks .................................................. 460
10.7.1 Bookmarking Items for TOC Inclusion ....................................................... 464
Structuring the Design in TOC Levels ................................................................. 466
Creating TOC Bookmarks ................................................................................. 469
10.7.2 Creating the TOC Template ....................................................................... 473
Levelrefs in the TOC Template ........................................................................... 475
TOC References: Name, Scope, Hyperlink ......................................................... 477
Formatting TOC Items ....................................................................................... 478
10.7.3 Example: Simple TOC .................................................................................. 480
10.7.4 Example: Hierarchical and Sequential TOCs ............................................. 484
10.7.5 Auto-Numbering in the Document Body .................................................... 488
10.7.6 Cross-referencing ...................................................................................... 492
10.7.7 Bookmarks and Hyperlinks ....................................................................... 494
Inserting Bookmarks ......................................................................................... 495
Defining Hyperlinks ......................................................................................... 497
10.8 Example: Multiple Languages ...................................................................... 502

11 SPS File: Presentation .................................................................................. 506
11.1 Predefined Formats ....................................................................................... 508
11.2 Output Escaping ............................................................................................ 510
11.3 Value Formatting (Formatting Numeric Datatypes) ...................................... 513
11.3.1 The Value Formatting Mechanism ............................................................. 514
11.3.2 Value Formatting Syntax ........................................................................... 517
11.4 Working with CSS Styles .............................................................................. 523
11.4.1 External Stylesheets ................................................................................... 525
11.4.2 Global Styles ................................................................. 530
11.4.3 Local Styles ................................................................. 533
11.4.4 Setting Style Values .................................................... 536
11.4.5 Style Properties Via XPath ........................................... 539
11.4.6 Composite Styles ....................................................... 543

11.5 Text-Styling Flexibility in Authentic ................................. 546
11.5.1 Composite Styles ....................................................... 547
11.5.2 RichEdit ................................................................. 549
11.5.3 Text State Icons ........................................................ 553

11.6 Designing Print Output ..................................................... 556
11.6.1 Document Sections ..................................................... 558
  Initial Document Section .................................................... 561
  Page Layout Properties ...................................................... 564
  Headers and Footers: Part 1 .................................................. 571
  Headers and Footers: Part 2 .................................................. 574
11.6.2 Keeps and Breaks ........................................................ 577
11.6.3 Footnotes ............................................................... 578
11.6.4 PDF Fonts ............................................................... 580
11.6.5 Pixel Resolution ........................................................ 584
11.6.6 PDF Bookmarks ......................................................... 587
11.6.7 Watermarks ............................................................. 592

12 SPS File: Additional Functionality 598

12.1 Altova Global Resources ................................................. 599
12.1.1 Defining Global Resources ......................................... 601
  Files ............................................................. 604
  Folders ........................................................... 610
  Databases ............................................................ 612
12.1.2 Using Global Resources .............................................. 615
  Assigning Files and Folders ............................................. 616
  Assigning Databases ...................................................... 620
  Changing the Active Configuration .................................. 622
12.2 Authentic Node Properties ............................................. 623
12.3 Replace Parent Node OnClick With .................................. 625
12.4 Additional Validation ..................................................... 628
12.5 Unparsed Entity URIs ...................................................... 630
12.6 New from XSLT, XSL-FO or FO File .................................. 632
12.7 User-Defined XPath Functions ....................................... 636
12.7.1 Defining an XPath Function ....................................... 639
12.7.2 Reusing Functions to Locate Nodes ............................................ 642
12.7.3 Parameters in XPath Functions ............................................... 643

Parameters and Sequences ............................................................... 646

Parameters and Nodes .................................................................. 651

12.8 Working with Dates .................................................................. 653
12.8.1 Using the Date-Picker ............................................................. 655
12.8.2 Formatting Dates ................................................................. 657

12.9 Using Scripts .............................................................................. 660
12.9.1 Defining JavaScript Functions ............................................. 662
12.9.2 Assigning Functions as Event Handlers ................................. 664
12.9.3 External JavaScript Files ..................................................... 665

12.10 HTML Import ......................................................................... 667
12.10.1 Creating New SPS via HTML Import .................................... 668
12.10.2 Creating the Schema and SPS Design ................................. 670
12.10.3 Creating Tables and Lists as Elements/Attributes ............... 673
12.10.4 Generating Output ............................................................. 676

12.11 XML Signatures ..................................................................... 677
12.11.1 XML Signature Configuration ........................................... 680
12.11.2 XML Signature Settings ..................................................... 682
12.11.3 Signing the Authentic XML Document ............................... 685
12.11.4 Signature Verification in Authentic .................................... 687
12.11.5 XML Document Validity .................................................... 689
12.11.6 Working with Certificates .................................................. 691

12.12 ASPX Interface for Web Applications ...................................... 695
12.12.1 Example: Localhost on Windows 7 ....................................... 697

12.13 PXF File: Container for SPS and Related Files ....................... 699
12.13.1 Creating a PXF File ............................................................ 700
12.13.2 Editing a PXF File ............................................................. 704
12.13.3 Deploying a PXF File ......................................................... 706

13 **SPS File and Databases** .......................................................... 708

13.1 DBs and StyleVision ................................................................. 710
13.2 Connecting to a Database ........................................................ 712
13.2.1 Connection Wizard .............................................................. 714
13.2.2 ADO Connections ............................................................... 719
13.2.3 ODBC Connections ............................................................ 726
13.2.4 JDBC Connections .............................................................. 729
13.2.5 Global Resources ............................................................... 733

13.3 DB Data Selection .................................................................... 734
13.3.1 Non-XML Databases ............................................................... 735
13.3.2 XML Databases ................................................................. 743
13.4 The DB Schema and DB XML files ............................................. 747
13.5 DB Filters: Filtering DB Data .................................................... 751
13.6 SPS Design Features for DB ..................................................... 756
13.7 Generating Output Files ........................................................... 760
13.8 Query Database ..................................................................... 762
   13.8.1 Data Sources ................................................................. 764
   13.8.2 Browser Pane: Viewing the DB Objects ............................... 767
   13.8.3 Query Pane: Description and Features ............................... 771
   13.8.4 Query Pane: Working with Queries ................................... 775
   13.8.5 Results and Messages ....................................................... 776

14 SPS File and XBRL .................................................................... 780
14.1 Creating an XBRL SPS File ....................................................... 781
14.2 Taxonomy Structure ............................................................... 783
14.3 Taxonomy Source Properties .................................................. 786
14.4 XBRL Templates ................................................................... 789
   14.4.1 Types of Template .......................................................... 790
   14.4.2 Context ........................................................................ 793
   14.4.3 Concept ........................................................................ 794
   14.4.4 Item ............................................................................ 795
   14.4.5 Concept QName .............................................................. 796
   14.4.6 Periods ......................................................................... 797
   14.4.7 Identifiers ..................................................................... 799
   14.4.8 Units and Unitrefs .......................................................... 800
   14.4.9 Dimensions ................................................................... 802
   14.4.10 Non-XDT Segments and Scenarios ................................ 804
   14.4.11 Footnote ..................................................................... 805
   14.4.12 XML Node ................................................................... 807
14.5 Open, Closed, and Classic Modes ............................................ 808
   14.5.1 Concept Properties ......................................................... 809
   14.5.2 Period Properties ............................................................ 811
   14.5.3 Identifier Properties ......................................................... 813
   14.5.4 Unit Properties ............................................................... 815
   14.5.5 Explicit Dimension Properties ......................................... 819
   14.5.6 Typed Dimension Properties ........................................... 821
   14.5.7 Non-XDT Segment/Scenario Properties ............................ 824
14.6 XBRL Tables ........................................................................ 826
14.7 The XBRL Table Wizard ................................................................. 828
  14.7.1 XBRL Table Structure ............................................................ 830
  14.7.2 XBRL Table Options ............................................................... 836
  14.7.3 Component Properties ........................................................... 838
14.8 The XBRL Chart Wizard ............................................................... 847
  14.8.1 Categories ........................................................................... 849
  14.8.2 Series ................................................................................ 851
  14.8.3 Component Properties ........................................................... 854
14.9 Properties and Formatting ............................................................ 864
14.10 Inline XBRL ............................................................................. 867
14.11 Altova XBRL Extension Functions ............................................. 871

15 Authentic View ............................................................................ 878
  15.1 Authentic View Interface ............................................................ 879
    15.1.1 Overview of the GUI ........................................................... 880
    15.1.2 Authentic View Toolbar Icons ........................................... 881
    15.1.3 Authentic View Main Window ............................................. 883
    15.1.4 Authentic View Entry Helpers ............................................. 886
    15.1.5 Authentic View Context Menus ............................................ 890
  15.2 Editing in Authentic View ............................................................ 892
    15.2.1 Basic Editing .................................................................... 893
    15.2.2 Tables in Authentic View .................................................... 898
      SPS Tables .............................................................................. 899
      CALS/HTML Tables ................................................................. 901
      CALS/HTML Table Editing Icons .............................................. 905
    15.2.3 Editing a DB ..................................................................... 907
      Navigating a DB Table ............................................................. 908
      DB Queries ............................................................................ 909
      Modifying a DB Table ............................................................ 913
    15.2.4 Working with Dates .............................................................. 915
      Date Picker ............................................................................ 916
      Text Entry ............................................................................. 917
    15.2.5 Defining Entities ................................................................. 918
    15.2.6 XML Signatures ................................................................. 920
    15.2.7 Images in Authentic View .................................................. 922
    15.2.8 Keystrokes in Authentic View ............................................. 923

16 Authentic Scripting .................................................................... 926
16.1 Scripting Editor .............................................................................................................. 929
16.2 Macros ............................................................................................................................. 930
  16.2.1 Macros on Design Elements .................................................................................. 931
  16.2.2 Macros on Context Menu Items ............................................................................ 933
  16.2.3 Custom Buttons ....................................................................................................... 934
16.3 Event Handlers ............................................................................................................... 936
16.4 Scripting Options .......................................................................................................... 937

17 Automated Processing 940
17.1 Command Line Interface ............................................................................................... 942
  17.1.1 StyleVision .............................................................................................................. 943
  17.1.2 StyleVision Server ................................................................................................ 945
17.2 Using RaptorXML .......................................................................................................... 946
  17.2.1 PDF Output ............................................................................................................ 947
17.3 Automation with FlowForce Server .............................................................................. 949
17.4 How to Automate Processing ......................................................................................... 951

18 StyleVision in Visual Studio 954
18.1 Installing the StyleVision Plugin .................................................................................. 955
18.2 Differences with StyleVision Standalone ..................................................................... 956

19 StyleVision in Eclipse 958
19.1 Installing the StyleVision Plugin for Eclipse ............................................................... 959
19.2 Stylevision Entry Points in Eclipse ............................................................................. 966

20 User Reference 972
20.1 Design View Symbols .................................................................................................... 973
20.2 Edit XPath Expression Dialog ...................................................................................... 978
  20.2.1 XPath Expression Builder ..................................................................................... 980
  20.2.2 XPath Expression Evaluator ............................................................................... 985
20.3 Toolbars ....................................................................................................................... 988
  20.3.1 Format .................................................................................................................... 991
  20.3.2 Table ..................................................................................................................... 992
  20.3.3 Authentic ............................................................................................................... 994
  20.3.4 RichEdit ................................................................................................................. 996
  20.3.5 Insert Design Elements ....................................................................................... 997
20.3.6 Design Filter ................................................................. 1000
20.3.7 Global Resources ............................................................ 1001
20.3.8 Standard ................................................................. 1002
20.4 File Menu ........................................................................... 1004
  20.4.1 New ............................................................................. 1005
  20.4.2 Open, Reload, Close, Close All ...................................... 1012
  20.4.3 Save Design, Save All .................................................. 1018
  20.4.4 Save As ........................................................................ 1023
  20.4.5 Save Authentic XML Data, Save As ................................ 1027
  20.4.6 Save Generated Files ................................................... 1028
  20.4.7 Deploy to FlowForce .................................................... 1032
  20.4.8 Web Design ................................................................... 1034
  20.4.9 Properties ...................................................................... 1035
  20.4.10 Print Preview, Print ...................................................... 1040
  20.4.11 Most Recently Used Files, Exit ........................................ 1041
20.5 Edit Menu ............................................................................ 1042
  20.5.1 Undo, Redo, Select All .................................................. 1043
  20.5.2 Find, Find Next, Replace ............................................... 1044
  20.5.3 Stylesheet Parameters ................................................... 1047
  20.5.4 Collapse/Expand Markup .............................................. 1048
20.6 Project Menu ........................................................................ 1049
  20.6.1 New Project, Open Project, Reload Project ....................... 1051
  20.6.2 Close Project, Save Project ............................................ 1052
  20.6.3 Add Files / Global Resource / URL to Project ................... 1053
  20.6.4 Add Active (and Related) Files to Project ....................... 1055
  20.6.5 Add Project and External Folders to Project ..................... 1056
20.7 View Menu .......................................................................... 1059
  20.7.1 Toolbars and Status Bar ................................................ 1060
  20.7.2 Design Sidebars .......................................................... 1061
  20.7.3 Design Filter, Zoom ...................................................... 1062
20.8 Insert Menu ........................................................................... 1063
  20.8.1 Contents ....................................................................... 1064
  20.8.2 Rest of Contents .......................................................... 1065
  20.8.3 RichEdit ........................................................................ 1066
  20.8.4 Form Controls ............................................................. 1068
  20.8.5 DB Control ..................................................................... 1069
  20.8.6 Auto-Calculation .......................................................... 1070
  20.8.7 Date Picker ................................................................. 1072
  20.8.8 Paragraph, Special Paragraph ........................................ 1073
  20.8.9 Barcode ........................................................................ 1074

Altova StyleVision 2015 Enterprise Edition
20.8.10 Chart ................................................................. 1076
20.8.11 Image ............................................................... 1078
20.8.12 Horizontal Line .................................................. 1081
20.8.13 Table ................................................................. 1082
20.8.14 Bullets and Numbering ........................................ 1083
20.8.15 Bookmark .......................................................... 1086
20.8.16 Hyperlink .......................................................... 1088
20.8.17 Footnote ............................................................. 1090
20.8.18 Condition, Output-Based Condition ..................... 1092
20.8.19 Template ............................................................ 1094
20.8.20 User-Defined Template ....................................... 1096
20.8.21 Variable Template ............................................... 1097
20.8.22 Design Fragment ................................................ 1098
20.8.23 Layout Container, Layout Box, Line ....................... 1099
20.8.24 Table of Contents ............................................... 1100
20.8.25 New Document .................................................. 1101
20.8.26 Page / Column / Document Section ....................... 1102
20.8.27 XBRL Element .................................................... 1104
20.8.28 User-Defined Item .............................................. 1105
20.9 Enclose With Menu .................................................. 1106
  20.9.1 Template .......................................................... 1107
  20.9.2 User-Defined Template ....................................... 1108
  20.9.3 Variable Template .............................................. 1109
  20.9.4 Paragraph, Special Paragraph ............................... 1110
  20.9.5 Bullets and Numbering ....................................... 1111
  20.9.6 Bookmarks and Hyperlinks .................................. 1112
  20.9.7 Condition, Output-Based Condition ....................... 1113
  20.9.8 TOC Bookmarks and TOC Levels ......................... 1115
  20.9.9 New Document .................................................. 1116
  20.9.10 User-Defined Element ....................................... 1117
20.10 Table Menu .......................................................... 1118
  20.10.1 Insert Table, Delete Table ................................... 1119
  20.10.2 Add Table Headers, Footers ............................... 1120
  20.10.3 Append/Insert Row/Column ............................... 1121
  20.10.4 Delete Row, Column ......................................... 1122
  20.10.5 Join Cell Left, Right, Below, Above ..................... 1123
  20.10.6 Split Cell Horizontally, Vertically ....................... 1124
  20.10.7 View Cell Bounds, Table Markup ......................... 1125
  20.10.8 Table Properties ............................................. 1126
  20.10.9 Edit CALS/HTML Tables ................................. 1127
  20.10.10 Vertical Alignment of Cell Content ..................... 1128
20.11 Authentic Menu .................................................................................................................. 1129
  20.11.1 Edit Authentic Scripts .................................................................................................. 1130
  20.11.2 Custom Toolbar Buttons ......................................................................................... 1131
  20.11.3 Check Macro References ....................................................................................... 1135
  20.11.4 Auto-Add Date Picker .......................................................................................... 1136
  20.11.5 Auto-Add DB Controls .......................................................................................... 1137
  20.11.6 Reload Authentic View, Validate XML ................................................................. 1138
  20.11.7 Select New Row with XML Data for Editing ......................................................... 1139
  20.11.8 XML Signature ........................................................................................................ 1140
  20.11.9 Define XML Entities ............................................................................................... 1142
  20.11.10 View Markup ........................................................................................................ 1143
  20.11.11 RichEdit ................................................................................................................ 1144
  20.11.12 (Dynamic Table) Row Commands .................................................................. 1145
20.12 Database Menu .................................................................................................................. 1146
  20.12.1 Query Database ..................................................................................................... 1147
  20.12.2 Edit DB Filter, Clear DB Filter ............................................................................... 1148
20.13 Properties Menu .............................................................................................................. 1150
  20.13.1 Edit Bullets and Numbering .................................................................................. 1151
  20.13.2 Predefined Value Formatting Strings .................................................................... 1152
20.14 Tools Menu ...................................................................................................................... 1154
  20.14.1 Spelling .................................................................................................................. 1155
  20.14.2 Spelling Options ..................................................................................................... 1157
  20.14.3 Global Resources .................................................................................................. 1161
  20.14.4 Active Configuration ......................................................................................... 1162
  20.14.5 Customize ............................................................................................................ 1163
  20.14.6 Restore Toolbars and Windows .......................................................................... 1168
  20.14.7 Options ................................................................................................................ 1169
20.15 Window Menu .................................................................................................................. 1175
20.16 Help Menu ....................................................................................................................... 1176
  20.16.1 Table of Contents, Index, Search ........................................................................... 1177
  20.16.2 Activation, Order Form, Registration, Updates .................................................... 1178
  20.16.3 Other Commands ................................................................................................. 1180

21  Programmers' Reference  .................................................................................................... 1182
  21.1 Scripting Editor ............................................................................................................. 1183
    21.1.1 Overview .............................................................................................................. 1185
      The Scripting Editor GUI ............................................................................................ 1186
      Components of a Scripting Project ............................................................................ 1190
    21.1.2 Creating a Scripting Project .................................................................................. 1192
### 21.1.3 Global Declarations

- Creating a New Form: 1195
- Form Design and Form Objects: 1197
- Form Events: 1200

### 21.1.4 Forms

- Creating a New Form: 1195
- Form Design and Form Objects: 1197
- Form Events: 1200

### 21.1.5 Events

- Creating and Editing a Macro: 1204
- Running a Macro: 1205
- Debugging a Macro: 1206

### 21.1.6 Macros

- Creating and Editing a Macro: 1204
- Running a Macro: 1205
- Debugging a Macro: 1206

### 21.2 Application API

#### 21.2.1 Overview

- Object Model: 1210
- Programming Languages: 1211

#### 21.2.2 Interfaces

- Application: 1234
- AppOutputLine: 1239
- AppOutputLines: 1244
- AppOutputLineSymbol: 1246
- AuthenticContextMenu: 1248
- AuthenticEventContext: 1250
- AuthenticRange: 1253
- AuthenticView: 1281
- Document: 1296
- Documents: 1306
- Parameter: 1309
- Parameters: 1310
- SchemaSource: 1312
- SchemaSources: 1315
- XMLData: 1317

#### 21.2.3 Enumerations

- ENUMApplicationStatus: 1328
- ENUMAppOutputLine_Severity: 1330
- ENUMAppOutputLine_TextDecoration: 1331
- ENUMSchemaSourceType: 1332
- ENUMSchemaType: 1333
- SPYAuthenticActions: 1334
- SPYAuthenticDocumentKind: 1335
- SPYAuthenticElementKind: 1336
- SPYAuthenticMarkupVisibility: 1337
- SPYAuthenticToolbarButtonState: 1338
21.3 ActiveX Integration ........................................................................................................... 1343
21.3.1 Integration at Application Level ...................................................................................... 1344
21.3.2 Integration at Document Level ........................................................................................ 1345
    Use StyleVisionControl ......................................................................................................... 1346
    Use StyleVisionControlDocument ......................................................................................... 1347
    Use StyleVisionControlPlaceHolder ..................................................................................... 1348
    Query StyleVision Commands ............................................................................................... 1349
21.3.3 Programming Languages ............................................................................................... 1350
    C# ........................................................................................................................................ 1351
    HTML ................................................................................................................................. 1352
    Java ..................................................................................................................................... 1357
21.3.4 Command Table for StyleVision .................................................................................... 1373
    File Menu .......................................................................................................................... 1374
    Edit Menu .......................................................................................................................... 1376
    Project Menu ..................................................................................................................... 1377
    View Menu ......................................................................................................................... 1378
    Insert Menu ........................................................................................................................ 1379
    Enclose With Menu .............................................................................................................. 1382
    Table Menu ........................................................................................................................ 1383
    Authentic Menu .................................................................................................................. 1384
    Database Menu ................................................................................................................... 1385
    Properties Menu .................................................................................................................. 1386
    Tools Menu .......................................................................................................................... 1387
    Window Menu ..................................................................................................................... 1388
    Help Menu .......................................................................................................................... 1389
    Misc Menu ........................................................................................................................... 1390
21.3.5 Accessing StyleVisionAPI ............................................................................................... 1406
21.3.6 Object Reference .............................................................................................................. 1407
    StyleVisionCommand ........................................................................................................... 1408
    StyleVisionCommands ......................................................................................................... 1410
    StyleVisionControl ............................................................................................................. 1411
    StyleVisionControlDocument ............................................................................................... 1417
    StyleVisionControlPlaceHolder ............................................................................................ 1423
    Enumerations ....................................................................................................................... 1426

22 Appendices ........................................................................................................................... 1428
22.5.4  Altova End User License Agreement ........................................ 1544

Index 1557
Chapter 1

Altova StyleVision 2015 Enterprise Edition
1 Altova StyleVision 2015 Enterprise Edition


A StyleVision Power Stylesheet (SPS) can be used for the following purposes:

- To control a graphical WYSIWYG view of XML documents in Authentic View, which is an XML document editor available in the following Altova products: Altova XMLSpy, Altova StyleVision, Altova Authentic Desktop, and Altova Authentic Browser. It enables you to easily create electronic forms based on XML documents.
- To enable the editing of databases (DBs) via Authentic View and to generate database reports in HTML, RTF, PDF, and Word 2007-and-higher format.
- To generate XSLT stylesheets based on the SPS design. (XSLT 1.0, XSLT 2.0, and XSLT 3.0 are supported.) The XSLT stylesheets can be used outside StyleVision to transform XML documents into outputs such as HTML, RTF (Rich Text Format, used by word processing applications such as MS Word), XSL-FO (used to generate PDF and PostScript files), and Word 2007-and-higher.
- To generate, directly from within StyleVision, HTML, RTF, XSL-FO, PDF, and Word 2007-and-higher output from an XML document. In the case of DB-based SPSs, StyleVision can additionally generate, for each SPS, an XML Schema based on the DB and an XML instance document that adheres to this schema and contains data from the DB.

StyleVision also enables you to import an HTML document and create an XML document from it.

Altova website: Stylesheet Designer, XSLT Designer
Last updated: 02-24-2015
Chapter 2

About this Documentation
2 About this Documentation

This documentation is the user manual delivered with StyleVision. It is available as the built-in Help system of StyleVision, can be viewed online at the Altova website, and can also be downloaded from there as a PDF, which you can print.

The user manual is organized into the following sections:

- **An introduction**, which explains what an SPS is and introduces the main features and concepts of StyleVision.
- A **description of the user interface**, which provides an overview of the StyleVision GUI.
- A **tutorial** section, which is a hands-on exercise to familiarize you with StyleVision features.
- **Usage Overview**, which describes usage at a high level: for example, schema sources used to create an SPS, the broad design process, Authentic View deployment, and projects.
- **SPS File Content**, which explains how static (stylesheet-originated) and dynamic (XML document-originated) components are created and edited in the SPS.
- **SPS File Structure**, which shows how an SPS file can be structured and modularized, and describes the handling of StyleVision's templates.
- **SPS File Advanced Features**, which describes advanced design features, such as the automatic generation of calculations, the setting up of conditions, grouping and sorting on user-defined criteria, and how to build tables of contents and cross-references in the output document.
- **SPS File Presentation**, which explains how SPS components are formatted and laid out.
- **SPS File Additional Editing Functionality**, which describes a range of additional features that can make your SPS more powerful. These features include: global resources for leveraging functionality in other Altova products, additional validation, scripts, and variables and parameters.
- **SPS File and Databases**, which explains how databases can be used with SPSs.
- **SPS File and XBRL**, which describes how output can be designed for XBRL documents in StyleVision's XBRL View.
- **Authentic View**, which describes how XML documents are edited in Authentic View. The StyleVision GUI contains an Authentic View preview tab, in which you can immediately test the Authentic View output.
- **Automated Processing**, which explains how the generation of output files can be automated.
- **StyleVision's integration features**, which contains the documentation for integrating StyleVision in other applications. There is also information on how to use StyleVision in Visual Studio and Eclipse.
- A **reference** section containing descriptions of all symbols and commands used in StyleVision.
- **Appendices** containing information about the Altova XSLT Engine information and the conversion of DB datatypes to XML Schema datatypes; technical data about StyleVision; and license information.

How to use

We suggest you read the Introduction, User Interface and Usage Overview sections first in order to get an overview of StyleVision features and general usage. Doing the tutorial next would provide
hands-on experience of creating an SPS. The SPS File sections (SPS File Content, SPS File Structure, SPS File Advanced Features, SPS File Presentation, SPS File Additional Functionality, SPS File and Databases) provide detailed descriptions of how to use various StyleVision features. For subsequent reference, the Reference section provides a concise description of all toolbar icon, design symbols, and menu commands, organized according to toolbar and menu. The Authentic View section provides information about editing in Authentic View.

File paths in Windows XP, Windows Vista, Windows 7, and Windows 8

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.

  | Windows XP | C:\Documents and Settings\<username>\My Documents |
  | Windows Vista, Windows 7/8 | C:\Users\<username>\Documents |

- **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.

  | Windows XP | C:\Program Files\Altova\ |
  | Windows Vista, Windows 7/8 | C:\Program Files\Altova\ |
  | 32 bit Version on 64-bit OS | C:\Program Files (x86)\Altova\ |

**Note**: StyleVision is also supported on Windows Server 2003, Windows 2008, and Windows Server 2012.

Support options

Should you have any question or problem related to StyleVision, the following support options are available:

1. Check the Help file (this documentation). The Help file contains a full text-search feature, besides being fully indexed.
2. Check the FAQs and Discussion Forum at the Altova Website.
3. Contact Altova’s Support Center.

Commonly used abbreviations
The following abbreviations are used frequently in this documentation:

- **SPS**: StyleVision Power Stylesheet
- **DB**: Database
- **CSS**: Cascading Style Sheets
- **FAQ**: Frequently Asked Questions
Chapter 3

New Features: Version 2015
3 New Features: Version 2015

Version 2015 Release 3
New features and updates in StyleVision Version 2015 Release 3 are listed below.

- Support for XPath/XQuery Functions and Operators 3.1.
- The Edit XPath Expression Dialog has improved operator/expression/function descriptions and entry mechanisms.
- Support for XBRL table linkbase in SPSs based on XBRL taxonomies.

Version 2015
New features and updates in StyleVision Version 2015 are listed below.

- Bug fixes
  - A footnote design component has been introduced that enables footnotes to be inserted in the design. In paged media output, footnote numbers are inserted at these locations and the corresponding footnote texts are automatically inserted at the bottom of the respective pages. Footnote numbering is automatic.
  - The StyleVisionBatch utility has been discontinued. It's functionality is now available in StyleVision Server.
  - Command line access to the StyleVision executable's XSLT-file-generation functionality.
  - Output of HTML <body> element descendants. HTML can be output without the containing html, head, and body elements. As a result, the output documents can be fragments of HTML code.
3.1 **Version 2014**

**Version 2014 Release 2**
StyleVision **Version 2014 Release 2** introduces **XSLT 3.0 functionality** and **validation for XML Schema 1.1**. A range of new **Altova extension functions** for XSLT and XPath/XQuery have also been added.

---

**Version 2014**
New features and updates in StyleVision **Version 2014** are listed below.

- **Bug fixes**
- When defining the page formats of print output documents, it can be specified whether the Initial Document Section must be followed by a page break or not. Sometimes it is useful to start the second document section directly after the Initial Document Section (that is, without a page break in between). An example of such a situation is one in which the Initial Document Section contains templates that do some XML document processing without generating any output. A page break generated after such an Initial Document Section would produce output with its first page blank. The **page layout properties** of the Initial Document Section, therefore, has a property called **Render a Section Break** which allows you to specify whether the Initial Document Section is followed by a page break or not.

- **Integration in Eclipse 4.3**. This extends support to the latest version of the Eclipse platform, which is in addition to support for versions 3.8 and 4.2. Support for Eclipse 3.6 and 3.7 have been discontinued.
3.2 Version 2013

Version 2013 Release 2

*StyleVision Version 2013* Release 2 introduces the ability to deploy PXF files to Altova FlowForce Server. After deployment to FlowForce Server, jobs can be created on FlowForce Server to run transformations according to user-defined triggers, such as at specific times every day.

Version 2013

Features that are new in *StyleVision Version 2013* are listed below.

- **Edit XPath Expression Dialog enhancement**: This dialog, which is used to enter XPath expressions in a number of design components, such as Auto-Calculations and conditional templates, has been improved. It now enables users to evaluate XPath expressions directly in the dialog. The dialog therefore now has two modes: a *Builder mode* and an *Evaluator mode*. Additionally, in Builder mode, operators and functions can be organized by functionality.
- **Conditional processing in tables** can be set on individual columns and rows of static and dynamic tables, as well as on column and row headers. Individual columns, rows, and headers can be displayed or hidden depending on the truth of the condition. If the condition evaluates to true, the column, row, or header is displayed. Otherwise it is not.
- **Watermarks** can be added to print-output pages. A watermark is an image or text that is displayed on the background of each page of a document section.
- **RichEdit for paragraphs in Authentic View**: The previous release introduced the RichEdit feature. Using this feature, elements in the design can be created as RichEdit components. The Authentic View user can select text fragments within these elements and assign one or more formatting properties from among those properties defined in the RichEdit template. These properties (from the previous release onwards) are: font, font-weight, font-style, font-decoration, font-size, color, background color and alignment of text. In the current release, RichEditing has been extended to cover block-level formatting. Text in paragraph-type blocks can now be aligned and justified using block-level formatting.
- In Design View, the **Formatting toolbar** has been extended to include font family, font size, foreground and background color, and the strike-through style. This allows stylesheet designers to quickly format text content in Design View.
- **Bug fixes**
3.3 Version 2012

**Version 2012 Release 2**
Features that are new in StyleVision Version 2012 Release 2 are listed below.

- **RichEdit for Authentic View**: Elements in the design can be created as RichEdit components, allowing Authentic View users to mark text fragments within that element and style it using the RichEdit styling properties of Authentic View. RichEdit enables Authentic View users to specify the font, font-weight, font-style, font-decoration, font-size, color, background color and alignment of text.
- **Replace Parent Node OnClick With**: The value of the parent node of a button or hyperlink can be selected by the Authentic View user. The SPS can be designed to modify presentation based on what the Authentic View user selects.
- **Support for IBM DB2 logical files**: A logical file in IBM iSeries editions of the DB2 database represents one or more physical files. A logical file allows users to access data in a sequence or format that can be different from the physical file. Users who connect to IBM iSeries computers may encounter existing databases constructed with logical files. These were previously not accessible, but are now supported in Version 2012 Release 2.
- **Bug fixes**.

---

**Version 2012 Release 1**
Features that are new in StyleVision Version 2012 Release 1 are listed below.

- **Composite Styles**: Composite Styles enable you to combine multiple CSS style declarations in a single style rule and to apply this combined style rule to the most frequently used design components.
- **Switching among external CSS Stylesheets**: You can choose between using all rules in all the external CSS stylesheets associated with the SPS (in their cascading order of precedence), or the rules in one of the external CSS stylesheets.
- **Additional support for class selector**: Dynamic selection via XPath expressions of class-selector values has been extended to RTF, PDF, and Word 2007+ output.
- **New SPS based on an SPS Module**: A new SPS file can created that will contain an empty main template and the selected SPS file as a module.
- **Find in Projects**: Enables project files and folders to be located quickly by searching for text strings in their names.
- **Support for JDBC connections to databases**.
- **New Application API interface for Java**.
3.4 Version 2011

Version 2011 Release 3
Features that are new in StyleVision Version 2011 Release 3 are listed below.

- **Internet Explorer 9 Compatibility**: StyleVision offers support for Internet Explorer 9 (IE 9). CSS styles and HTML5 elements that are supported by IE 9 are supported in the SPS design interface. Authentic View and HTML Preview will show the IE 9 rendering of the output document. In the Properties dialog, you can modify the IE-compatibility of the SPS.
- **Improved support for image formats**: Better support for the TIFF, SVG, and JPEG XR formats.
- **PXF file support**: The PXF file format has been specially developed by Altova to package the SPS design with related files (such as the schema file, source XML file, image files used in the design, and XSLT files for transformation of the source XML to an output format). The benefit of the PXF file format is that all the files required for Authentic View editing and for the generation of output from Authentic View can be conveniently distributed in a single file.
- **XML Signatures**: An SPS can have an XML signature configured for it. When this is done, the Authentic View user can sign the Authentic XML document based on that SPS. The signature of signed documents can also be verified in Authentic View. This enables Authentic users to transport their documents with a built-in security measure.

Version 2011 Release 2
Features that are new in StyleVision Version 2011 Release 2 are listed below.

- **Multiple Document Output**: The output generated by the SPS can be designed to be split into multiple documents. In the design, New Document templates are created and content placed in them. Each New Document template generates a separate document in the output.
- **User-Defined XPath Functions**: The user can define XPath functions which can be used anywhere in the document where XPath functions may be used.
- **Images from inline data**: Images can be generated from Base-16 and Base-64 encoded text in the XML document. Consequently, images can be stored directly in the source XML document as text. An SPS can now decode such text and render the image.
- **ASPX Interface for Web Applications**: With this feature, HTML web pages can be quickly updated. StyleVision generates, from an SPS, all the files necessary for an ASPX application. When the web page (a .aspx file) is refreshed, the source data (including any updates) is dynamically transformed via XSLT to the web page.
- **Combo Boxes**: The design capability for combo boxes has been extended. The visible values in the dropdown list of the combo box as well as the corresponding values placed in the XML file can be separately specified.
- **PDF Bookmarks**: A simpler mechanism to generate bookmarks in PDF output. In previous versions, the Table of Contents was used as the template for generating PDF bookmarks. While the TOC mechanism is a powerful one and is very useful for complex structures, the new PDF Bookmarks alternative simplifies the creation of non-complex bookmark tree structures.
- **Charts**: The following additional chart types are supported: stacked bar charts, area charts, stacked area charts, candlestick charts, and overlay charts.
• **Barcodes**: This new design component enables barcodes to be easily inserted in the design. Barcode images are generated on the fly and placed in the output documents.

• **Conditional presence**: Certain design components have a conditional presence property. A conditional design component will be created in the output only if the condition specified for it is fulfilled.

---

**Version 2011 Release 1**

Features that are new in *StyleVision Version 2011 Release 1* are listed below.

• **CALS/HTML Tables**: The XML Tables feature of earlier versions has been improved and renamed to **CALS/HTML Tables**. If a CALS or HTML table structure is defined in the SPS design's DTD or schema, you can specify in the design that CALS/HTML tables should be processed. These table structures, if present in the XML instance file, will then be correctly sent to the output as tables.

• **Charts**: This SPS design element provides a graphical representation of data in the source XML document. It is set up by specifying a sequence of items for each axis of the chart. StyleVision then automatically generates the chart in the various output formats.

• **XBRL Charts**: Data from XBRL instance files can be represented as charts in the various output formats.

• **New from XSLT**: An SPS can be created from an XSLT-for-HTML or an XSLT-for-FO. Template structure and styling in the XSLT will be created in the SPS. You can then modify the SPS components and add content and formatting to the SPS.

• **Integration in Microsoft Visual Studio 2010**: This extends support to the latest version of Visual Studio, which is in addition to support for versions 2005 and 2008. Support for Visual Studio 2003 has been discontinued.

• **Integration in Eclipse 3.6**: This extends support to the latest version of the Eclipse platform, which is in addition to support for versions 3.4 and 3.5. Support for Eclipse 3.3 has been discontinued.

• **Authentic Scripting**: Enables additional flexibility and interactivity in Authentic View.
3.5 Version 2010

Version 2010 Release 3
Features that are new in StyleVision Version 2010 Release 3 are listed below.

- **Value Formatting (Formatting Numeric Datatypes):** The earlier Input Formatting mechanism has been extended to enable—only in the Enterprise Edition—the formatting of Inline XBRL values when they are output in an (X)HTML report. The older Input Formatting feature remains unchanged but has been renamed to Value Formatting.
- **Inline XBRL:** The SPS design can generate Inline XBRL elements in the (X)HTML output. These Inline XBRL elements can subsequently be extracted by an external processor that generates valid XBRL documents from (X)HTML documents.
- **Global templates** can now be created for any node or type in the schema. In earlier versions of StyleVision, global templates could only be created for global elements and global types. They can now be created on any node or type, and even for any item returned by an XPath expression.
- **Integration in Microsoft Visual Studio 2010.** This extends support to the latest version of Visual Studio, which is in addition to support for versions 2005 and 2008. Support for Visual Studio 2003 has been discontinued.

---------------------------------------------------------------

Version 2010 Release 2
Features that are new in StyleVision Version 2010 Release 2 are listed below.

- Enterprise and Professional editions are each available as separate 64-bit and 32-bit applications.
- **Editable Variables in Authentic** enable the Authentic View user to edit variables and consequently to control the display of content in Authentic View.
- **Parameters for Design Fragments** allow design fragments to be used with different parameter values for each usage instance. A different parameter value can be assigned to a design fragment at each location where the design fragment is used in the SPS.
- **First and last page headers and footers** can be specified separately. This is in addition to different headers and footer for odd-numbered and even-numbered pages and for different document sections.
- **Layout Boxes** and **Lines** can be moved and resized using the keyboard.
- **Templates around table rows or columns** can be added or deleted without modifying the content or formatting of the row or column involved.
- **Text in tables** and **in layout boxes** can be rotated clockwise or anti-clockwise so that it is vertical.
- **Filters can be set on global templates** where these are used in the main template.
- Design fragments can be dragged from the **Schema Tree**, in addition to being available in the **Design Tree**.

---------------------------------------------------------------

Version 2010 Release 1
Features that are new in StyleVision Version 2010 Release 1 are listed below. Some of these new features have required a modification in the way older features are handled. In such cases,
the existing feature continues to behave as before, but uses one or more of the newer mechanisms. The way a new feature affects existing features is noted in the list below.

- **Layout Containers**: A Layout Container is a block in which Design Elements can be laid out and absolutely positioned within the block.
- **Blueprints**: Within a Layout Container an image of a form can be used as an underlay blueprint for the design. With the help of a blueprint, an existing design can be reproduced accurately.
- **Document Sections**: Documents can be divided into sections, with each section having its own properties, such as page layout properties. This enables different parts of a document to be presented differently. *Older features affected: Previous designs had no sections. These designs will now be created as documents with one section, the Initial Document Section. Page properties and page layout properties, which were previously specified for the document as a whole, are now specified for the Initial Document Section. The cover page for print output of previous versions will be created in the new version as a template within the Initial Document Section.*
- **Page columns**: Pages can be specified to have columns.
- **User-Defined Templates**: A template can be generated for a sequence of items by an XPath expression you specify. These items may be atomic values or nodes. An XPath expression enables the selection of nodes to be more specific, allowing conditions and filters to be used for the selection. Furthermore, templates can be built for atomic values, thus enabling structures to be built that are independent of the schema structure. *Older features affected: Variable Iterators, which were used to create a template for a variable, now create a variable on a node template and then a User-Defined template for that variable.*
- **User-Defined Elements**: This feature is intended to enable presentation language elements (such as HTML, XSLT, and XSL-FO) to be freely inserted at any location in the design.
- **User-Defined XML Text Blocks**: XML Text blocks can be freely inserted at any location in the design, and these blocks will be created at that location in the generated XSLT stylesheet.
- **XSLT Templates**: XSLT files can be imported into the generated stylesheets. If a node in the XML instance document is matched to a template in the imported XSLT file and no other template takes precedence over the imported template, then the imported template will be used. Additionally, named templates in the imported XSLT file can be called from within the design.
- **Variables**: A variable can now be declared on a template and take a value that is specified with an XPath expression. Previously, the value of a variable was limited to the selection of the node on which it was created. Variables in the 2010 version allow any XPath expression to be specified as the value of the variable. *Older features affected: Variables and Variable Iterators. Variables from older versions are now created on the relevant template and are given a value that selects the same template. Variable Iterators are replaced with a combination of a Variable and a User-Defined Template; see User-Defined Templates below.*
- **Inserting Design Elements**: Design Elements (paragraphs, lists, images, etc) can be inserted first, and an XML node from the schema tree assigned to the Design Element afterwards. This is in addition to the existing mechanism by which a schema nodes is dragged into the design and a Design Element created for it.
- **Hide Markup in Design View**: Markup tags in Design View can be hidden and collapsed, thus freeing up space in Design View.
- **Java and .NET functions in Auto-Calculations**: Native Java and .NET functions can be used in the XPath expressions of Auto-Calculations. This makes the function libraries of Java and .NET accessible to the SPS.
- **Disable output escaping**: A setting that defines whether text output will be escaped or not. A character is said to be escaped when it is written as a character entity (such as `&amp;` or `&#65;`). This feature is useful when outputting text that contains program code.
- **Pixel Resolution**: Pixel length units in the SPS are converted to absolute units for print output according to a factor that the SPS designer specifies.
- **Default length units**: can be specified in the Options dialog (Tools | Options).
- **XHTML output**: When XHTML is specified as the HTML output preference in the document's properties (File | Properties), an XHTML document is generated for the HTML output.
- **Printout of Design**: The design in Design View can be printed with or without tags.
Chapter 4

Introduction
4 Introduction

This section introduces you to Altova® StyleVision® 2015. It consists of the following sub-sections:

- **What Is an SPS?**, which explains the role of an SPS in an XML environment and with respect to StyleVision.
- **Product Features**, which provides an overview of the key features of StyleVision.
- **Terminology**, which lists terms used in the StyleVision user interface and in this documentation.
- **Setting up StyleVision**, which describes how StyleVision is to be correctly set up.

See also

- User Interface
- General Usage Procedure
4.1 What Is an SPS?

A StyleVision Power Stylesheet (or SPS) is an extended XSLT stylesheet which is used:

- to control the display and entry of data in the Authentic View of XML documents and databases (DBs); and
- to specify the output design of an XML document transformation.

An SPS is saved with the file extension .sps.

Design of the SPS

An SPS is created graphically in StyleVision. It is based on a schema (DTD or XML Schema); if the SPS is to be used with a DB, it is based on an XML Schema generated automatically by StyleVision from the DB structure. The design of the SPS is flexible. It can contain dynamic and static content. The **dynamic content** is the data in one or more XML documents or DBs. The **static content** is content entered directly in the SPS. Dynamic content can be included in the design either as straight text or within components such as input fields, combo boxes, and tables. Additionally, dynamic content can be manipulated (using Auto-Calculations) and can be displayed if certain conditions in the source document are fulfilled. Different pieces of content can be placed at various and multiple locations in the SPS. Also, the SPS can contain various other components, such as images, hyperlinks, and JavaScript functions. Each component of the SPS can then be formatted for presentation as required.

The SPS and Authentic View

When a finished SPS is associated with an XML document or DB, that XML document or DB can be edited in Authentic View. Authentic View is an ideal solution for enabling the distributed and graphical editing of an XML document or DB. Multiple users can edit an XML document or DB in the graphical user interface presented by Authentic View. In StyleVision, as you design an SPS, you can preview and test the SPS (in the Authentic View tab for that SPS). For a detailed description of how SPSs work with Authentic View, see SPS and Authentic View.

The SPS and XSLT stylesheets

After you have completed designing the SPS, you can generate XSLT stylesheets based on the design you have created. StyleVision supports XSLT 1.0, XSLT 2.0 and XSLT 3.0, and from a single SPS, you can generate XSLT stylesheets for HTML, RTF, XSL-FO, and Word 2007-and-higher output (XSL-FO and Word 2007-and-higher in Enterprise edition only; RTF in Enterprise and Professional Editions; in Basic Edition only HTML output is supported). The generated XSLT stylesheets can be used in external transformations to transform XML documents based on the same schema as the SPS from which the XSLT stylesheet was generated. For more information about procedures used with XSLT stylesheets, see the section Generated Files.
The SPS and output
You can also use StyleVision to directly generate output (HTML, RTF, XSL-FO, and PDF in Enterprise Edition; HTML in Professional and BasicEditions). The tabs for Output Views display the output for the active SPS document directly in the StyleVision GUI. The required output can also be generated to file from within the GUI via the File | Save Generated Files command or via StyleVision Server.
Authentic View in Altova Products
Authentic View is a graphical XML document editor available in the following Altova products:

* Altova XMLSpy
* Altova Authentic Desktop
* Altova Authentic Browser
* Altova StyleVision

See also
- General Usage Procedure
4.2 Product Features

The main product features of StyleVision are listed below:

---

**General product features**
Given below is a list of the main high-level features of StyleVision.

- Enterprise and Professional editions are each available as separate 64-bit and 32-bit applications.
- StyleVision functionality can be called via [StyleVision Server](#).
- StyleVision functionality can be integrated in external applications, and StyleVision an integrated in [Visual Studio](#) and [Eclipse](#).

---

**Sources**
SPS designs can be based on XML Schemas and DTDs. A design uses other source files, such as XML and CSS files. The following additional features concerning sources are supported:

- [Altova Global Resources](#) can be used to locate source files such as schema, XML, and CSS. The Global Resources mechanism enables faster and better development and testing by allowing developers to quickly change source data and to use the functionality of other Altova applications from within StyleVision.
- The SPS can use multiple schemas and/or a single database (DB) source, thus enabling the use of multiple XML document sources together with a DB source.
- XBRL taxonomies can be loaded and an SPS can be built based on the taxonomy.
- HTML documents can be converted to XML.

---

**Interface**
Given below are some general GUI features:

- [Multiple SPS designs](#) can be open simultaneously, with one being active at any given time. Each SPS design is shown in a separate tab.
- [Template filters](#) allow you to customize the display of the design document. With this feature you can disable the display of templates that are not currently being edited, thus increasing editing efficiency.
- [Hide Markup in Design View](#): Markup tags in Design View can be hidden and collapsed, thus freeing up space in Design View.
- While designing the SPS, [Authentic View](#), output views and stylesheets can be displayed by clicking the respective tabs. This enables you to quickly preview the output and the XSLT code, and test Authentic View features.
- When an SPS is associated with an [XML source document](#) or [source DB](#), the source document can be edited directly in the Authentic View of StyleVision.


**Databases**
The following DB features are supported:

- [DB reports](#) can either be viewed in StyleVision or saved as HTML, RTF, PDF, and Word 2007-and-higher files.
- IBM DB2 databases, which contain XML columns, are supported.
- A DB can be queried directly from StyleVision.

---

**Output**

Various output formats are supported depending upon the edition that has been installed. The following output-related features are supported:

- [XSLT versions 1.0, 2.0, and 3.0](#) are supported.
- In the Enterprise and Professional Editions, multiple output formats (HTML, RTF, PDF, and Word 2007-and-higher) are generated from a single SPS design.
- Conditions can be set on SPS components to process them differently for different outputs. With this level of granularity, different outputs can be flexibly structured to take in the requirements of that particular output.
- Both [XSLT files and output files](#) can be generated and saved, either directly from within the GUI or via [StyleVision Server](#).
- Altova has developed a special [PXF File format](#) that enables an SPS file to be saved together with related source and data files. This enables entire SPS projects to be transported rather than just the SPS file.
- [ASPX Interface for Web Applications](#): With this feature, HTML web pages can be quickly updated. StyleVision generates, from an SPS, all the files necessary for an ASPX application. When the web page (a .aspx file) is refreshed, the source data (including any updates) is dynamically transformed via XSLT to the web page.

---

**SPS design features**

Given below is a list of the main StyleVision features specific to designing the SPS.

- The SPS can contain [static text](#), which you enter in the SPS, and [dynamic text](#), which is selected from the [source document/s](#).
- [Dynamic content](#) is inserted in the design by dragging-and-dropping nodes from the [schema sources](#). Design Elements (paragraphs, lists, images, etc) can also be inserted first, and an XML node from the schema tree can be assigned to the Design Element afterwards.
- Dynamic content can be inserted as text, or in the form of a [data-entry device](#) (such as an [input field](#) or [combo box](#)). When inserted as a data-entry device such as a combo box, additional possibilities are available. For example, the value of the node can be selected (by the Authentic View user) from a list of enumerations.
- An [XBRL Table Wizard](#) can automatically generate a dynamic SPS table according to your input.
- The [structure of the design](#) is specified and controlled in a single [main template](#). This structure can be modified by optional templates for individual elements—known as [global](#)
templates because they can be applied globally for that element.

- **Global templates** can also be created for individual datatypes, thus enabling processing to be handled also on the basis of types.

- **Multiple Document Output**: The output generated by the SPS can be designed to be split into multiple documents. In the design, New Document templates are created and content placed in them. Each New Document template generates a separate document in the output.

- **User-Defined Templates**: A template can be generated for a sequence of items by an XPath expression you specify. These items may be atomic values or nodes. An XPath expression enables the selection of nodes to be more specific, allowing conditions and filters to be used for the selection.

- **User-Defined Elements**: This feature is intended to enable presentation language elements (such as HTML, XSLT, and XSL-FO) to be freely inserted at any location in the design.

- **User-Defined XML Text Blocks**: XML Text blocks can be freely inserted at any location in the design, and these blocks will be created at that location in the generated XSLT stylesheet.

- **Design Fragments** enable the modularization and re-use of templates within an SPS, and also across multiple SPSs (see modular SPSs), in a manner similar to the way functions are used.

- **SPS modules** can be added to other SPS modules, thus making objects defined in one SPS module available to other modules. This enables re-use of module objects across multiple SPSs and makes maintenance easier.

- **XSLT Templates**: XSLT files can be imported into the generated stylesheets. If a node in the XML instance document is matched to a template in the imported XSLT file and no other template takes precedence over the imported template, then the imported template will be used. Additionally, named templates in the imported XSLT file can be called from within the design.

- **New from XSLT**: An SPS can be created from an XSLT-for-HTML or an XSLT-for-FO. Template structure and styling in the XSLT will be created in the SPS. You can then modify the SPS components and add content and formatting to the SPS.

- **User-Defined XPath Functions**: The user can define XPath functions which can be used anywhere in the document where XPath functions may be used.

- **Charts**: This SPS design element provides a graphical representation of data in the source XML document. It is set up by specifying a sequence of items for each axis of the chart. StyleVision then automatically generates the chart in the various output formats.

- **XBRL Charts**: Data from XBRL instance files can be represented as charts in the various output formats.

- **Document Sections**: Documents can be divided into sections, with each section having its own properties, such as page layout properties. This enables different parts of a document to be presented differently.

- **Layout Containers**: A Layout Container is a block in which Design Elements can be laid out and absolutely positioned within the block.

- **Blueprints**: Within a Layout Container an image of a form can be used as an underlay blueprint for the design. With the help of a blueprint, an existing design can be reproduced accurately.

- A common feature of XML documents is the repeating data structure. For example, an office department typically has several employees. The data for each employee would be stored in a data structure which is repeated for each employee. In the SPS, the processing for each such data structure is defined once and applied to each relevant node in turn (the employee node in our example).

- **Multiple tables of contents** can be inserted in XSLT 2.0 and 3.0 SPSs.

- Repeating data structures can also be inserted as **dynamic tables**. This provides looping
in a structured, table format, with each loop through the data structure producing a row (or, if required, a column) of the table.

- A repeating element can be sorted on one or more sort-keys you select, and the sorted element set is sent to the output (HTML, RTF, PDF, and Word 2007-and-higher).
- Variables: A variable can now be declared on a template and take a value that is specified with an XPath expression. Previously, the value of a variable was limited to the selection of the node on which it was created. Variables in the 2010 version allow any XPath expression to be specified as the value of the variable.
- Nodes can be grouped on the basis of common data content (for example, the common value of an attribute value) and their positions.
- The conditional templates feature enables one of a set of templates to be processed according to what conditions in the XML document or system environment are fulfilled. This enables processing that is conditional on information contained in the source document or that cannot be known to the SPS document creator at the time of creation (for example, the date of processing). The available conditions are those that can be tested using XPath expressions.
- Auto-Calculations enable you to manipulate data from the source document/s and to display the result. This is useful, when you wish to perform calculations on numbers (for example, sum the prices in an invoice), manipulate strings (for example, change hyphens to slashes), generate content, etc. The available manipulations are those that can be effected using XPath expressions. Native Java and .NET functions can be used in the XPath expressions of Auto-Calculations.
- When data is edited in Authentic View, the result of Auto-Calculations can also be passed to a node in the source document. This procedure is referred to as updating the XML node (with the value of the Auto-Calculation).
- Additional validation enables individual XML document nodes to be validated (additionally to schema validation) against an XPath expression defined for that node. In this way, Authentic View users can be alerted when the data they enter is invalid; a customized error message for the node can indicate the problem.
- Barcodes: This new design component enables barcodes to be easily inserted in the design. Barcode images are generated on the fly and placed in the output documents.
- Conditional presence: Certain design components have a conditional presence property. A conditional design component will be created in the output only if the condition specified for it is fulfilled.
- Images can be inserted in the design. The URI for the image can be static (entered in the SPS), or dynamic (taken from a node in the source document), or a combination of both static and dynamic parts.
- Images from inline data: Images can be generated from Base-16 and Base-64 encoded text in the XML document. Consequently, images can be stored directly in the source XML document as text. An SPS can now decode such text and render the image.
- Two types of lists can be created: static and dynamic. In a static list, each list item is defined in the SPS. In a dynamic list, a node is created as a list item; the values of all instances of that node are created as the items of the list.
- Static and dynamic links can be inserted in the design. The target URI can be static (entered in the SPS), or dynamic (taken from a node in the source document), or a combination of both static and dynamic parts.
- Static bookmarks can be inserted. These serve as anchors that can be linked to with a hyperlink.
- PDF Bookmarks: A simpler mechanism to generate bookmarks in PDF output. In previous versions, the Table of Contents was used as the template for generating PDF bookmarks. While the TOC mechanism is a powerful one and is very useful for complex structures, the new PDF Bookmarks alternative simplifies the creation of non-complex bookmark tree structures.
• **Parameters** can be declared globally for the entire SPS. A parameter is declared with a name and a string value, and can be used in XPath expressions in the SPS. The parameter value you declare is the default value. It can be overridden by a value passed via StyleVision Server.

• With the **Input Formatting** feature, the contents of numeric XML Schema datatype nodes can be formatted as required for Authentic View display and, in the case of some formats, optionally for output display. Input Formatting can also be used to format the result of an Auto-Calculation.

• **JavaScript functions** can be used in the SPS to provide user-defined functionality for Authentic View and HTML output.

• **Authentic Scripting**: Enables additional flexibility and interactivity in Authentic View.

• A number of **predefined HTML formats** are available via the GUI and can be applied to individual SPS components.

• A large number of CSS text formatting and layout properties can be applied to individual SPS components via the Styles sidebar.

• Additionally, CSS styles can be defined for HTML selectors at the **global level** of an SPS and in external CSS stylesheets. These style rules will be applied to Authentic View and HTML output, thus providing considerable formatting and layout flexibility.

• **Styles can also be assigned using XPath expressions**, This enables style property values to be selected from XML documents and to set property values conditionally.

• For paged output (typically RTF, PDF, and Word 2007-and-higher), a number of **page layout options**, such as orientation, margins, page-numbering, and headers and footers, can be specified in the SPS.

---

**See also**

- **User Interface**
- **General Usage Procedure**
4.3 Terminology

This section lists terms used in the StyleVision GUI and in this documentation. Terms are organized into the groups listed below, and within each group, they are listed alphabetically.

---

**Altova product-related terms**

A list of terms that relate to Altova products.

*Authentic View*  
An XML document editor view available in the following Altova products: Altova XMLSpy; Altova StyleVision; Altova Authentic Desktop; Altova Authentic Browser. For more details about Authentic View and Altova products, visit the Altova website.

*SPS*  
The abbreviated form of StyleVision Power Stylesheet, it is used throughout this documentation to refer to the design document created in StyleVision and saved as a file with the .sps extension. For a detailed description, see What Is an SPS?

*Global resource*  
An alias for a set of files, a set of folders, or a set of databases. Each alias has a set of configurations and each configuration is mapped to a resource. In StyleVision, when a global resource is used, the resource can be changed by changing the active configuration in StyleVision.

---

**General XML terms**

Definitions of certain XML terms as used in this documentation.

*schema*  
A schema (with lowercase 's') refers to any type of schema. Schemas supported by StyleVision are XML Schema (capitalized) and DTD.

*XML Schema*  
In this documentation, XML Schema (capitalized) is used to refer to schemas that are compliant with the W3C’s XML Schema specification. XML Schema is considered to be a subset of all schemas (lowercased).

*URI and URL*  
In this documentation, the more general URI is used exclusively—even when the identifier has only a "locator" aspect, and even for identifiers that use the http scheme.

---

**XSLT and XPath terms**

There have been changes in terminology from XSLT 1.0 and XPath 1.0 to XSLT 2.0 and XPath 2.0. For example, what was the root node in XPath 1.0 is the document node in XPath 2.0. In this documentation, we use the newest, XSLT 3.0 and XPath 3.0, terminology.
**Introduction Terminology**

<table>
<thead>
<tr>
<th><strong>absolute XPath</strong></th>
<th>A path expression that starts at the root node of the tree containing the context node. In StyleVision, when entering path expressions in dialogs, the expression can be entered as an absolute path if you check the Absolute XPath check box in the dialog. If this check box is unchecked, the path is relative to the context node.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>context item / context node</strong></td>
<td>The context item is the item (node or string value) relative to which an expression is evaluated. A context node is a context item that is a node. The context item can change within an expression, for example, with each location step, or within a filter expression (predicate).</td>
</tr>
<tr>
<td><strong>current node</strong></td>
<td>The current node is the node being currently processed. The current node is the same as the context node in expressions that do not have sub-expressions. But where there are sub-expressions, the context node may change. Note that the current() function is an XSLT function, not an XPath function.</td>
</tr>
<tr>
<td><strong>document element</strong></td>
<td>In a well-formed XML document, the outermost element is known as the document element. It is a child of the document node, and, in a well-formed XML document, there is only one document element. In the GUI the document element is referred to as the root element.</td>
</tr>
<tr>
<td><strong>document node</strong></td>
<td>The document node represents and contains the entire document. It is the root node of the tree representation of the document, and it is represented in an XPath expression as: '/'. In the Schema Tree window of StyleVision, it is represented by the legend: '/ Root elements'.</td>
</tr>
</tbody>
</table>

**StyleVision-specific terms**

Terms that refer to StyleVision mechanisms, concepts, and components.

<table>
<thead>
<tr>
<th><strong>Blueprint image</strong></th>
<th>A blueprint image is one that is used as the background image of a layout container, and would typically be the scan of a form. The SPS design can be modelled on the blueprint image, thus recreating the form design.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dynamic items</strong></td>
<td>Items that originate in XML data sources. Dynamic items may be text, tables, and lists; also images and hyperlinks (when the URIs are dynamic).</td>
</tr>
<tr>
<td><strong>global element</strong></td>
<td>An element in the Global Elements list in the Schema Tree window. In an XML Schema, all elements defined as global elements will be listed in the Global Elements list. In a DTD, all elements are global elements and are listed in the Global Elements list. Global templates can be defined only for global elements.</td>
</tr>
<tr>
<td><strong>global template</strong></td>
<td>A global template may be defined for a global element. Once defined, a global template can be used for that element wherever that element occurs in the document. Alternatively to the global template, processing for a global element may be defined in a local template.</td>
</tr>
<tr>
<td><strong>Layout container</strong></td>
<td>A Layout Container is a design block in which design elements can be laid out and absolutely positioned. If a design is to be based on a form, it can be created as a Layout Container, so that design elements of the form can be absolutely positioned. Alternatively, a design can be free-flowing and have layout containers placed within the flow of the document.</td>
</tr>
</tbody>
</table>
| **local** | A local template is the template that defines how an element (global or non-global)
**Template** is processed within the **main template**. The local template applies to that particular occurrence of the element in the **main template**. Instead of the local template, a **global template** can be applied to a given occurrence of an element in the **main template**.

One of the assigned schema sources is designated the main schema; the document node of the **Working XML File** associated with the main schema is used as the starting point for the **main template**.

The main entry-point template. In StyleVision, this template matches the **document element** and is the first to be evaluated by the XSLT processor. In the Schema Tree window, it is listed as the child of the **document node**. The **main template** defines the basic output document structure and defines how the input document/s are to be processed. It can contain **local templates** and can reference **global templates**.

**Output** is the output produced by processing an **XML** document with an XSLT stylesheet. Output files that can be generated by StyleVision would be HTML, RTF, PDF, and Word 2007-and-higher format. Authentic View is not considered an output, and is referred to separately as Authentic View. XSLT stylesheets generated by StyleVision are also not considered output and are referred to separately as XSLT stylesheets.

**Static items** are items that originate in the SPS and not in XML data sources. Static items may be text, tables, and lists; also images, hyperlinks, and bookmarks (when the URIs are static).

An SPS component can be: (i) a schema node (for example, an element node); (ii) a static SPS component such as an **Auto-Calculation** or a text string; or (iii) a predefined format (represented in the SPS by its start and end tags).

**Template XML File** is assigned to an SPS in StyleVision (Enterprise and Professional editions). It is an XML file that provides the starting data of a new XML document created with a given SPS when that SPS is opened in Authentic View. The Template XML File must be conformant with the schema on which the SPS is based.

An element that is neither a node in the schema tree nor a predefined element or a design element, but one that is specified by the user. An element can be specified with attributes.

**User-defined template** is a template that is created for a sequence specified in an XPath expression.

**User-defined XML text blocks** are XML Text blocks can be freely inserted at any location in the design.

A **Working XML/XBRL File** is an XML data file that is assigned to an SPS in StyleVision in order to preview the Authentic View and output of the XML document in StyleVision. Without a Working XML/XBRL File, the SPS in StyleVision will not have any dynamic XML data to process. If the SPS is based on a schema that has more than one global element, there can be ambiguity about which global element is the document element. Assigning a Working XML/XBRL
File resolves such ambiguity (because a valid XML document will, by definition, have only one document element). Note that XBRL functionality is available only in the Enterprise edition.

**XML** document is used in two senses: (i) to refer to a specific XML document; (ii) to refer to any XML data source, including DB sources (from which XML data documents are generated for use with an SPS). Which sense is intended should be clear from the context.

**See also**
- What Is an SPS?
- General Usage Procedure
4.4 Setting up StyleVision

Altova StyleVision runs on Windows XP, Windows Vista, and Windows 7. After downloading StyleVision from the Altova website, double-click the executable (.exe) file to run the setup program. The setup program will install StyleVision at the desired location. The Altova XSLT Engines (1.0 and 2.0) are built into StyleVision and are used for all internal transformations. You, therefore, do not need to install an XSLT Engine additionally to your StyleVision installation. The setup program will also install the Apache FOP processor (for the generation of PDF from XSL-FO) in the folder `C:\ProgramData\Altova\SharedBetweenVersions`.

You will, however, need to have the following components installed:

- Internet Explorer 5.5 or later, for Authentic View, HTML Preview and Design View. Internet Explorer 6.0 and later has better XML support and is recommended.
- Microsoft Word 2000 or later, for RTF Preview. Microsoft Word 2003 or later is recommended.
- Adobe Acrobat Reader, for PDF Preview (Acrobat Reader 6.0 or later is recommended).

Note: In this documentation, we use the abbreviation **Word 2007+** to refer to Microsoft Word 2007 or higher versions of this program.

Note: If there is a problem with an embedded preview, StyleVision will attempt to open the preview document in an external application (usually MS Word or Adobe Reader). An error message about the embedded preview will appear in StyleVision. If the preview document is opened in an external application, you will need to close the external application before regenerating the temporary output document, otherwise you will get an error message saying the file is being used by another process. You should also close the external application before closing the SPS design, otherwise StyleVision will not be able to close the temporary output document due to the file lock placed on the document by the external application.

---

**Note on FO processors**

StyleVision can pass the XSL-FO document (generated by transforming the source document) to an FO processor of your choice. In the FO Processor input field of the Options dialog (**Tools | Options | XSL-FO**) (screenshot below), enter the location of the file that starts the FO processor. The FO processor you select is used to generate the PDF Preview as well as the PDF files created via the **File | Save Generated Files** command.
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 FO Processor input field (see screenshot above). 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 (XMLSpy and Authentic Desktop).

After you have installed the FO processor, you should set the XSL-FO compliance level in the Options dialog (Tools | Options | XSL-FO).

**Note:** Since conformance levels between any two FO processors can differ widely, and since the XSL-FO specification may be interpreted differently from processor to processor, the PDF produced by different processors from a single FO file could be different.

**Note:** When a large document is processed by FOP (Apache's FO processor), data in memory is discarded each time a new document section is processed. If you wish to reduce memory use, therefore, it is best to structure your document into multiple document sections.

---

**RaptorXML**

RaptorXML can be used to transform XML to the required output format.
See also

- Generated Files
- General Usage Procedure
- Automated Processing
4.5 Authentic View in Altova Products

Authentic View is a graphical XML document editor available in the following Altova products:

* Altova XMLSpy
* Altova Authentic Desktop
* Altova Authentic Browser
* Altova StyleVision

In StyleVision, Authentic View can be viewed in the Authentic eForm tab of the Main Window.

Enterprise editions of Authentic View applications

The following SPS functionality is enabled only in the Enterprise editions of Altova's Authentic View applications:

- Absolute positioning (layout containers)
- Java and .Net function calls from XPath expressions in Auto-Calculations
- Variables
- User-Defined Elements and XML Text Blocks

If any of this functionality is present in an SPS that is opened in a non-Enterprise edition of an Authentic View application (say, XMLSpy Professional Edition), then the application displays a message saying that this functionality is available only in the Enterprise edition of the application.


See also

- Product Features
- New Features
Chapter 5
User Interface
5 User Interface

The StyleVision GUI (illustration below, in which not all sidebars are shown) consists of the following parts:

- A **menu bar**. Click on a menu to display the items in that menu. All menus and their items are described in the **User Reference** section. The menu bar also contains the Minimize, Restore, and Close Active Document buttons.
- A **toolbar area**. The various toolbars and the command shortcuts in each toolbar are described in the **User Reference** section.
- A tabbed **Main Window**, which displays one or more open SPS documents at a time. In this window, you can edit the design of the SPS, edit the content of Authentic View (in the Authentic eForm tab), and preview the XSLT stylesheets and output.
- The **Design Sidebars**—the Design Overview, Schema Tree, Design Tree, Style Repository, Styles, Properties, and Project windows—which can be docked within the application GUI or made to float on the screen.
- A **status bar**, which displays application status information. If you are using the 64-bit version of StyleVision, this is indicated in the status bar with the suffix (x64) after the application name. There is no suffix for the 32-bit version.

The **Main Window** and **Design sidebars** are described in more detail in the sub-sections of this section.

**Note**: The menu bar and toolbars can be moved by dragging their handles to the required location.

See also

- **Menu Commands section**
- **Reference section | Toolbars**
5.1 Main Window

The Main Window (illustration below) is where the SPS design, Authentic View, XSLT stylesheets, and output previews are displayed.

SPS documents in the Main Window
- Multiple SPS documents can be open in StyleVision, though only one can be active at any time. The names of all open documents are shown in tabs at the bottom of the Main Window, with the tab of the active document being highlighted.
- To make an open document active, click its tab. Alternatively, use the options in the Windows menu.
- If so many documents are open that all document tabs are not visible in the document-tab bar, then click the appropriate scroll button (at the right of the document-tab bar; see illustration above) to scroll the tabs into view.
- To close the active document, click the Close Document button in the menu bar at the top right of the application window (or select File | Close).

Document views
A document is displayed in the following views, one of which can be active at a time:
- **Design View**, in which you design the SPS and edit JavaScript functions for use in that SPS. The view can be toggled between the design document and the JavaScript Editor by clicking the dropdown menu arrow and selecting Design or JavaScript, as required.
- **Authentic View** (Authentic eForm tab), which enables you to immediately see the Authentic View of an XML document (the Working XML File). The SPS is dynamically applied to the Working XML File, thus enabling you to try out Authentic View.
- **Output Views** (HTML, RTF, PDF, and Word 2007+ output). These views are a preview of the actual output format and of the XSLT stylesheet used to generate that output. The view can be toggled between the output preview and the XSLT stylesheet by clicking the dropdown menu arrow and making the appropriate selection.
Each of the views listed above is available as a tab at the bottom of the Main Window in the Views Bar. To select a view, click on its tab. The tab of the selected view is highlighted.

- **See also**
  - Design sidebars
Design View

The Design View (illustration below) is the view in which the SPS is designed. In Design View, you create the design of the output document by (i) inserting content (using the sidebars, the keyboard, and the various content creation and editing features provided in the menus and toolbars); and (ii) formatting the content using the various formatting features provided in the sidebars and menus. These aspects of the Design View are explained in more detail below.

Design View can also be switched to a JavaScript Editor or (Authentic) Scripting Editor. In the JavaScript Editor you can create and edit JavaScript functions which then become available in the GUI for use in the SPS. In the Scripting Editor you can created and edit scripts for Authentic View. To switch to the JavaScript Editor or (Authentic) Scripting Editor, click the dropdown button in the Design tab (see illustration) and select JavaScript or (Authentic) Scripting Editor from the dropdown menu. To switch back to Design View, click the dropdown button in the JavaScript or (Authentic) Scripting Editor tab and select Design from the dropdown menu.

In Design View, the SPS can have several templates: the main template, global templates, page layout templates, and Design Fragments. You can control which of these template types is displayed in Design View by using Template Display Filters, which are available as toolbar icons. These display filters will help you optimize and switch between different displays of your SPS.

Displaying markup tags

The display of markup tags in Design View can be controlled via the markup icons (below).

The icons shown above are toggles. They are, from left: (i) Show small design markups (tags without names); and (ii) Show large design markups (tags with names). When small markup is switched on, the path to a node is displayed when you mouseover that node.
See also
- Using Scripts
- General Usage Procedure
- SPS File Structure
- Presentation Procedures
Authentic View

In the Authentic View tab of the Main Window (the Authentic eForm tab), you can view and edit the Working XML File in its Authentic View. This view enables you (i) to see how your Authentic XML document will look, and (ii) to test the Authentic View created by the SPS. This is particularly useful if you wish to test the dynamic features of Authentic View. For example, you could test how Authentic View behaves when you:

- Add new elements and attributes
- Add new paragraphs or table rows
- Change values that affect conditional templates

Authentic View and the Working XML File

In order for Authentic View to be displayed, a Working XML File must be assigned to the active SPS document. This Working XML File must be valid according to the schema on which the SPS is based.

StyleVision creates a temporary XML file that is based on the Working XML File, and it is this temporary file that is displayed in the Authentic View tab of the Main Window. Modifications that you make in Authentic View will modify the temporary XML file. The Working XML File itself will not be modified till you explicitly save the modifications (with the menu command File | Save Authentic XML Data).

If no Working XML File is assigned, you will be prompted to assign a Working XML File when you click the Authentic View tab.

Authentic View limitations

The Authentic View in the Main Window is similar to the full-fledged Authentic View available in XMLSpy and Authentic Desktop except in the following major respects:
• Authentic View entry helpers are not available in the GUI. To insert or append nodes, you must right-click and use the context menus.
• CALS/HTML tables are not available for insertion.
• Text state icons are not available.

To test these features, you should use the full-fledged Authentic View in XMLSpy or Authentic Desktop. A full description of how to use Authentic View is given in the section Editing in Authentic View. For additional information, please see the Authentic View tutorial in the XMLSpy or Authentic Desktop user manual.

Enterprise editions of Authentic View applications
The following SPS functionality is enabled only in the Enterprise editions of Altova’s Authentic View applications:

• Absolute positioning (layout containers)
• Java and .Net function calls from XPath expressions in Auto-Calculations
• Variables
• User-Defined Elements and XML Text Blocks

If any of this functionality is present in an SPS that is opened in a non-Enterprise edition of an Authentic View application (say, XMLSpy Professional Edition), then the application displays a message saying that this functionality is available only in the Enterprise edition of the application.


See also
• SPS and Authentic View
• Editing in Authentic View
Output Views

There are four Output View tabs, one each for HTML, RTF, PDF, and Word 2007+ outputs. Each output view tab (illustration below) displays: (i) the XSLT stylesheet for that output; and (ii) a preview of the output produced by transforming the Working XML File with the XSLT stylesheet. In the PDF Preview tab there are two additional tab options: (i) XSL-FO, which displays the FO document produced by processing the XML document with the XSLT-FO (XSLT stylesheet); and (ii) FO Result, which displays the message output of the FO Processor when processing the FO document for PDF output.

In an Output View tab, the view can be switched between the XSLT stylesheet and the output preview by clicking the dropdown button in the Output View tab and selecting the XSLT option or the output preview option as required.

XSLT view

The XSLT view displays the XSLT stylesheet generated for that output format from the currently active SPS. The stylesheet is generated afresh each time the XSLT view is selected.

A stylesheet in an Output View tab is displayed with line-numbering and expandable/collapsible elements; click the + and – icons in the left margin to expand/collapse elements. The stylesheet in XSLT view cannot be edited, but can be searched (select Edit | Find) and text from it can be copied to the clipboard (with Edit | Copy).

Note: The XSLT stylesheets generated from the SPS can be separately generated and saved using the File | Save Generated Files command.
Output preview
The Output preview displays the output produced by transforming the Working XML File with the XSLT stylesheet for that output format. The output is generated afresh each time the Output preview tab is clicked. Note that it is the saved version of the Working XML File that is transformed—not the temporary version that is edited with Authentic View. This means that any modifications made in Authentic View will be reflected in the Output preview only after these modifications have been saved to the Working XML File (File | Save Authentic XML Data).

If no Working XML File is assigned when the Output preview is selected in the Output View tab, you will be prompted to assign a Working XML File. For DB-based SPSs, there is no need to assign a Working XML File since a temporary non-editable XML file is automatically generated when the DB is loaded and this XML file is used as the Working XML File.

Note: The output files generated from the SPS can be separately generated and saved using the File | Save Generated Files command.

See also
- Setting up StyleVision
- Save Generated Files
- Automated Processing
5.2 Sidebars

The Sidebars (also called sidebar windows or windows) are GUI components that help you design the SPS and provide you with information related to the active view. Each sidebar (listed below) is described in a sub-section of this section.

- Design Overview
- Schema Tree
- Design Tree
- Style Repository
- Styles
- Properties
- Project

Layout of the views

The layout of a view refers to what sidebars are available in that view and how these sidebars are positioned within the GUI. Layouts can be customized for separate view categories, and the customization consists of two parts: (i) switching on or off the display of individual sidebars in a view (via the View menu or by right-clicking the sidebar's title bar and selecting Hide); (ii) positioning the sidebar within the GUI as required. The layout defined in this way for a view category is retained for that particular view category till changed. So, for example, if in Design View, all the sidebars except the Styles sidebar are switched on, then this layout is retained for Design View over multiple view changes, till the Design View layout is changed. Note that the layout defined for any output preview (HTML, RTF, PDF, and Word 2007+ Previews) applies to all output previews. The view categories are: (i) no document open; (ii) Design View; (iii) Output Views; and (iv) Authentic View.

Docking and floating the Sidebar windows

Sidebar windows can be docked in the StyleVision GUI or can be made to float on your screen. To dock a window, drag the window by its title bar and drop it on any one of the four inner or four outer arrowheads that appear when you start to drag. The inner arrowheads dock the dragged window relative to the window in which the inner arrowheads appear. The four outer arrowheads dock the dragged window at each of the four edges of the interface window. To make a window float, (i) double-click the title bar; or (ii) drag the title bar and drop it anywhere on the screen except on the arrowheads that appear when you start to drag.

Alternatively, you can also use the following mechanisms. To float a docked window, click the Menu button at the top-right of a docked window (see screenshot below) and select Floating. This menu can also be accessed by right-clicking the title bar of the docked window.
To dock a floating window, right-click the title bar of the floating window and select Docking from the menu that appears; the window will be docked in the position in which it was last docked.

---

**Auto-Hiding Design sidebar windows**

A docked window can be auto-hidden. When a sidebar window is auto-hidden, it is minimized to a tab at the edge of the GUI. In the screenshot below, five sidebars have been auto-hidden: two at the left edge of the GUI, two at the bottom edge, and one at the right edge.

Placing the cursor over the tab causes that window to roll out into the GUI and over the Main Window. In the screenshot below, placing the cursor over the Styles tab causes the Styles sidebar to roll out into the Main Window.
Moving the cursor out of the rolled-out window and from over its tab causes the window to roll back into the tab at the edge of the GUI.

The Auto-Hide feature is useful if you wish to move seldom-used sidebars out of the GUI while at the same time allowing you easy access to them should you need them. This enables you to create more screen space for the Main Window while still allowing easy access to Design sidebar windows.

To auto-hide a window, in a docked window, click the Auto Hide button (the drawing pin icon) at the top right of the window (screenshot below). Alternatively, in the Menu, select Auto Hide; (to display the Menu, right-click the title bar of the window or click the Menu button in the title bar of the docked window).

The window will be auto-hidden.

To switch the Auto-Hide feature for a particular window off, place the cursor over the tab so that the window rolls out, and then click the Auto Hide button (screenshot below). Alternatively, in the Menu, deselect Auto Hide; (to display the Menu, right-click the title bar of the window or click the Menu button in the title bar of the window).
Note: When the Auto-Hide feature of a sidebar window is off, the drawing pin icon of that window points downwards; when the feature is on, the drawing pin icon points left.

---

Hiding (closing) sidebar windows
When a sidebar window is hidden it is no longer visible in the GUI, in either its maximized form (docked or floating) or in its minimized form (as a tab at an edge of the GUI, which is done using the Auto-Hide feature).

To hide a window, click the Close button at the top right of a docked or floating window. Alternatively, in the Menu, select Hide; (to display the Menu, right-click the title bar of the window or click the Menu button in the title bar of the window).

To make a hidden (or closed) window visible again, select the name of the Design sidebar in the View menu. The Design sidebar window is made visible in the position at which it was (docked or floating) when it was was hidden.

- See also
  - Design View
  - View Menu
Design Overview

The **Design Overview** sidebar *(screenshot below)* enables you to add schema sources, global parameters, SPS modules, and CSS files to the active SPS. It gives you an overview of these components and enables you to manage them conveniently in one location.

![Design Overview Sidebar](image)

---

**Adding schema sources**

Schema sources may be added to an empty SPS or to an SPS with an already existing schema source. A schema source is added by clicking the command **Add New Source** under the Sources heading. This pops up a menu *(screenshot below)* that enables you to add: (i) an XML Schema or DTD or an XML Schema that is automatically generated by StyleVision from an XML file; (ii) a schema generated by StyleVision from a DB; (iii) a user-defined schema; or (iv) an XBRL taxonomy.

![Add Schema/DB/Module/CSS/Parameters/Taxonomy](image)
The Working XML File and Template XML File
When a schema is added, it is listed under the Sources item. Each schema has two sub-items:

- The Working XML File.
- The Template XML File.

Enabling XML Signature in Authentic
The XML file of the main source can have an XML signature enabled. This enables the Authentic View user to sign and verify XML documents. Note that a signature can only be enabled for the main source and not for additional sources. For information about enabling and configuring XML signatures, see the section, XML Signatures.

Adding modules, CSS files, parameters, and XSLT files
Click the respective Add New commands at the bottom of the Modules, CSS Files, Parameters and XSLT Files sections to add a new item to the respective section.

Design Overview features
The following features are common to each section (Sources, Parameters, etc) in the Design Overview sidebar:

- Each section can be expanded or collapsed by clicking the triangular arrowhead to the left of the section name.
- Files in the Sources, Modules, and CSS Files sections are listed with only their file names. When you mouseover a file name, the full file path is displayed in a popup.
- Items that are listed in gray are present in an imported module, not in the SPS file currently active in the GUI.
- Each section also has a Add New <Item> command at the bottom of the section, which enables you to add a new item to that section. For example, clicking the Add New Parameter command adds a new parameter to the SPS and to the Parameters list in the Design Overview.
- Each item in a section has a context menu which can be accessed either by right-clicking that item or clicking its Context Menu icon (the downward-pointing arrow to the right of the item).
- The Remove icon in the context menu removes the selected item. This command is also available in context menus if the command is applicable.
- The context menu command Edit File in XMLSpy opens the selected file in the Altova application XMLSpy.
- The context menu commands, Move Up and Move Down, are applicable only when one of multiple modules in the Modules section is selected. Each button moves the selected module, respectively, up or down relative to the immediately adjacent module.
Sources
The Sources section lists the schema that the SPS is based on and the Working XML File and Template XML File assigned to the SPS. You can change each of these file selections by accessing its context menu (by right-clicking or clicking the Context Menu icon), and then selecting the appropriate Assign... option.

In the Enterprise Edition of XMLSpy, multiple schema sources can be used in an SPS. In the Enterprise Edition, therefore, the Add New Schema Source command enables you to add additional schema sources. When more than one schema source is present, one of these must be set as the main schema source, and it is indicated by the word (main) next to its header. By default the main schema source is the first schema source to have been added. If you wish to set another schema source as the main schema, select it, and in its context menu select the Set as Main Schema command.

Modules
The Modules section lists the SPS modules used by the active SPS. New modules are appended to the list by clicking the Add New Module command and browsing for the required SPS file. Since the order in which the modules are listed is significant, if more than one module is listed, the Move Up / Move Down command/s (in the context menu of a module) can be used to change the order. The context menu also provides a command for opening the selected module in StyleVision.

Note: The Design Overview sidebar provides an overview of the modules, enabling you to manage modules at the file level. The various module objects (objects inside the modules), however, are listed in the Design Tree sidebar.

CSS Files
The CSS Files section lists the CSS files used by the active SPS. New CSS files are appended to the list by clicking the Add New CSS File command and browsing for the required CSS file. Since the order in which the CSS files are listed is significant, if more than one CSS file is listed, the Move Up / Move Down command/s (in the context menu) become active when a CSS file is selected. The selected CSS file can be moved up or down by clicking the required command. The context menu also provides a command for opening the selected module in XMLSpy.

Note: The Design Overview sidebar provides an overview of the CSS files, enabling you to manage CSS files at the file level. The various CSS rules inside the CSS files, however, are listed in the Style Repository sidebar.
Parameters
The Parameters section lists the global parameters in the SPS. You can add new parameters using the Add New Parameter command at the bottom of the section. Double-clicking the parameter name or value enables you to edit the name or value, respectively. To remove a parameter, select the parameter and then click the Remove command in the context menu.

XSLT Files
The XSLT Files section lists the XSLT files that have been imported into the SPS. XSLT templates in these XSLT files will be available to the stylesheet as global templates. For a complete description of how this works, see XSLT Templates.

- See also
  - Parameters
  - Schema Tree
  - Modular SPSs
  - Design Fragments
  - Using Scripts
  - Designing Print Output
Schema Tree

The Schema Tree sidebar (screenshot below) enables you to do the following:

- Select multiple root elements (document elements) for a schema.
- Drag nodes (elements, attributes, global types) from a schema tree and drop them into the design. These nodes represent the XML content that is to be included in the output.
- View listings of all global elements and types in all schema sources. Enables a global element or global type to be created as a global template.
- View a listing of all namespaces used in the SPS.
- Insert and edit Design Fragments.
- Insert and edit user-defined XPath functions for the SPS.

![Schema Tree](image)

Root elements

For each schema, under the $XML heading, the selected Root elements (or document elements) are listed. This list consists of all the root elements you select for the schema (see below for how to do this). Each root element can be expanded to show its content model tree. It is from the nodes in these root element trees that the content of the main template is created. Note that the entry point of the main template is the document node of the main schema, which you can select or change at any time (see below for how to do this).

To select the root elements for a schema, do the following: Click the Select button at the right of the Root Elements item. This pops up the Select Root Elements dialog (screenshot below), in which you can select which of the global elements in the schema is/are to be the root elements. See SPS Structure | Schema Sources for an explanation of the possibilities offered by a selection of multiple root elements.
Global elements and global types
Global elements and global types can be used to create global templates which can be re-used in other templates. Additionally, global types can also be used directly in templates.

Design Fragments
All the Design Fragments in the document are listed under this item and can be viewed when the Design Fragments item is expanded. The following Design Fragment functionality is available:

- Creating a Design Fragment by clicking the Add icon of the Design Fragments item.
- Double-clicking the name of a Design Fragment in the Schema Tree enables the name of that Design Fragment to be edited.
- A Design Fragment can be enabled or disabled by, respectively, checking or unchecking the check box next to the Design Fragment.
- A Design Fragment can be dragged from the schema tree into the design.

See the section Design Fragments for information about working with Design Fragments.

User-defined XPath Functions
A user-defined XPath function can be added by clicking the Add icon of the XPath Functions item. After an XPath function has been created, it is listed in the Schema Tree. Double-clicking an XPath function opens the function in its dialog for editing. The following XPath functionality is available:

- An XPath function can be enabled or disabled by, respectively, checking or unchecking the check box next to the XPath Functions item.
- Right-clicking an XPath function also opens a context menu which contains commands to rename and remove an XPath function.

See the section, User-Defined XPath Functions, for detailed information about working with XPath functions.

Namespaces
The namespaces used in the SPS are listed under the Namespaces heading together with their prefixes. The namespaces in this list come from two sources: (i) namespaces defined in the referenced schema or schemas (see note below); and (ii) namespaces that are added to every newly created SPS by default. Referring to such a list can be very useful when writing XPath expressions. Additionally, you can set an XPath default namespace for the entire SPS by double-clicking the value field of the xpath-default-ns entry and then entering the namespace.

Note: If you wish to add a namespace to an SPS or to an XSLT stylesheet being generated from an SPS, the namespace must be added to the top-level schema element of the XML Schema on which the SPS is based.

Toolbar and schema tree icons
The following toolbar icons are shortcuts for common Schema Tree sidebar commands.

- In a user-defined schema, adds a child element to the document element or appends a sibling element to the selected element. More commands for building a user-defined schema are available by clicking the dropdown arrow of the icon.
- Make/Remove Global Template, enabled when a global element or global type is selected.
- Remove the selected item.
- Synchronize tree toggle. When toggled on (icon has border), selecting a node in the tree selects (i) the corresponding node in the design, and (ii) the corresponding node in the Design Tree if the Synchronize Tree icon in the Design Tree window is toggled on. When toggled off, the corresponding node in the design is not selected. Switch the toggle off if dragging a node from the tree and dropping it to the desired location in the design proves difficult.
- Auto-collapses other items in the schema tree when the Synchronize Tree toggle is turned on and an item is selected in the design. Note that this toggle is enabled only when the Synchronize Tree toggle is turned on.
Symbols used in schema trees
Given below is a list of the symbols in schema trees.

- **Name**: Element.
- **Mgr**: Attribute.
- **Person**: Element with child elements. Double-clicking the element or the +/- symbol to its left causes the element to expand/collapse.
- **Addresses**: DB Filter applied. Applies only to top-level data table elements in the schema tree.
- **n1:PersonType n1:emailType**: Global types can be either complex or simple. Complex types are indicated with a cyan icon, simple types with a brown icon.

**See also**
- Creating the SPS Structure | Schema Sources
Design Tree

The Design Tree sidebar (screenshot below) provides an overview of the SPS design.

At the root of the Design Tree is the name of the SPS file; the location of the file is displayed in a pop-up when you mouseover. The next level of the Design Tree is organized into the following categories:

- **Scripts**, which shows all the JavaScript functions that have been defined for the SPS using the JavaScript Editor of StyleVision.
- **Main Template**, which displays a detailed structure of the main template.
- **Global Templates**, which lists the global templates in the current SPS, as well as the global templates in all included SPS modules.
- **Design Fragments**, which shows all the Design Fragments in the design, and enables you to create, edit, rename, and delete them.
- **XSLT Templates**, which provides the capability to view XSLT templates in imported XSLT files.
- **User-Defined XPath Functions**, which enables you to create, edit, rename, and remove your own XPath functions.

Toolbar icons

The following toolbar icons are shortcuts for common Schema Tree sidebar commands.

- Adds a Design Fragment, main template, or layout item to the design. Clicking the left-hand part of the icon adds a Design Fragment. Clicking the dropdown arrow drops down a list with commands to add a Design Fragment or any of various layout items.
- Remove the selected item; icon is active when item in the Global Templates or Layout sub-trees is selected.
- Synchronize tree toggle. When toggled on (icon has border), selecting a node in the tree selects (i) the corresponding node in the design, and (ii) the corresponding node in the schema tree if the Synchronize Tree icon in the schema tree is toggled on. When toggled off, the corresponding nodes in the design and schema tree are not selected.
- Auto-collapses other items in the design tree when the Synchronize Tree toggle is turned on and an item is selected in the design. Note that this toggle is enabled only
when the Synchronize Tree toggle is turned on.

### Modifying the Design Tree display

The display of the Design Tree can be modified via the context menu (*screenshot below*), which pops up on right-clicking an item in the Design Tree.

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove</td>
</tr>
<tr>
<td>Make Design Fragment</td>
</tr>
<tr>
<td>Expand All</td>
</tr>
</tbody>
</table>

The **Remove** command removes the selected Design Tree item from the Design Tree. **Make Design Fragment** creates a Design Fragment in the design and adds an item for it in the Design Tree. **Expand All** expands all the items of the Design Tree.

### Scripts and Main Template

The Scripts listing displays all the scripts in the Design, including those in imported modules. The Main Template listing displays a tree of the main template. Items in the tree and the design can be removed by right-clicking the item and selecting **Remove**.

### Global Templates

The **Global Templates** item lists all global templates in the current SPS and in all added SPS modules. Global templates defined in the current SPS are displayed in black, while global templates that have been defined in added modules are displayed in gray (*see screenshot below*). Each global template has a check box to its left, which enables you to activate or deactivate it. When a global template is deactivated, it is removed from the design.

A global template in the current SPS (not one in an added module) can be removed by selecting it.
and clicking the **Remove** button in the toolbar or the **Remove** command in the context menu. The component is removed from the design and the tree.

---

**Design Fragments**

The **Design Fragments** item lists all the Design Fragments in the current SPS and in all added SPS modules. Design Fragments defined in the current SPS are displayed in black, while Design Fragments that have been defined in added modules are displayed in gray (*see screenshot below*). Each Design Fragment has a check box to its left, which enables you to activate or deactivate it. A Design Fragment in the current SPS (not one in an added module) can be removed by selecting it and clicking the **Remove** command in the context menu. The component is removed from the design and the tree.

A Design Fragment can be added by clicking the Add icon to the right of the Design Fragments item. Each Design Fragment is designed as a tree with expandable/collapsible nodes. Any component in a Design Fragment tree (that is defined in the current SPS) can be removed by selecting it and clicking the **Remove** button in the toolbar or the **Remove** command in the context menu. The component is removed from the design and the tree.

---

**XSLT Templates**

In the Design Tree sidebar (*screenshot below*), the XSLT Templates contained in the imported XSLT file are displayed under the XSLT Templates heading.
There are two types of imported XSLT templates: (i) match templates (indicated by *Match*), and (ii) named templates (indicated by *Name*). In the Design Tree, these two types are listed with (i) the value of the `select` attribute of match templates, and (ii) by the value of the `name` attribute of named templates, respectively. For a complete description of how XSLT Templates work, see [XSLT Templates](#).

---

**See also**
- Design Fragments
- Using Scripts
- Designing Print Output
Style Repository

In the Style Repository sidebar (screenshot below), you can assign external CSS stylesheets and define global CSS styles for the SPS. Style rules in external CSS stylesheets and globally defined CSS styles are applied to Authentic View and the HTML output document.

The Style Repository sidebar contains two listings, External and Global, each in the form of a tree. The External listing contains a list of external CSS stylesheets associated with the SPS. The Global listing contains a list of all the global styles associated with the SPS.

The structure of the listings in the Style Repository is as follows:

- **External**
  - CSS-1.css (Location given in popup that appears on mouseover)
  - Media (can be defined in Style Repository window)
  - Rules (non-editable; must be edited in CSS file)
    - Selector-1
      - Property-1
      - ...
      - Property-N
    - ...
    - Selector-N
  + ...
  + CSS-N.css

- **Global**
  - Selector-1
    + Selector-1 Properties
  - ...
  + Selector-N
**Precedence of style rules**

If a global style rule and a style rule in an external CSS stylesheet have selectors that identify the same document component, then the global style rule has precedence over that in the external stylesheet, and will be applied. If two or more global style rules select the same document component, then the rule that is listed last from among these rules will be applied. Likewise, if two or more style rules in the external stylesheets select the same document component, then the last of these rules in the last of the containing stylesheets will be applied.

---

**Managing styles in the Style Repository**

In the Style Repository sidebar you can do the following, using either the icons in the toolbar and/or items in the context menu:

**Add:** The Add icon adds a new external stylesheet entry to the External tree or a new global style entry to the Global tree, respectively, according to whether the External or Global tree was selected. The new entry is appended to the list of already existing entries in the tree. The Add command is also available in the context menu. For more details about using external stylesheets and global styles, see Working with CSS Styles. Note that an external CSS stylesheet can also be added or a stylesheet removed via the Design Overview sidebar.

**Insert:** The Insert icon inserts a new external stylesheet entry above the selected external stylesheet (in the External tree) or a new global style entry above the selected global style (in the Global tree). The Insert command is also available in the context menu. For more details about using external stylesheets and global styles, see Working with CSS Styles.

**Move Up/Down:** The Move Up icon and Move Down icon move the selected external stylesheet or global style respectively up and down relative to the other entries in its tree. These commands are useful for changing the priority of external stylesheets relative to each other and of global style rules relative to each other. The Move Up and Move Down commands are also available in the context menu. For more details about how to change the precedence of styles, see Working with CSS Styles.

**Views of a selector's styles:** Any selector, whether in an external stylesheet or defined globally, can be displayed in a view obtained by using three view settings. These settings are: List Non-Empty, Expand All, and Collapse All and they are available as toolbar buttons and context menu commands: Toggling the List Non-Empty setting on causes only those style properties to be listed that have a value defined for them. Otherwise all available style properties are displayed (which could make the view very cluttered). The Expand All and Collapse All settings combine with the List Non-Empty setting, and respectively expand and collapse all the style definitions of the selected selector. These commands are also available in the context menu.

**Toggle Important:** Clicking the Toggle Important icon sets the CSS value !important on or off for the selected CSS rule.

**Reload All:** The Reload All icon reloads all the external CSS stylesheets.
Reset: The Reset icon \( \times \) deletes the selected external stylesheet or global style.

---

**Editing CSS styles in the Style Repository**
The following editing mechanisms are provided in the Style Repository:

- You can add and remove a CSS Stylesheet, and you can specify the media to which each external CSS stylesheet applies. How to do this is explained in the section External CSS Stylesheets.
- Global styles can have their selectors and properties directly edited in the Style Repository window. How this is done is described in the section Defining CSS Styles Globally.

- **See also**
  - Design Overview
  - Working with CSS Styles
  - Styles sidebar
Styles

The Styles sidebar (screenshot below) enables CSS styles to be defined locally for SPS components selected in the Design View. This is as opposed to styles which are set globally in the Styles Repository sidebar.

The Styles sidebar is divided into two broad parts:

- The left-hand-side, Styles-For column, in which the selected component types are listed. You should note that when a selection is made in Design View, it could contain several components. The selected components are listed in the Styles-For column, organized by component type. One of these component types may be selected at a time for styling. If there is only one instance of the component type, then that one instance is selected for styling. If there are several instances of the component type, then all the selected instances can be styled together. The defined styles are applied locally to each instance. If you wish to style only one specific instance, then select that specific component instance in Design View and style it locally in the Styles sidebar. You can also select a component range by selecting the start-of-range and then the end-of-range component with the Shift-key pressed. For detailed information about the selection of component types, see Defining CSS Styles Locally.

- The right-hand-side, Style Definitions pane, in which CSS styles are defined for the component type/s selected in the Styles-For column. The Style Definitions pane can be displayed in three views (see below for description). For the details of how to set style definitions, see Setting CSS Style Values. The XPath icon \[\text{XPath}\] toggles on and off the application of XPath expressions as the source of style values. If a style property is selected and if the XPath icon is toggled on, then an XPath expression can be entered for this property and the return value of the XPath expression is used as the value of that style property. In this way, the value of a node in an XML document can be returned at runtime as the value of a property. When the XPath icon is toggled off, a static value can be entered as the value of the property.

Settings for Definitions-View

The view of definitions can be changed to suit your editing needs. Three view-settings (listed below) are available as buttons in the toolbar and as commands in context menus.
• **List Non-Empty** ![icon]: When this setting is toggled on, for the component type selected in the left-hand column, only those properties with values defined for them are displayed, in alphabetical order. Otherwise all properties are displayed. This setting is very useful if you wish to see what properties are defined for a particular component type. If you wish to define new properties for the selected component type, this setting must be toggled off so that you can access the required property.

• **Expand All** ![icon]: For the component type selected in the left-hand column, all the properties displayed in the right-hand pane are expanded. This setting can be combined with the **List Non-Empty** setting.

• **Collapse All** ![icon]: For the component type selected in the left-hand column of the window, all the properties displayed in the right-hand pane are collapsed. This setting can be combined with the **List Non-Empty** setting.

---

**Toggle Important and Reset toolbar icons**

Clicking the Toggle Important icon ![icon] sets the CSS value `!important` on or off for the selected CSS rule. Clicking the Reset icon ![icon] resets the value of the selected property.

---

**See also**

- Working with CSS Styles
- Style Repository
- Predefined Formats
Properties

The Properties sidebar (screenshot below) enables properties to be defined for SPS components selected in Design View.

The Properties sidebar is divided into two broad parts:

- The Properties-For column, in which the selected component-types are listed. One of these component types may be selected at a time and properties assigned for it. (In the screenshot above, the template component is selected.) For detailed information about how components with properties are grouped, see the section Components and their Property Groups below.
- The Property Definitions pane, in which component properties are defined for the component type selected in the Properties For column. The Property Definitions pane can be displayed in three views (see below). For the details of what properties are in each property group, see the section Property Groups below.

**Settings for Definitions-View**

The view of definitions can be changed to suit your editing needs. Three view-settings (listed below) are available as buttons in the toolbar and as commands in context menus.

- **List Non-Empty**: When this setting is toggled on, for the component type selected in the left-hand column, only those properties with values defined for them are displayed, in alphabetical order. Otherwise all properties are displayed. This setting is very useful if you wish to see what properties are defined for a particular component type. If you wish to define new properties for the selected component type, this setting must be toggled off so that you can access the required property.

- **Expand All**: For the component type selected in the left-hand column, all the properties displayed in the right-hand pane are expanded. This setting can be combined with the List Non-Empty setting.
• **Collapse All** For the component type selected in the left-hand column of the window, all the properties displayed in the right-hand pane are collapsed. This setting can be combined with the **List Non-Empty** setting.

---

**Reset toolbar icon**
Clicking the Reset icon resets the value of the selected property to its default.

---

**Components and their property groups**
The availability of property groups is context-sensitive. What property groups are available depends on what design component is selected. The table below lists SPS components and the property groups they have.

<table>
<thead>
<tr>
<th>Component</th>
<th>Property Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template</td>
<td>Template; Authentic</td>
</tr>
<tr>
<td>Content</td>
<td>Content; Authentic; Common; Event</td>
</tr>
<tr>
<td>Text</td>
<td>Text; Common; Event</td>
</tr>
<tr>
<td>Auto-Calculation</td>
<td>AutoCalc; Authentic; Common; Event</td>
</tr>
<tr>
<td>Condition Branch</td>
<td>When</td>
</tr>
<tr>
<td>Data-Entry Device</td>
<td>Authentic; Common; [Data-Entry Device]; Event; HTML</td>
</tr>
<tr>
<td>Image</td>
<td>Image; Authentic; Common; Event; HTML</td>
</tr>
<tr>
<td>Link</td>
<td>Link; Authentic; Common; Event; HTML</td>
</tr>
<tr>
<td>Table</td>
<td>Table; Authentic; Common; Event; HTML; Interactive</td>
</tr>
<tr>
<td>Paragraph</td>
<td>Paragraph; Authentic; Common; Event; HTML</td>
</tr>
</tbody>
</table>

The following points about component types should be noted:

- **Template components** are the main template, global templates, and all schema nodes in the design.
- **Content components** are the content and rest-of-contents placeholders. These represent the text content of a node or nodes from the XML document.
- A text component is a single string of static text. A single string extends between any two components other than text components, and includes whitespace, if any is present.
- **Data-entry devices** are input field, multiline input fields, combo boxes, check boxes, radio buttons and buttons; their properties cover the data-entry device as well as the contents of the data-entry device, if any.
- A table component refers to the table structure in the design. Note that it contains sub-components, which are considered components in their own right. The sub-components
are: row, column, cell, header, and footer.
- A paragraph component is any predefined format.
The table below contains descriptions of each property group.

<table>
<thead>
<tr>
<th>Property Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoCalc</strong></td>
<td>These properties are enabled when an Auto-Calculation is selected. The Value Formatting property specifies the formatting of an Auto-Calculation that is a numeric or date datatype. The XPath property specifies the XPath expression that is used for the Auto-Calculation.</td>
</tr>
<tr>
<td><strong>Authentic</strong></td>
<td>These are SPS-specific properties that are available for templates, contents, AutoCalculations, data-entry devices, images, links, tables, and paragraphs. What properties within the group are available are component-specific. For more details, see Authentic Node Properties.</td>
</tr>
<tr>
<td><strong>Common</strong></td>
<td>The Common property group is available for all component types except the Template and AutoCalc component types. It contains the following properties that can be defined for the component: class (a class name), dir (the writing direction), id (a unique ID), lang (the language), and title (a name).</td>
</tr>
<tr>
<td><strong>Data-Entry Device</strong></td>
<td>Specifies the value range of combo boxes, check boxes, and radio buttons. Note that this property group does not apply to input fields and buttons.</td>
</tr>
<tr>
<td><strong>Event</strong></td>
<td>Contains properties that enable JavaScript functions to be defined for the following client-side HTML events: onclick, ondblclick, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup.</td>
</tr>
<tr>
<td><strong>HTML</strong></td>
<td>Available for the following component types: data-entry devices; image; link; table; paragraphs. Note that there are different types of data-entry devices and paragraphs, and that tables have sub-components. These properties are HTML properties that can be set on the corresponding HTML elements (img, table, p, div, etc). The available properties therefore vary according to the component selected. Values for these properties can be selected using XPath expressions.</td>
</tr>
</tbody>
</table>

In addition, there are component-specific properties for images, links, paragraphs and other predefined formats, and condition branches. These properties are described in the respective sections.

---

**Setting property values**

Property values can be entered in one, two, or three ways, depending on the property:

- Entered directly in the Value column. To do this, select a property, double-click in its Value column, enter the value using the keyboard, and press Enter or click anywhere in the GUI.
- By selecting a value from the dropdown list of the combo box for that property. Click the down arrow of the combo box to drop down the list of property-value options.
- By using the Edit button at the right-hand side of the Value column for that property. Clicking the Edit button pops up a dialog relevant to that property.
For some properties, in the Common and HTML groups of properties, XPath expressions can be used to provide the values of the property. The XPath icon \(\text{XPath}\) toggles on and off the application of XPath expressions as the source of property values. With a property selected, if the XPath icon is toggled on, then an XPath expression can be entered for this property and the return value of the XPath expression is used as the value of that property. In this way, the value of a node in an XML document can be returned, at runtime, as the value of a property. When the XPath icon is toggled off, a static value can be entered as the value of the property. Also see Style Properties Via XPath.

---

**Modifying or deleting a property value**

To modify a property value, use any of the applicable methods described in the previous paragraph, Setting Property Values. To delete a property value, select the property and click the Reset icon in the toolbar of the Properties sidebar.

- See also
  - Styles
Project

StyleVision projects are an efficient way of grouping, managing, and working with related files. Once collected in a project, files can be accessed easily from the Project sidebar when designing an SPS. For example, an SPS file can be dragged from the Project sidebar to the Design Tree sidebar and created there as a module; or an image file can be dropped into the design as a static image; or a CSS stylesheet can be dragged to the Style Repository sidebar as an external stylesheet.

A complete description of how to work with projects is given in the section Projects in StyleVision.

The Project sidebar (screenshot below) shows the currently active project. Commands in the Project menu apply to the currently active project. The currently active project can be changed by either creating a new project (Project | New Project) or by opening an existing project (Project | Open Project). These two commands are also available as the first and second buttons in the toolbar of the Project sidebar (see screenshot below). You can name a newly created project when you save it (Project | Save Project; third button in the toolbar of the Project sidebar).

The Add Files button in the toolbar (fourth button in the screenshot below) contains the commands shown in the screenshot below. These commands enable files and folders to be added to the active project and are described in the User Reference. A useful command is the Add Active and Related Files command, which adds to the project all the files related to the currently active SPS. For example, with the tutorial file, QuickStart.sps, active, clicking this command will add all related files as shown in the screenshot above.
The **Project Properties** button (fifth button in the toolbar) pops up the Properties dialog of (i) the project; (ii) a folder; or (iii) a file, depending on which type of these three items is currently selected in the Project sidebar. If the project or a file is selected, the Properties dialog displays the location of the file. The Properties dialog of a folder allows you to edit the name of the folder and to specify the file extensions to associate with that folder; only files with the associated file extensions are displayed in that project folder. See [Projects in StyleVision](Projects in StyleVision) for details.

- **See also**
  - [Projects in StyleVision](Projects in StyleVision)
  - [Project menu](Project menu)
6 Quick Start Tutorial

The objective of this tutorial is to take you quickly through the key steps in creating an effective SPS. It starts with a section on creating and setting up the SPS, shows you how to insert content in the SPS, how to format the components of the SPS, and how to use two powerful SPS features: Auto-Calculations and conditions. Along the way you will get to know how to structure your output efficiently and how to use a variety of structural and presentation features.

Files required
Files related to this Quick Start tutorial are in the (My) Documents folder, C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial\QuickStart:

- QuickStart.xsd, the XML Schema file on which the SPS is based.
- QuickStart.xml, the Working XML File, which is the source of the data displayed in the output previews.
- QuickStart.sps, which is the finished SPS file; you can compare the SPS file you create with this file.
- QuickStart.css, which is the external CSS stylesheet used in the tutorial.
- NewsItems.BMP, an image file that is used in the SPS.

Doing the tutorial
It is best to start at the beginning of the tutorial and work your way through the sections. Also, you should open the XSD and XML files before starting the tutorial and take a look at their structure and contents. Keep the XSD and XML files open while doing the tutorial, so that you can refer to them. Save your SPS document with a name other than QuickStart.sps (say MyQuickStart.sps) so that you do not overwrite the supplied SPS file. And, of course, remember to save after successfully completing every part.

Links
- Next: Creating and Setting Up a New SPS
6.1 Creating and Setting Up a New SPS

In this section, you will learn:

- How to create a new SPS document.
- How to add a schema source for the SPS.
- How to select the XSLT version of the SPS.
- How to assign the Working XML File.
- How to specify the output encoding.
- How to save the SPS document.

Creating a new SPS document

Create a new SPS document by clicking `File | New | New (Empty)` or select `New (Empty)` in the dropdown list of the `New` icon in the application toolbar. The Create New Design dialog pops up.

The Create New Design dialog (screenshot below) prompts you to select either: (i) a free-flowing document design, or (ii) a form-based document design (in which components are positioned absolutely, as in a layout program).

![Create New Design](image)

In a free-flowing document design, document content is laid out to fit the output media object or viewer (paper or screen). Items in the document content can only be placed relative to each other, and not absolutely. This kind of design is suited for documents such as reports, articles, and books.
In a form-based document, a single Layout Container is created, in which design components can be positioned absolutely. The dimensions of the Layout Container are user-defined, and Layout Boxes can be positioned absolutely within the Layout Container and document content can be placed within individual Layout Boxes. If you wish the design of your SPS to replicate a specific form-based design, you can use an image of the original form as a blueprint image. The blueprint image can then be included as the background image of the Layout Container. The blueprint image is used to help you design your form; it will not be included in the output.

You will be creating a free-flowing document, so select this option by clicking the Create a free-flow document radio button, then click OK.

A new document titled SPS1.sps is created and displayed in Design View (screenshot below).

In Design View, an empty main template is displayed. In the Design Overview and Schema Tree sidebars, there are no schema entries.

Adding a schema source
For this SPS, you will use the schema, QuickStart.xsd. To add this schema as the schema source, do the following:

1. In the Design Overview sidebar, under the Sources heading, click the Add New Source command (screenshot above). In the menu that pops up (screenshot below), select Add XML Schema/DTD/XML.

2. In the Open dialog that pops up browse for the file in the (My) Documents folder, C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2015 \StyleVisionExamples\Tutorial\QuickStart\QuickStart.xsd, and click Open.

3. You will be prompted to select a Working XML File. Select the option to select the file from the filesystem, then browse for the file (in the (My) Documents folder) C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2015 \StyleVisionExamples\Tutorial\QuickStart\QuickStart.xml, and click Open. The
Creating and Setting Up a New SPS

Quick Start Tutorial

Creating and Setting Up a New SPS

A schema will be added as a schema source in the Design Overview sidebar and in the Schema Tree sidebar (screenshot below). Also, in the Design Overview, the Working XML File you chose will be assigned to the schema.

You should note the following points: (i) In Design Overview, the $XML entry for the schema source lists the schema and the Working XML File and Template XML File; (ii) In the Schema Tree sidebar, the Root Elements tree would list the one or more root elements (document elements) you select from among the global elements defined in the schema. In the case of this schema, the element presswatch is selected by default because it is the one global element in the schema that lies clearly at the top of the hierarchy defined in the schema; (iii) All global elements in the schema are listed in the All Global Elements tree.

Selecting the XSLT version

For this SPS you will use XSLT 2.0. To specify the XSLT version, in the application toolbar, click the icon.

Assigning or changing the Working XML File

While adding the XML Schema to the SPS in the previous step, you also assigned a Working XML File to the schema. A Working XML File provides the SPS with a source of XML data to process. To assign, change, or unassign a Working XML File for a given schema, in the Design Overview sidebar, right-click anywhere in the Working XML File line you wish to modify (or click the Context Menu icon at the right), and select the required command from the context menu that pops up. The Working XML File is now assigned, and the filename is entered in the Design Overview. Before proceeding, ensure that you have correctly assigned the file C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples
\Tutorial\QuickStart\QuickStart.xml, which is in the (My) Documents folder, as the Working XML File.

---

**Specifying the encoding of output**

In the Default Encoding tab of the Options dialog (Tools | Options), set the HTML encoding to Unicode UTF-8, and RTF and PDF encoding to UTF-8.

---

**Saving the SPS document**

After you have set up the SPS as described above, save it as MyQuickStart.sps in the (My) Documents folder C:\Documents and Settings\<username>\My Documents\Altova \StyleVision2015\StyleVisionExamples\Tutorial\QuickStart folder. Do this by clicking the menu command File | Save Design or Ctrl+S, and then entering the file name in the Save Design dialog that pops up.

---

- **Links**
  - Next: Inserting Dynamic Content
  - Tutorial Start Page
6.2 Inserting Dynamic Content (from XML Source)

This section introduces mechanisms to insert data from nodes in the XML document. In it you will learn how to drag element and attribute nodes from the schema tree into the design and create these nodes as contents. When a node is created as contents, the data in it is output as a string which is the concatenation of the content of that element's child text nodes and the text nodes of all descendant elements.

---

Inserting element contents

In your SPS, do the following:

1. In the Schema Tree sidebar, expand the schema tree up to the children of the `newsitem` element (screenshot below).

![Schema Tree](image)

2. Select the `headline` element (notice that the element's datatype is displayed in a pop-up when you mouseover; screenshot above). Drag the `headline` element into Design View, and, when the arrow cursor turns to an insertion point, drop it into the main template.

3. In the context menu that pops up, select Create Contents. The start and end tags of the `headline` element are inserted at the point where you dropped the `headline` element, and they contain the content placeholder. The `headline` tags are surrounded by the start and end tags of the ancestor elements of `headline` (screenshot below).

4. In the design put elements on different lines (by pressing Enter) as shown in the screenshot below.
Click the HTML tab to see a preview of the HTML output (screenshot below). The HTML preview shows the contents of the headline child elements of newsitem, each as a text string.

You should also check the preview of Authentic View and the RTF, PDF, and Word 2007 output.

**Note:** You can also create the contents of a node by using the following steps: (i) Click the the Insert Contents icon in the Insert Design Elements toolbar, (ii) Click the location in the design where you wish to insert the content, (iii) Select, from the Schema Selector tree that pops up, the node for which you wish to create contents.

---

**Inserting attribute contents**

When an element is inserted into the design as contents, the contents of its attributes are not automatically inserted. You must explicitly drag the attribute node into the design for the attribute’s value to be output. In your SPS, now do the following:

1. Place the cursor after the end tag of the headline element and press Enter. This produces an empty line (screenshot below).
2. In the Schema Tree sidebar, expand the `dateline` element (screenshot below).

![Schema Tree]

Notice that the `dateline` element has two child elements, `date` and `place`, and that the `place` element has two attributes, `city` and `country`.

3. Drag the `dateline` element into the design and drop it at the beginning of the newly created empty line (screenshot below).
4. Switch to HTML Preview and look carefully at the output of dateline (screenshot below).

NanoNull Inc Launches Version 2.0 of NanoPower
2006-04-01

NanoNull Inc Jumps 3% on Release of New NanoPower Version
2006-04-01

NanoNull Shares Up 10% on Month Following New NanoPower Version
2006-04-25

NanoDiamonds Project to Go Ahead
2006-05-06

Notice that while the contents of the date children of dateline elements have been output, no contents have been output for the place children of dateline. This is because the place data is contained in the attributes of the place element (in the attributes city and country) and attribute contents are not output when the attribute's parent element is processed.

5. In Design View, go to the menu command Authentic | Auto-Add Date Picker, and toggle it off to deactivate the auto-addition of the date picker. (The icon will have no border when toggled off.) This step is required if the date picker is not to be inserted automatically when a node of type xs:date or xs:dateTime is inserted into the design (which you will do in the next step). Drag the date element from the Schema Tree sidebar and drop it (create it as contents) in between the start and end tags of the dateline element.

6. Select the city attribute of the dateline/place element (screenshot below) in the Schema Tree sidebar.
7. Drag the `@city` attribute node into **Design View**, and drop it (create as contents) just after the end tag of the `date` element.

8. Drag the `@country` attribute node into **Design View**, and drop it (create as contents) just after the end tag of the `@city` attribute.

When you are done, the SPS design should look something like this:

![Schema Tree Diagram]

The **HTML Preview** will look like this:
**Quick Start Tutorial I nserting Dynamic Content (from XML Source)**

NanoNull Inc Launches Version 2.0 of NanoPower
2006-04-01BostonUSA

NanoNull Inc Jumps 3% on Release of New NanoPower Version
2006-04-01New YorkUSA

NanoNull Shares Up 10% on Month Following New NanoPower Version
2006-04-25New YorkUSA

NanoDiamonds Project to Go Ahead
2006-05-06LondonUK

Notice that the values of the @city and @country attributes are now included in the output.

---

**Adding more dynamic content**

The contents of elements and attributes from the XML data source can be inserted anywhere in the design using the method described above. To complete this section, add the synopsis and source elements to the design so that the design now looks like this:

Notice that the synopsis element has been placed before the source element, which is not the order in which the elements are in the schema. After you have added the synopsis and source elements to the design, check the HTML preview to see the output. This is an important point to
note: That the order in which nodes are placed in the main template is the order in which they will appear in the output (see the section, Templates and Design Fragments, for more information about structuring the output document).

Another important point to note at this stage is the form in which a node is created in the design. In the HTML preview, you will see that all the nodes included in the design have been sent to the output as text strings. Alternatively to being output as a text string, a node can be output in some other form, for example, as a table or a combo box. In this section, you have, by creating all the nodes as (contents), specified that the output form of all nodes are text strings. In the section, Using Conditions, you will learn how to create a node as a combo box, and in the section, Using Global Templates and Rest-of-Contents, how to create a node as a (dynamic) table.

Make sure to save the file before moving to the next section.

---

**Links**

- Next: Inserting Static Content
- Previous: Creating and Setting Up a New SPS
- Tutorial Start Page
- SPS File Structure
6.3 Inserting Static Content

Static content is content you enter or insert directly in the design—as opposed to content that comes from the XML source. A variety of static components can be placed in an SPS design. In this part of the tutorial, you will learn how to insert the following static components:

- An image
- A horizontal line
- Text

Inserting a static image

The static image to insert is in the (My) Documents folder: C:\Documents and Settings \<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial \QuickStart\NewsItems.BMP. It will be used as the header of the document. To insert this image at the head of the document, do the following:

1. Place the cursor between the start-tags of newsitems and newsitem (screenshot below).

   ![Initial Document Section Screenshot]

   Notice that the cursor is within the newsitems element but outside the newsitem element. It will therefore be inserted in the output once, at the start of processing of the newsitems element (because there is only one newsitems element defined in the schema).

2. Right-click, and select Insert | Image. The Insert Image dialog pops up (screenshot below).

   ![Insert Image Screenshot]

3. In the Static tab, click the Absolute Path check box, then browse for the file NewsItems.BMP and select it.

4. Click OK to finish.
The HTML preview will look something like this:

---

**Summary of News Items**

NanoNull Inc Launches Version 2.0 of NanoPower
2006-04-01BostonUSA
NanoNull Inc today launched version 2.0 of its market-leading NanoPower line of hardware and software. The highlights of the new version of NanoPower are improved chip design capabilities in NanoSoft, the software used to design computer chips; and higher precision sand-filtering processes in NanoMeld, the hardware in which the transformation from sand to silicon chips is executed.

NewTech Online
---

---

**Inserting horizontal lines**

The first horizontal line you will insert is between the document header and document body. Do this as follows:

1. Place the cursor immediately after the recently inserted static image.
2. Right-click, and select **Insert | Horizontal Line**. A horizontal line is inserted.

Set properties for the line as follows:

1. With the line selected in **Design View**, in the **Properties sidebar**, select the **line** component (in the Properties For column) and then the **HTML** group of properties.
2. Assign **color** and **size** properties for the line.
3. With the line selected in **Design View**, in the **Styles sidebar**, select the **line** component and then the **box** group of properties. Define a **margin-bottom** property of **12pt**.
4. Check the output in **HTML Preview**.

Now insert a horizontal line at the end of each news item. To do this the cursor would have to be placed immediately before the end-tag of the **newsitem** element. This will cause the line to be output at the end of each **newsitem** element. You can change the thickness of the line by setting the line’s **size** property to a number with no unit (in the Properties sidebar, select **line**, and set a value of, say **3**).

---

**Inserting static text**

You have already added static text to your design. When you pressed the **Enter** key to obtain new lines (in the section **Inserting Dynamic Content (from XML Source)**), whitespace (static text) was added. In this section, you will add a few static text characters to your design.

The SPS you have designed up to this point will produce output which looks something like this:
Notice that in the output of the dateline element, the contents of the date element and place/@city and place/@country attributes are run together without spacing. You can add the spacing as static text. In the design, place the cursor after the date element and enter a colon and a space. Next, enter a comma and space after the @city attribute (screenshot below).

This part of the output will now look like this:

Notice the colon, spacing and comma in the dateline output. All of these text items are static text items that were inserted directly in the design.

You will now add one more item of static text. In the design, type in the string "Source: " just before the start-tag of the source element (screenshot below).
Formatting static text

To format static text, highlight the text to be formatted and specify local style properties. In the design, highlight the text "Source:" that you just typed. In the Styles sidebar (screenshot below), notice that the text component is selected. Now expand the font group of properties as shown in the screenshot below, and, for the font-style property, select the italic option from the dropdown menu.

The static text (that is, the string "Source:" will be given an italic style in the design, and will look like this:

```
<synopsis>(content)</synopsis>
Source: <source>(content)</source>
```

The output will look like this in HTML Preview:

Summary of News Items

NanoNull Inc Launches Version 2.0 of NanoPower
2006-04-01: Boston, USA
NanoNull Inc today launched version 2.0 of its market-leading NanoPower line of hardware and software. The highlights of the new version of NanoPower are improved chip design capabilities in NanoSoft, the software used to design computer chips; and higher precision sand-filtering processes in NanoMeld, the hardware in which the transformation from sand to silicon chips is executed.

Source: NewTech Online

If you think there is too little vertical space between the source item and the horizontal line
separating two `newsitem` elements, then, in the design, insert a blank line between the source and the horizontal line (by pressing `Enter`).

After you are done, save the file.

In this section you have learned how to insert static content and format it. In the next section you will learn more about how design components can be formatted using CSS principles and properties.

Links

- [Next: Formatting the Content](#)
- [Previous: Inserting Dynamic Content](#)
- [Tutorial Start Page](#)
### 6.4 Formatting the Content

StyleVision offers a powerful and flexible **styling mechanism**, based on CSS, for formatting components in the design. The following are the key aspects of StyleVision's styling mechanism:

- **CSS style rules** can be defined for both block components and inline components.
- **Predefined formats** are block components that have inherent styles and can be used as wrappers for a group of components that need to be treated as a block. The inherent styles of these predefined formats can be overridden by styles you specify locally on each component. This is in keeping with the cascading principle of CSS.
- **Class attributes** can be declared on components in the design, and the class can be used as a selector of **external** or **global** style rules.
- You can specify styles at three levels. These are, in increasing order of priority: (i) style rules in [external stylesheets](#), (ii) **global style rules**, and (iii) **local style rules**. Note, however, that certain types of selectors in external and global style rules, such as name-based selectors (h1, a, img, etc), will apply only to Authentic View and HTML output, not to RTF, PDF, and Word 2007+ output. Rules that have class selectors will apply to HTML, RTF, PDF, and Word 2007+ formats.

In this section, you will learn how to:

- Assign predefined formats
- Assign a component a class attribute
- Define styles in an external CSS stylesheet and add this stylesheet to the style repository of the SPS
- Define global style rules
- Define local styles for a selection of multiple design components
- Define local styles for a single component

### Assigning predefined formats

One reason to assign a **predefined format** is to give a component the inherent styling of that predefined format. In the design, select the headline element and then select **Enclose with | Special Paragraph | Heading 3 (h3)** (alternatively use the Predefined Formats combo box in the toolbar). The predefined format tags are created around the headline element ([screenshot below](#)).

![Screenshot](#)

Notice that the font properties of the contents change and that vertical spacing is added above and below the predefined format. These property values are inherent in the h3 predefined format.

Another use of predefined formats is to group design components in a block so that they can be formatted as a block or assigned inline properties as a group. The most convenient predefined property for this purpose is the **div** predefined format, which creates a block without spacing above or below. In your design, assign the newsitem, dateline, synopsis, and source nodes separate div components. Your design should look something like the screenshot below. Note that the static text "Source: " is also included in the div component that contains the source element, and that the entire newsitem element is inside a div component.
You have now grouped components together in different div blocks. Later in this section, you will learn how to assign styles to such blocks of grouped components.

Assigning components to class attributes
A style rule can be defined for a class of components. For example, all headers can be defined to have a set of common properties (for example, a particular font-family, font-weight, and color). To do this you must do two things: (i) assign the components that are to have the common properties to a single class; (ii) define the styling properties for that class.

In your design, select the h3 tag, and, in the Styles sidebar, select paragraph (to select the predefined format), and the common group of properties. Expand the common group of properties, then double-click in the Value field of the class property and enter header.

This particular instance of the h3 format is now assigned to a class named header. When you define styling properties for the header class (styles from an external stylesheet or global SPS styles), these properties will be applied to all components in the SPS that have the header class.
Adding an external CSS stylesheets to the style repository

Style rules in an external CSS stylesheet can be applied to components in the SPS design. External stylesheets must, however, first be added to the style repository in order for rules in them to be applied to components. In the Style Repository sidebar (in Design View), do the following:

1. Select the External item.
2. Click the Add button in the toolbar of the Style Repository sidebar. This pops up the Open dialog.
3. Browse for the file C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial\QuickStart\QuickStart.css, which is in the (My) Documents folder, and click Open.

The stylesheet is added to the style repository. It contains the following rules that are relevant at this stage:

```
.header {
    font-family: "Arial", sans-serif;
    font-weight: bold;
    color: red;
}

h3 {
    font-size: 12pt;
}
```

The style rules for the header class and h3 element are combined and produce the following HTML output for the headline element.

---

**NanoNull Inc Launches Version 2.0 of NanoPower**

2006-04-01, Boston, USA

Nanonull Inc today launched version 2.0 of its market-leading NanoPower line of hardware and software. The highlights of the new version of NanoPower are improved chip design capabilities in NanoSoft, the software used to design computer chips, and higher precision sand-filtering processes in NanoMeld, the hardware in which the transformation from sand to silicon chips is executed.

*Source: NewTech Online*

---

**Defining global style rules**
Global style rules can be defined for the entire SPS using CSS selectors. The rules are defined directly in the Style Repository sidebar. Create a global style rule for the header class as follows:

1. With Design View active, in the Style Repository sidebar, select the Global item.
2. Click the Add button in the toolbar. This creates an empty rule for the wildcard selector (**), which is highlighted.
3. Type in .header to replace the wildcard as the selector.
4. Expand the color group of properties, and select green from the dropdown list of the color property values (screenshot below).

Where the global style rule defines a property that is also defined in the external stylesheet (the color property), the property value in the global rule takes precedence. In the HTML preview, the contents of the headline will therefore be green. Other property definitions from the external stylesheet (not over-ridden by a property in a global style rule) are retained (in this case, font-family and font-weight).
NanoNull Inc Launches Version 2.0 of NanoPower

2006-04-01, Boston, USA
NanoNull Inc today launched version 2.0 of its market-leading NanoPower line of hardware and software. The highlights of the new version of NanoPower are improved chip design capabilities in NanoSoft, the software used to design computer chips, and higher precision sand-filtering processes in NanoMeld, the hardware in which the transformation from sand to silicon chips is executed.
Source: NewTech Online

Note: Since the global style rule uses the class selector, this rule also applies to RTF, PDF, and Word 2007+ output—in addition to Authentic View and HTML output.

Defining local styles for multiple components at once
Local styles can be defined for multiple components at once. In your design, to specify that the entire text contents of a news item should have Arial as its font, click the div component surrounding the newsitem element and, in the Styles sidebar, in the Styles For column, select 1 paragraph. Then, in the font group of properties, assign Arial as the font-family. This property setting will be inherited by all five descendant predefined formats.

Now, in the design, select the three div components surrounding the dateline, synopsis, and source nodes (by keeping the Shift key pressed as you click each div component). In the Styles sidebar, select 3 paragraphs, then the font group of properties, and set a font-size of 10pt. (The h3 component was not selected because it already has the required font-size of 12pt.)

Finally, in the design, select the div component surrounding the dateline element. In the Styles For column of the Styles sidebar, select 1 paragraph. In the font group of properties, set font-weight to bold and font-style to italic. In the color group of properties, set color to gray. The output of the dateline will look like this

2006-04-01: Boston, USA

Notice that the styling defined for the div component has been applied to the static text within the div component as well (that is, to the colon and the comma).

Defining local styles for a single component
A local style defined on a single component overrides all other styles defined at higher levels of the SPS for that component. In the design, select the headline element and assign it a color of navy (color property in the color group of style properties). The locally defined property
(color:navy) overrides the global style for the .header class (color:green).

Select the div component surrounding the source element. In the Styles sidebar, with the paragraph item in the Styles For column selected, set the color property (in the color group of style properties) to gray. In the font group of style properties, set font-weight to bold. These values are applied to the static text. Remember that in the last section the static text "Source: " was assigned a font-style value of italic. The new properties (font-weight:bold and color:gray) are additional to the font-style:italic property.

Now, in Design View, select the (content) placeholder of the source element. In the Styles For column, with content selected, set the color property (in the color group of style properties) to black. In the font group of properties, set font-weight to normal. The new properties are set on the contents placeholder node of the source element and override the properties defined on the div component (see screenshot below).

---

**Completing the formatting**

To complete the formatting in this section, select the div component on the synopsis element and, in the Predefined Formats combo box in the toolbar, select p. This gives the block the inherent styles of HTML's p element. The HTML preview should now look something like this:

---

**NanoNull Inc Launches Version 2.0 of NanoPower**

*2006-04-01: Boston, USA*

NanoNull Inc today launched version 2.0 of its market-leading NanoPower line of hardware and software. The highlights of the new version of NanoPower are improved chip design capabilities in NanoSoft, the software used to design computer chips; and higher precision sand-filtering processes in NanoMeld, the hardware in which the transformation from sand to silicon chips is executed.

*Source:* NewTech Online

---

After you are done, save the file.

---

---

**Links**

- Next: Using Auto-Calculations
- Previous: Inserting Static Content
- Tutorial Start Page
- Working with CSS Styles
6.5 Using Auto-Calculations

Auto-Calculations are a powerful mechanism for providing additional information from the available XML data. In this section you will add two pieces of information to the design: the total number of news items and the time period covered by the news items in the XML document. Neither piece of information is directly available in the XML document but has to be calculated or manipulated from the available data.

Counting the news item nodes
In the design, do the following:

1. Create space, as shown in the screenshot below, for a line of static text (on which the Auto-Calculation will also be placed). Use the Return key to add new lines and insert a horizontal line below the space you create (see screenshot).

2. Type in the static text "Total number of news items:" as shown in the screenshot above.

3. Apply local styling of your choice to the static text. Do this as described in the section Formatting the Content.

4. Place the cursor after the colon and select Insert | Auto-Calculation | Value. This pops up the Edit XPath Expression dialog (screenshot below). (Alternatively, you can right-click and select the command in the context menu.)
5. In the schema tree, note that the context node is `newsitems`, which is highlighted. Now, in the Expression text box either type in the expression `count(newsitem)` or build the expression using the entry-helper panes below the Expression text box. (Double-click the `count` function (found in the Sequence group of functions) to enter it, then (in the expression in the text box) place the cursor within the parentheses of the function and double-click the `newsitem` node in the schema tree. You can see what the XPath expression returns by clicking the Evaluator button. The result of the evaluation will be in the Results pane (see screenshot below). For a detailed description of the Edit XPath Expression dialog, see the section Edit XPath Expression.
6. Click **OK** to finish. The Auto-Calculation is inserted in the design at the cursor location *(screenshot below)*. Format the Auto-Calculation using **local styles**.

Your HTML output will look like this:

**Summary of News Items**

*Total number of news items:* \(=\text{(AutoCalc)}\)

In the design, insert the static text and Auto-Calculation as shown in the screenshot below. Apply whatever local styling you like.
**Total number of news items:**=(AutoCalc)
**Period covered by news items:**=(AutoCalc)

The HTML preview will look something like this:

**Total number of news items:** 4  
**Period covered by news items:** 4/2006 to 5/2006

After you are done, save the file.

- **Links**
  - Next: Using Conditions  
  - Previous: Formatting the Content  
  - Tutorial Start Page
6.6 Using Conditions

If you look at QuickStart.xml, you will see that each newsitem element has a metainfo child element, which in turn can contain one or more relevance child elements. In the SPS design, you can create a combo box that has a dropdown list which you can populate with unique relevance element values. When the Authentic View user selects an item from the dropdown list in the combo box, that item can be passed as a value to a node in the XML document. A condition can test what the user selection is (by looking up that node) and provide appropriate processing (displays) for each user selection. In this section, you will create a conditional template that displays those news items that have a relevance element that matches the user selection.

We will proceed as follows:

1. Create a combo box in which the Authentic View user can select the byrelevance value. The values in the dropdown list of the combo box are obtained by using an XPath expression, which dynamically compiles a list of all unique relevance node values.
2. Insert a condition around the newsitem element. This condition selects all newsitem elements that have a relevance element with content matching the content of the byrelevance node. The content that is surrounded by a branch of a condition is known as a conditional template.
3. Within the conditional template, list each relevance node of that news item.
4. Highlight the relevance element (in the list of relevance elements) that matches the byrelevance element. This is done by creating a condition to select such relevance elements and then applying special formatting to this conditional template.
5. In the condition for the newsitem element, insert a branch that selects all news items.

Creating the combo box to select unique node values

In the XML document, the node that will contain the user selection is /presswatch/selection/byrelevance. This is the node you will create as the combo box. Do this as follows:

1. Insert the static text “Select by relevance: ” at the head of the document and just below the second Auto-Calculation (screenshot below).

   Total number of news items: :=(AutoCalc)
   Period covered by news items: :=(AutoCalc)
   Select by relevance: |

2. Drag the byrelevance node from the Schema Tree sidebar (screenshot below), and drop it after the newly entered static text.
3. In the context menu that appears, select Create Combo Box. This pops up the dialog shown below.
4. In the Edit Combo Box dialog (screenshot above), select Use XPath Expression and then Use the Same XPath for XML Values and Visible Entries. In the XPath for XML Values and Visible Entries, enter the XPath expression: `distinct-values(//relevance)`. This expression selects unique values of all relevance elements in the XML document.

5. Click OK to finish. The combo box is inserted and the design will look something like this:

```
Total number of news items: 4
Select by relevance: NanoPower
NanoPower
NanoNull
Stockmarket
NanoDiamonds
```

6. Switch to Authentic View. When you click the dropdown arrow of the combo box, notice that the list contains the unique values of all relevance nodes (screenshot below). Check this against the XML document. This is a dynamic listing that will be augmented each time a new relevance value is added to the XML document.

---

**Inserting a condition to display news items having the selected relevance**

The condition selects `newsitem` elements that have a `metainfo/relevance` element with a value that is the same as that selected by the user (and passed to the `/presswatch/selection/byrelevance` element). Insert the condition as follows:

1. Select the contents of the `newsitem` part of the design which is to be contained inside the condition (highlighted in the screenshot below).
2. Select the menu command (or context menu command) **Enclose with | Condition**. This pops up the **Edit XPath Expression dialog**.

3. Enter the expression `metainfo/relevance=/presswatch/selection/byrelevance`. This expression evaluates to true when the value of the `metainfo/relevance` descendant of the current `newsitem` is the same as the value of the `/presswatch/selection/byrelevance` element (the user selection).

4. Click **OK**. The condition is created around the contents of the `newsitem` element (screenshot below).

Note that there is a single branch in this condition. News items for which the condition test evaluates to true are displayed, those for which the condition test does not evaluate to true are not displayed. The condition in this case, therefore, works as a filter. Later in this section, you will add a second branch to this condition.
Inserting the relevance node as a list

In order to display the relevance nodes of each newsitem element, insert them in the design as follows (see screenshot below):

1. Create some vertical space below the div component for the source element and within the end-tag of the conditional template.
2. Type in the static text "Relevance:" and create a predefined format of div around it (highlight the static text and insert the predefined format).
3. Drag the relevance element from the Root elements tree in the Schema Tree sidebar and drop it into the design below the static text Relevance:
4. Create it as a list. (In the context menu that pops up when you drop the node in the design, select Bullets and Numbering, and then select the desired list format.)
5. Apply text formatting to the contents of the list. When you are done, the design should look something like this:

Now, in Authentic View, check the results for different selections of relevance; use the combo box to change the selection.

Making the selected relevance element bold

Some news items have more than one relevance element. In such cases, the design would be improved if the relevance that matches the user-selection were visually highlighted while the others were not. You can do this in the following way:

1. Select the relevance element in the design.
2. Insert a condition, giving it an XPath expression of: ./presswatch/selection/byrelevance. This creates a condition with a single branch (screenshot below) that selects relevance elements that match the byrelevance element.
3. Select the contents placeholder and give it a local formatting (in the Styles sidebar) of bold (font group) and yellow background-color (color group).

4. Right-click the condition and, from the context menu, select Copy Branch.

5. In the Edit XPath Expression dialog that pops up, check the Otherwise check box (top right-hand side).

6. Click OK to finish. A new branch (Otherwise) is created (screenshot below). This condition branch selects all relevance elements that do not match the byrelevance element.

7. Notice that the contents of the Otherwise branch are a copy of the first branch; the contents placeholder is bold and has a yellow background. Remove this formatting (bold and background-color) from the contents placeholder.

You have put a condition with two branches (each with its conditional template) that carries out the following test on each relevance element: If the contents of relevance match those of /presswatch/selection/byrelevance, then the contents of relevance are displayed bold and with a yellow background. Otherwise (the second branch) they are displayed normal. Check this in Authentic View.

---

**Modifying the combo box and inserting a second condition branch**

In the combo box where the Authentic View user selects a byrelevance value, there is no dropdown list option for selecting all news items. To include this option do the following:

1. In Design View, select the combo box.

2. In the Properties sidebar, with combobox selected in the Properties For column, click the Edit button of the Combo box entry value property (in the combo box group of properties).

3. In the Edit Combo Box that pops up, modify the XPath expression from distinct-
values(//relevance) to distinct-values(//relevance), 'All'. This adds the string All to the sequence of items returned by the XPath expression.

4. Check the dropdown list of the combo box in Authentic View (screenshot below).

![Image of combo box with dropdown menu](image)

Now if the user selection is All, then this value (All) is passed to the node /presswatch/selection/byrelevance. The idea is that when the byrelevance node contains the value All, all news items should be displayed.

The condition that displays the news item template has a single branch with the expression metainfo/relevance=/presswatch/selection/byrelevance. Since no metainfo/relevance node has the value All, no news item will be displayed when All is the value of the byrelevance node. What you have to do is create a second branch for the condition, which will test for a value of All. By creating the news item template within this branch, you will be outputting the news item if the test is true. Do this as follows:

1. In Design View, select the news item condition.
2. Right-click the condition and, from the context menu, select Copy Branch.
3. In the Edit XPath Expression dialog that pops up, enter the expression: /presswatch/selection/byrelevance='All'.
4. Click OK to finish. A second branch is created.

The second branch has as its contents the same template as the first branch. What the second branch does is output the news item template if the user selection is All.

After you have completed this section, save the design.
6.7 Using Global Templates and Rest-of-Contents

Global templates are useful for specifying the processing of an element globally. This enables the rules of the global template (defined in one location) to be used at multiple locations in the stylesheet. A global template can be used in two ways:

- The rules of the global template can be copied to the local template.
- A local template (in the main template) can pass processing of that node to the global template. After the global template is executed, processing resumes in the main template. In this case, the global template is said to be invoked or used from the main template.

There are two mechanisms that are used to invoke a global template from the main template:

- A local template references a global template.
- A (rest-of-contents) instruction in the main template applies templates to the descendant elements of the current element (that is, to the rest-of-contents of the current element). If a global template exists for one of the descendant elements, the global template is applied for that element. Otherwise the built-in template for elements is applied. (The built-in template for elements processes child elements and outputs the text content of elements. As a result, the text content of all descendants elements will be output. Note that the values of attributes are not output.)

In this section, you will create a design for the team-members' template using the rest-of-contents instruction and a global template for the global element member.

Inserting the rest-of-contents instruction

The broad structure of the schema is shown in the screenshot below.

The document element presswatch contains three children: (i) selection; (ii) newsitems; and
(iii) team. The main template you have created this far processes the /presswatch element. Within the presswatch element, only the newsitems element is processed. The selection and team elements are not processed within the presswatch element (although selection has been processed within the newsitems element). Inserting the rest-of-contents instruction within presswatch will therefore cause the selection and team elements to be processed.

Insert the rest-of-contents instruction in the design by placing the cursor between the end-tags of newsitems and presswatch, and selecting the menu command or context menu command Insert | Rest of Contents. The rest-of-contents placeholder is inserted (screenshot below).

If you look at the HTML preview, you will see a string of text (screenshot below):

```
AllAndrewBentincka.bentinck@nanonul.comNadiaEdwardsn.edwar
```

This string is the result of the application of the built-in templates to the selection and team elements. The built-in template for elements processes child elements. The built-in template for text nodes outputs the text in the text node. The combined effect of these two built-in templates is to output the text content of all the descendant nodes of the selection and team elements. The text All comes from selection/byrelevance, and is followed by the text output of team/member descendant nodes, first, last, email, in document order. Note that the id attribute of member is not output (because, as an attribute, it is not considered a child of member).

**Creating a global template for selection**

Since the content of selection is not required in the output, you should create an empty global template for selection so that its contents are not processed. Do this as follows:

1. In Design View, right-click selection in the All Global Elements tree in the Schema Tree sidebar.
2. In the context menu that pops up, select Make / Remove Global Template. A global template for selection is created (screenshot below).

```
() selection (content) () selection
```

3. In the global template, click the contents placeholder and press the Delete key of your keyboard. The contents placeholder is deleted.
4. Check the HTML preview. The text All is no longer present in the line of text output by the built-in templates (screenshot below).
Since the global template for `selection` is empty, the child elements of `selection` are not processed.

---

**Creating a global template for `team/member`**

The objective is to create a table to display details of the members of the press monitoring team. This table will be created in a global template for the `team` element. Do this as follows:

1. Create a global template for the element `team` (right-click `team` in the All Global Elements list of the Schema Tree sidebar and select Make / Remove Global Template).
2. In the All Global Elements list, expand the `team` element and drag its `member` child element into the global template of `team` (in the design).
3. In the context menu that pops up when you drop the element into the global template of `team`, select Create Table. This pops up the Create Dynamic Table dialog (screenshot below).

![Create Dynamic Table dialog](image)

4. In the attributes/elements list deselect `@id`, `department` and `telephone` (see screenshot), and click OK. The dynamic table is created.
5. Place the cursor in a cell of the table body, and in the Properties sidebar, with `table` selected in the Properties For column, specify table properties as shown in the screenshot below.
6. Set additional properties as required in the Properties and Styles sidebars. For example, a background color can be set for the header row by placing the cursor in the header row, and with \texttt{trow} selected in the Styles For column of the Styles sidebar, specifying a value for the \texttt{background-color} property (\texttt{color} group). You can also edit the headers, which are strings of static text. Also, if the content placeholder of the \texttt{team} element is still present in the global template, delete it.

The HTML preview of the table will look something like this:

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrew</td>
<td>Bentinck</td>
<td><a href="mailto:a.bentinck@nanonull.com">a.bentinck@nanonull.com</a></td>
</tr>
<tr>
<td>Nadia</td>
<td>Edwards</td>
<td><a href="mailto:n.edwards@nanonull.com">n.edwards@nanonull.com</a></td>
</tr>
<tr>
<td>John</td>
<td>Edwards</td>
<td><a href="mailto:j.edwards@nanonull.com">j.edwards@nanonull.com</a></td>
</tr>
<tr>
<td>Janet</td>
<td>Ashe</td>
<td><a href="mailto:j.ashe@nanonull.com">j.ashe@nanonull.com</a></td>
</tr>
</tbody>
</table>

- **Links**
  - Next: That's It!
6.8 That's It!

Congratulations for having successfully completed the tutorial. You have learned the most important aspects of creating an SPS:

- How to create the structure of the document (main template and global templates).
- How to insert dynamic and static content in the design, using a variety of dynamic and static SPS components.
- How to use CSS styles, in external stylesheets, in global style rules, and in local style rules.
- How to use Auto-Calculations to derive additional information from the available XML data.
- How to use conditions to filter the XML data and how to obtain different outputs depending on values in the XML data.
- How to use global templates and rest-of-contents.

For a more detailed description of these features, see the corresponding sections in the following four sections:

- SPS File: Content
- SPS File: Structure
- SPS File: Advanced Features
- SPS File: Presentation
- SPS File: Additional Functionality

These sections also contain descriptions of several other StyleVision features not encountered in the Quick Start tutorial.

Using the SPS

After completing the SPS, you should also try out the two main uses of SPS:

- Editing XML documents in the Authentic View of XMLSpy or Authentic Desktop. (The Enterprise and Professional editions contain an Authentic View preview tab, which does not have a few features such as sidebars and Text State Icons.) These two products provide a full-feature Authentic View, in which you can try out the sidebars and context menu. To edit QuickStart.xml in Authentic View in XMLSpy or Authentic Desktop, associate the XML file with MyQuickStart.sps and switch to Authentic View.
- Generating XSLT stylesheets for transforming the XML file to HTML, RTF, and PDF output. The XSLT stylesheets can be generated using the File | Save Generated Files command or via the command line. Try generating XSLT stylesheets from MyQuickStart.sps and then using these stylesheets to transform QuickStart.xml.
Chapter 7

Usage Overview
7 Usage Overview

Objectives
SPS documents that you create in StyleVision can be used for two broad purposes:

- To control the display of XML source documents in Authentic View and to enable data to be entered in XML documents or DBs via the Authentic View interface.
- To generate XSLT stylesheets for HTML, RTF, PDF, and Word 2007+ output.

In this way, the SPS can be used to enable XML document editing and to generate HTML, RTF, PDF, and Word 2007+ output from the edited XML document. Additionally, the generated XSLT stylesheets can be used to transform other XML documents based on the same schema as the SPS.

Steps for creating an SPS
Given below is an outline of the steps involved in creating a new SPS.

1. **Assign a schema** to the newly created empty SPS. The schema may be: (i) a schema file (DTD or XML Schema); (ii) an XML Schema generated from a DB (*Enterprise and Professional editions only*); (iii) a schema based on an XBRL taxonomy (*Enterprise edition only*); (iv) a user-defined schema (created directly in StyleVision). This is done in the Design Overview sidebar. If required, additional schemas can be added (via the Design Overview sidebar) so that nodes in multiple XML documents can be addressed. Alternatively, a new SPS can be created directly with a schema via the **File | New** command.

2. **Assign a Working XML File** to the SPS. The Working XML File provides the XML data processed by the SPS when generating Authentic View and output previews. The Working XML File is assigned in the Design Overview sidebar. The Working XML File enables you to preview output in StyleVision. If required, the Working XML Files for additional schemas can be assigned (via the Design Overview sidebar) to enable previews involving multiple XML documents.

3. **Select the required XSLT version**.

4. **Select the Internet Explorer Compatibility** to match the installed Internet Explorer version.

5. The SPS document is designed in **Design View** using the various design components available to the designer. The design process consists of creating a document structure and defining presentation properties. If print output is required, then additional print formatting properties can be specified.

6. The Authentic View and outputs are tested. If modifications to the design are required, these are made and the SPS document is re-tested.

7. If **XSLT files or output files** are required, these are generated.

8. If required, assign a Template XML File. The Template XML File provides the starting data for a new XML document that can be edited in Authentic View using the SPS.

9. The SPS is deployed for use among multiple Authentic View users.

*See also*
- SPS File Structure
- Working with Databases
7.1 SPS and Sources

Creating a new SPS file
To create a new SPS document, select an option from under the File | New (Ctrl+N) command or click the New Design icon in the Standard toolbar. A new SPS document is created and is displayed in Design View. The new document is given a provisional name of SPSX.sps, where X is an integer corresponding to the position of that SPS document in the sequence of new documents created since the application was started.

After a new SPS document is created, the source files for the SPS must be assigned.

Assigning source files for the SPS
There are three types of source files that can be assigned to an SPS:

- Schema sources
- Working XML File
- Template XML File

These source file assignments are made in the Design Overview sidebar. How to make the assignments is described in the section, Design Overview. The significant points about each type of source file are given below.

Schema sources
At least one schema source file must be assigned to an SPS so that a structure for the design document can be created. Optionally, additional schema sources can be assigned. This enables nodes from other documents to be addressed, and thus included, in the SPS. Schema sources are assigned in the Design Overview sidebar. A schema may be an XML Schema file (.xsd file), an XML Schema generated from an XML file, an XML Schema generated from a DB file, a DTD, a schema based on an XBRL taxonomy, or a user-defined schema. One of the assigned schema sources is designated the main schema. The main schema is significant because the document node of the Working XML File associated with the main schema is used as the starting point for the main template. For each schema, one optional Working XML File and one optional Template XML File can be assigned.

Note: If you wish to add a namespace to an SPS or to an XSLT stylesheet being generated from an SPS, the namespace must be added to the top-level schema element of the XML Schema on which the SPS is based.

Working XML File
Each schema source can, optionally, have a Working XML File associated with it. The function of the Working XML File is to provide the XML data source for output previews in StyleVision, and it must therefore be valid according to the schema with which it is associated. The Working XML...
File is assigned in the **Design Overview sidebar**.

---

**Template XML File**

Each schema source can have a [Template XML File](#) optionally associated with it. The function of the [Template XML File](#) is to provide the starting data of the new XML document that is created each time that SPS is opened in the [Authentic View](#) of a product other than StyleVision. The [Template XML File](#) must be valid according to the schema with which it is associated. It is assigned in the **Design Overview sidebar**.

- **See also**
  - [Schema Tree](#)
7.2 Creating the Design

In the SPS design, you specify:

1. **What content** (from the XML document or DB) should go to the output; additionally content can be inserted directly in the SPS for inclusion in the output;
2. **How the output should be structured**; and
3. **What presentation (formatting) properties** are applied to the various parts of the output.

### Content for output

The content for the output can come from:

1. The XML document or DB, or from multiple XML documents and the DB, to which the SPS is applied. Content from the XML document is included in the SPS by dragging the required XML data node from the relevant schema tree in the Schema Tree sidebar and dropping this node at the desired place in the SPS.
2. An external XML document that is accessible to the application (that is, to StyleVision or an Authentic View product). By using the `doc()` function of XPath 2.0 in an Auto-Calculation, content from external XML document sources can be accessed. An XML document accessed via the `doc()` function in an XPath expression does not need to be referenced via the Schema Sources associations.
3. The SPS itself. Text and other content (such as images and tables) can be inserted directly in the SPS using the keyboard and other GUI features. Such input is independent of the XML document.
4. Manipulated dynamic (XML source) data, with the manipulations being achieved using XPath expressions. Manipulations are typically achieved with Auto-Calculations.
5. For the HTML output, JavaScript functions can be used to generate content.

### Structure of output

In the SPS design, the *structure of the output* can be controlled by using either: (i) a procedural approach, in which the output structure is specified in an entry-level template (StyleVision's main template) and can be independent of the structure of the XML document; (ii) a declarative approach, in which template rules are declared for various nodes (StyleVision's global templates), thus generating an output that follows the structure of the XML document; or (iii) a combination of the procedural and declarative approaches. In Design View, you can use a mix of main template and global templates to obtain the desired structure for the output document. The use of Modular SPSs and Design Fragments provides additional flexibility in the way an SPS is structured.

### Presentation (or formatting) of the output

In Design View, presentation properties are applied to design components using CSS styles. Styles can be defined locally on the component, for HTML selectors declared at the document level, and for HTML selectors declared in an external CSS stylesheet. Additionally, certain HTML elements can be applied to components using predefined formats. Specifying presentation
properties is described in detail in the section, Presentation Procedures.

- See also
  - SPS File Structure
  - Design View
7.3 XSLT and XPath Versions

An SPS is essentially an XSLT stylesheet. For each SPS you must set the XSLT version: 1.0, 2.0, or 3.0. You do this by clicking the appropriate toolbar icon: [XSL 1.0] or [XSL 2.0] or [XSL 3.0]. The selection you make determines two things:

- Which of the three XSLT engines in StyleVision is used for transformations; StyleVision has separate XSLT 1.0, XSLT 2.0, and XSLT 3.0 engines.
- What XSLT functionality (1.0, 2.0, or 3.0) is displayed in the interface and allowed in the SPS. For example, XSLT 3.0 uses XPath 3.0, which is a much more powerful language than XPath 1.0 (which is used in XSLT 1.0) or XPath 2.0 (which is used in XSLT 2.0). Additionally, some SPS features, such as the table-of-contents feature, is available only with XSLT 2.0 and XSLT 3.0.

XSLT transformations

XSLT transformations in StyleVision are used: (i) to generate output views in the interface; and (ii) to generate and save output files (HTML, RTF, PDF, and Word 2007+) from within the interface and via StyleVision Server. The XSLT engine used for transformations (Altova XSLT 1.0, 2.0, or 3.0 Engines) corresponds to the XSLT version selected in the SPS.

XSLT functionality in GUI

The functionality appropriate for each XSLT version relates mostly to the use of the correct XPath version (XPath 1.0 for XSLT 1.0, XPath 2.0 for XSLT 2.0, XPath 3.0 for XSLT 3.0). XPath expressions are widely used in StyleVision—most commonly in features such as Auto-Calculations and Conditional Templates—and there are interface mechanisms that require, and help you build, XPath expressions. The functionality of the correct XPath version is automatically made available in the interface according to the XSLT version you select.

See also

* Generated Files
7.4 **Internet Explorer Compatibility**

Internet Explorer (IE) must be installed on the StyleVision machine to correctly display the SPS design (in Design View) and output previews (in Authentic View and HTML Preview). Given below are notes about the IE versions that are supported:

- Internet Explorer 5.5 or higher
- Internet Explorer 6.0 and higher has better XML support and is recommended.
- Internet Explorer 9 (IE9) or higher provides additional features, such as support for more image formats and for new CSS styles. If you plan to use these additional features in your design, you might want to consider using IE9.

---

**IE9 feature-support in StyleVision**

The following features of IE9 or higher are supported in StyleVision:

- Additional image formats supported: TIFF, JPEG XR, and SVG. (SVG documents must be in XML format and must be in the SVG namespace.) These image formats will be displayed in IE9, but not in older versions of IE. For a complete listing of images supported in the various outputs, see [Image Types and Output](#).
- Support for new CSS styles (including CSS3 styles), which are listed below. Application of these styles is limited to Authentic View and HTML output.
  - `background-clip`
  - `background-origin`
  - `background-size`
  - `box-sizing`
  - `box-shadow`
  - `border-radius` (border-*radius)
  - `font-stretch`
  - `ruby-align`
  - `ruby-overhang`
  - `ruby-position`
  - `overflow-x`, `overflow-y`
  - `outline` (outline-color, outline-style, outline-width)
  - `text-align-last` (partial)
  - `text-overflow` (partial)

- Support for the new CSS length function `calc()`
- Support for the new CSS color functions `rgba()`, `hsl()` and `hsla()`
- Support for the new CSS length units `rem`, `vw`, `vm`, `vh` and `ch`
- HTML5 elements that are supported by IE9 can be inserted in the design as [user-defined elements](#).

---

**Design View and IE versions**

You can set up Design View for a specific IE version by specifying, in the Properties dialog, the IE version with which you wish Design View to be compatible. This has the following effects:

- All CSS styles that can be rendered by the selected IE version will be automatically
displayed in the Styles sidebars of StyleVision. (Note, however, that if IE9 is selected, then IE9 must be installed for the IE9-supported CSS styles to be available in the design interface.) For example, if IE9 is installed and IE9 is selected as the compatibility version, then the CSS3 styles supported in IE9 will be available in the design interface.

- HTML elements corresponding to the selected IE version can be entered as predefined formats or as user-defined elements. The HTML element will be rendered in Authentic View and HTML Preview according to how the installed IE version renders this element. For example, if IE9 is installed and IE9 selected as the compatibility version, then the supported HTML5 elements will be rendered in Authentic View and HTML Preview.

---

**Setting up Design View for a specific IE version**

To set up Design View for a specific IE version, select the menu command File | Properties and, in the Output tab, select the required IE (compatibility) version. See File | Properties for details.

---

**Compatibility of older SPS designs with IE9**

If you open an SPS design that has been created for an older IE version, and if the newer IE9 version or higher is installed on the StyleVision machine, then StyleVision will detect the newer version and ask in a dialog whether you wish to change the compatibility to IE9-compatibility. Changing to the new compatibility will provide additional Design View options as indicated above. The appearance of the document in Design View, Authentic View and HTML output will remain unchanged except for table columns, which are handled differently by IE9. If you change the IE compatibility to IE9-compatibility, then check whether the table columns are generated as required. If not, you can modify the properties of the table columns or switch, in the Properties dialog, the IE compatibility back to that of the previously selected IE version.

- **See also**
  - File | Properties
  - Image Types and Output
  - Working with CSS Styles
7.5 SPS and Authentic View

One of the core uses of the SPS you create with StyleVision is to control the input of data and the display of an XML document in Authentic View, which is a document view available in Altova products. With Authentic View, users who are unfamiliar with XML can easily enter and edit XML document content correctly.

A document creation and editing process that involves Authentic View consists of two separate stages:

- **Document design.** The Authentic View of the XML document, which is graphical view, is designed in StyleVision. The design document is an SPS. The SPS not only processes the XML document for display in Authentic View and for final output; it also provides mechanisms, in Authentic View, for **inputting data into the XML file or DB.**

- **Content editing.** This SPS created in the document design stage is linked to the XML document to be edited. (The XML document must be valid according to the schema on which the SPS is based.) An XML document which is linked to an SPS is presented graphically in the Authentic View of an Altova product as the Authentic View of that XML document. When a new Authentic XML document is created, it can be assigned an SPS and then be edited in Authentic View using the document template (Template XML File) and controls specified in the SPS. If an existing XML document is opened and assigned an SPS, the existing data is displayed in Authentic View according to the design in the SPS, and the document can be edited in Authentic View.

The user of Authentic View is not expected to be knowledgeable about either XML or the schema being used for the document. The document display in Authentic View should make content editing as easy and non-technical as possible. It is, therefore, the task of the person who designs the SPS to produce a user-friendly Authentic View display. For detailed information about using Authentic View, see the Authentic View documentation in the user manual of XMLSpy or Authentic Desktop.
SPSs for standard industry schemas

Altova’s Authentic View package includes SPSs for a number of standard industry schemas. Users can therefore immediately create an XML document based on a standard schema in Authentic View. The screenshot below shows a partial Authentic View of the NCA Invoice standard.

![NCA XML Invoice](image)

You can easily customize any of the supplied standard industry SPSs, which are available in the Examples/IndustryStandards folder of your application folder.

See also
- Editing in Authentic View
7.6 Synchronizing StyleVision and Authentic

Each new release of StyleVision contains features that add to the power and capability of Authentic View. However, the following must be taken into account:

- An SPS file created with a later version of StyleVision might be incompatible with an older version of Authentic View.
- New SPS file functionality (created using a later version of StyleVision) will be interpretable only by a corresponding version (or later) of Authentic View.

So, if a later version of StyleVision is used to create an SPS file, all deployed Authentic View products must be synchronized with this version of StyleVision. This means, for example, that if StyleVision 2008 release 2 was used to create an SPS file, then Authentic Desktop 2008 release 2 (or another Authentic View product from this release) must be used to properly edit this SPS file.

Note that a later version of an Authentic View product will be able to interpret SPSs created with previous versions of StyleVision.

---

Synchronization steps when a deployed SPS file is modified using a later version of StyleVision

If an SPS is already deployed among multiple Authentic View users, and if, subsequently, new Authentic View functionality is added to the SPS using a later version of StyleVision, then the developer should go about the task of synchronization in the following sequence:

1. The developer obtains a license key for the new version of Authentic View for himself.
2. The developer successfully tests SPS modifications using the new StyleVision and Authentic View pair.
3. The new version of the Authentic View product is distributed to all Authentic View users.
4. Only after all three steps above have been successfully carried out, should the modified SPS be deployed to Authentic View users.

- See also
  - Authentic View
7.7 Generated Files

In StyleVision, XSLT stylesheets and output files can be generated using the `File | Save Generated Files` command or `StyleVision Server`. Alternatively, if you wish only to validate or transform XML using XSLT, you can do this directly with `RaptorXML(+XBRL) Server`.

The following files can be generated from StyleVision:

- XSLT stylesheets based on the SPS design. Separate XSLT stylesheets are generated for HTML, RTF, PDF, and Word 2007+ output.
- Output files, generated by processing the `Working XML File` assigned in the SPS with the XSLT stylesheets generated from the SPS.

The markup for the output is contained in the SPS. The data for the output is contained in the XML document or DB. It is the XSLT stylesheet that brings markup and data together in the output. Both the XSLT stylesheets as well as the actual output can be previewed in StyleVision in the `Output Views`.

**Note:** If you wish to add a namespace to an SPS or to an XSLT stylesheet being generated from an SPS, the namespace must be added to the top-level `schema` element of the XML Schema on which the SPS is based.

---

Given below are important points to note about the generated documents:

- **HTML output and stylesheets:** (1) The formatting and layout of the generated HTML document will be identical to the HTML Preview of StyleVision and near-identical to the Authentic View of the XML document. (2) Data-input devices (text input fields, check boxes, etc) in the HTML file do not allow input. These data-input devices are intended for XML data input in Authentic View and, though they are translated unchanged into the graphical HTML equivalents, they cannot be used for data-entry in the HTML document.

- **RTF output and stylesheets:** (1) The RTF design requires specifications for paged media. You can provide these specifications (cover page design, left/right pagination, etc) in the `Properties sidebar` and the `Design Tree sidebar`. Note that the design specifications for paged media are used for RTF, PDF (XSL-FO), and Word 2007+ output. (2) If data-input devices have been used in the SPS, then, where possible, these are rendered as graphics on the RTF page. When a data-entry device cannot easily be simulated as a graphic (e.g. check boxes), a substitute presentation is used.

- **PDF output and stylesheets for FO:** (1) To render a document to paged media requires some special specifications, such as for left/right pagination, the location and composition of headers and footers, the design of a cover page, etc. You can design these additional features for your document (using the `Properties sidebar` and `Design Tree sidebar`), and these are included in the XSLT stylesheet for XSL-FO. Note that these design specifications for paged media are used not only for PDF (XSL-FO) output, but also for RTF and Word 2007+. (2) If data-input devices have been used in the SPS, then, where possible, these are rendered as graphics on the PDF page. When a data-entry device cannot easily be simulated as a graphic (e.g. check boxes), a substitute presentation is used.
- **Word 2007+ output and stylesheets**: You can design paged media features, such as pagination, headers and footers, and a cover page using the Properties sidebar and Design Tree sidebar, and these are included in the XSLT stylesheet for Word 2007+. Note that the design specifications for paged media are used not only for Word 2007+ output, but also for PDF (XSL-FO) and RTF output.

---

**RTF output**

RTF output is generated from your XML file in a single step by processing the XML document with the XSLT-for-RTF file generated from the SPS. The properties of the RTF output are defined in the SPS, and you can preview the output in the RTF Preview window. To obtain the RTF file, you must generate it (using File | Save Generated Files or StyleVision Server).

**Note**: If there is a problem with an embedded preview, StyleVision will attempt to open the preview document in an external application (usually MS Word or Adobe Reader). An error message about the embedded preview will appear in StyleVision. If the preview document is opened in an external application, you will need to close the external application before regenerating the temporary output document, otherwise you will get an error message saying the file is being used by another process. You should also close the external application before closing the SPS design, otherwise StyleVision will not be able to close the temporary output document due to the file lock placed on the document by the external application.

---

**PDF output**

Generating PDF output from your XML file is a two-step process:

1. An XSLT transformation is used to transform the XML file into an XSL-FO file (FO file). In the transformation process, page sequencing and page layout properties are added, so that the FO file contains this information in addition to the content.
2. The FO file is processed by an FO processor to generate the PDF.

The XSLT-for-FO and the PDF output can be previewed in the PDF Preview tab. The XSLT-for-FO, the FO document, and the PDF document can be generated using File | Save Generated Files or StyleVision Server. The FO document can also be generated using RaptorX ML.

**Note**: If there is a problem with an embedded preview, StyleVision will attempt to open the preview document in an external application (usually MS Word or Adobe Reader). An error message about the embedded preview will appear in StyleVision. If the preview document is opened in an external application, you will need to close the external application before regenerating the temporary output document, otherwise you will get an error message saying the file is being used by another process. You should also close the external application before closing the SPS design, otherwise StyleVision will not be able to close the temporary output document due to the file lock placed on the document by the external application.
**Word 2007+ output**

Word 2007+ output is generated from your XML file in a single step by processing the XML document with the XSLT-for-Word 2007+ file generated from the SPS. The properties of the Word 2007+ output are defined in the SPS, and you can preview the output in the Word 2007+ Preview window. To obtain the Word 2007+ file, you must generate it (using File | Save Generated Files or StyleVision Server).

**Note:** If there is a problem with an embedded preview, StyleVision will attempt to open the preview document in an external application (usually MS Word or Adobe Reader). An error message about the embedded preview will appear in StyleVision. If the preview document is opened in an external application, you will need to close the external application before regenerating the temporary output document, otherwise you will get an error message saying the file is being used by another process. You should also close the external application before closing the SPS design, otherwise StyleVision will not be able to close the temporary output document due to the file lock placed on the document by the external application.

**See also**

- File menu | Save Generated Files
7.8 Projects in StyleVision

Files that are related to each other can be collected in a project in the Project sidebar (screenshot below). This enables the files in a project to be accessed easily when designing an SPS. For example, an SPS file can be dragged from the Project sidebar to the Design Tree sidebar and created there as a module; or an image file can be dropped into the design as a static image; or a CSS stylesheet can be dragged to the Style Repository sidebar as an external stylesheet.

Creating and saving a project
A new project is created using the Project | Create Project command. When it is created, a project contains separate folders for separate types (see screenshot above). File types are assigned to a folder via the folder's Properties dialog. A project is named when it is saved (with the extension .svp) for the first time. To subsequently change the name of a project, you change the project file's name at its location, using an application such as Windows File Explorer.

Project folders
Folders can be added both to the main project folder as well as to folders within the main project folder and to sub-folders down to an unlimited number of levels. Three types of folders can be added: (i) a project folder; (ii) an external folder (which is added by browsing and selecting); (iii) an external web folder (which is added via a URL). Each of these three folder types is added to the main project folder using the following Project commands, respectively: (i) Add Project Folder to Project; (ii) Add External Project Folder to Project; (iii) Add External Web Folder to Project. To add each of these folder types to a folder or sub-folder within the main project, select the relevant command from the context menu for that folder or sub-folder.

Each folder can be assigned one or more file types in its Properties dialog (screenshot below). To pop up the Properties dialog, right-click the folder for its context menu, and select Properties.
In the Properties dialog, you can edit the folder name and the file type extensions for that folder (each file type extension must be separated by a semi-colon). Project folder names can also be edited by selecting the folder in the Project sidebar, pressing F2, and editing the name. When a folder has file type extensions defined for it, files with that extension, when added using the Add File to Project command, are added to the folder. If more than one folder has the same file type extension defined, the file is added to the first folder in the Project sidebar having that extension. In the Project sidebar, folders can be reordered using drag-and-drop. However, on the first level (that is at the level immediately below the main project folder), folders are ordered as follows: (i) project folders; (ii) external folders; (iii) external web folders.

---

**Project files**

Files can be added both to the main project folder as well as to folders and sub-folders within the main project folder. Files can be added using the following commands in the Project menu:

- **Add Files to Project**: One or more files are selected in a Browse window for addition. Each of the added files goes into the first folder for which its file type extension has been defined.
- **Add Global Resource to Project**: A file is added via a global resource.
- **Add URL to Project**: A file is added via its URL (which is defined in the Add URL to Project dialog).
- **Add Active File to Project**: The active (SPS) file is added to the first folder in the Project sidebar that has .sps defined as its file type extension.
- **Add Active and Related Files to Project**: The active (SPS) file plus related files, such as the schema/s, Working XML Files, CSS files, static image files, etc, are added to the project, in their respective folders as determined by the file type extensions of the folders. This is a very useful command for quickly gathering into a project all the files relevant to a given design.
- **Properties**: When any project file is selected, clicking the Properties icon in the Project Window toolbar or selecting the Properties command from the context menu pops up a window that displays the location of the file. XML files additionally enable you to select an SPS with which the file can be associated (see screenshot below).
When an SPS file is associated with an XML file that is in a project, double-clicking that XML file in the Project window will open the associated SPS file with the XML file assigned as the Working XML File of the SPS. This is useful if multiple XML files use a single SPS. This feature helps to speed up your work by cutting out the bother of browsing for the SPS file and/or Working XML File.

The commands listed above add files to the main project according to the file type extensions of the folders in the main project folder. To add files to specific folders or sub-folders, right-click the required folder, and in the context menu that pops up, select the corresponding command. Within a folder, files are listed in alphabetical order. Note files can also be dragged to another folder. To see the location of a file, click the Properties command in its context menu.

---

**Global resources**

A global resource of file- or folder-type can be added to a folder. A file-type global resource is an alias for a file resource. An alias can have multiple configurations with each configuration pointing to a file resource. So if a global resource is used in a project, it can link to any of the target resources, depending on which configuration is currently active in StyleVision. A folder-type global resource, similarly, is an alias that can target any one of multiple folders according to the configuration that is currently active. If a folder-type global resource is used in the design to identify a file (say, a Working XML File or CSS file), the folder-type global resource will identify a folder only; the path from that folder to the required file will need to be specified additionally. For more information on how to use global resources, see Using Global Resources.

---

**Drag-and-drop**

In the Project sidebar, 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 sidebar.

---

**Using projects**
Files in a project can be used in various ways depending on what kind of file it is. For each file in the Project sidebar, the actions available are listed in its context menu (right-click to display). Additionally, dragging the file to a location where an action can be executed pops up a menu that contains the relevant command/s; in the case of some commands, the command is executed directly the file is dropped at the relevant location. Given below is a list of available actions for various file types.

**SPS Files**
- *Open Design* opens the SPS in a new design window. (This command also becomes available when you drag an SPS file from the Project sidebar into Design View.)
- *Import as Module* imports the SPS as a module in the currently active SPS; the imported file will be listed under the Modules heading in the Design Tree. (You can also import the file as a module by dragging it to the Modules heading in the Design Overview sidebar.)

**XML Files**
- *Edit File in XMLSpy* opens the XML file in XMLSpy.
- *Create New Design* creates a new SPS. The schema for the SPS is an XML Schema generated from the XML document. The Working XML File of the SPS is the XML file. (You can also drag the file into the Main Template bar of Design View to use this command.)
- *Add as New Schema Source* adds an XML Schema generated from the XML file as an additional schema source; the XML file is assigned as the Working XML File for this new schema source. (You can also drag the file to the Schema entry of the Design Overview sidebar to generate a schema and add the schema as a new schema source.)
- *Assign as Working XML File* assigns the XML File as the Working XML File of the SPS. (You can also drag the file to the Working XML entry of the Design Overview sidebar to add it as the Working XML File.)
- *Assign as Template XML File* assigns the XML File as the Template XML File of the SPS. (You can also drag the file to the Template XML entry of the Design Overview sidebar to add it as the Template XML File.)
- **Note:** When an XML file is double-clicked, one of three actions will be executed according to what is specified in the Project tab of the Options dialog: (i) Edit file in XMLSpy; (ii) Create a new design based on the XML file; (iii) Ask the user which action to execute.

**XML Schema / DTD Files**
- *Edit File in XMLSpy* opens the schema file in XMLSpy.
- *Create New Design* creates a new SPS based on the selected schema. (You can also drag the file into the Main Template bar of Design View to create a new SPS with the selected schema as the schema source.)
- *Add as New Schema Source* adds an XML Schema generated from the XML file as an additional schema source; the XML file is assigned as the Working XML File for this new schema source. (You can also drag the file to the Schema entry of the Design Overview sidebar to create an additional schema source.)
- *Assign as Schema File* assigns the selected schema as the schema source of the currently active SPS, replacing the current schema source. This command is most useful for quickly changing schemas, for example, if the schema location has changed or to correct a wrong assignment. (To use this command, you can also drag the file to the Schema entry of the Design Overview sidebar.)
- **Note:** When a schema file is double-clicked, one of three actions will be executed according to what is specified in the Project tab of the Options dialog: (i) Edit file in XMLSpy; (ii) Create a new design based on the schema file; (iii) Ask the user which action to execute.
CSS Files
- *Edit File in XMLSpy* opens the CSS file in XMLSpy.
- *Import into Style Repository* adds the CSS file to the External CSS files of the Style Repository (External heading in the Styles Repository sidebar). (You can also drag the file to the External heading in the Styles Repository sidebar to import the file into the style repository.)

HTML Files
- *Edit File in XMLSpy* opens the HTML file in XMLSpy.
- *Open* opens the HTML file in the default browser.
- *Create New Design* creates a new SPS, in which you can create the schema based on the HTML document. (You can also drag the file into Design View to use this command.)

Image Files
- *Open* opens the image file in the default image viewer / editing application.
- *Insert Image in Design* inserts the image as a static image in the SPS. (You can also insert the image at a particular location by dragging it there.)

All file types
- *Explore Containing Folder* opens a Windows File Explorer window displaying the contents of the folder in which the selected file is located.
- *Cut, Copy, Paste, Delete* commands work in the standard Windows way, cutting and copying the selected file to the clipboard; pasting files from the clipboard; and deleting. These commands also work for a selection of multiple files.
- *Select All* selects all the files in the project.
- *Properties* pops up the Properties dialog, in which the location of the file is given.

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 also 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.

### See also
- [Project sidebar](#)
- [Project menu](#)
7.9 **Catalogs in StyleVision**

StyleVision supports a subset of the OASIS XML catalogs mechanism. The catalog mechanism enables StyleVision 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 StyleVision works as outlined below.

---

**RootCatalog.xml**

When StyleVision 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"
  xsi:schemaLocation="urn:oasis:names:tc:entity:xmlns:xml:catalog Catalog.xsd">
  <nextCatalog catalog="%PersonalFolder%/Altova/%AppAndVersionName%/
  CustomCatalog.xml"/>
  <nextCatalog catalog="CoreCatalog.xml"/>
  <!-- Include all catalogs under common schemas folder on the first directory level -->
  <nextCatalog spy:recurseFrom="%AltovaCommonFolder%/Schemas" catalog="catalog.xml" spy:depth="1"/>
  <!-- Include all catalogs under common XBRL folder on the first directory level -->
  <nextCatalog spy:recurseFrom="%AltovaCommonFolder%/XBRL" catalog="catalog.xml" spy:depth="1"/>
</catalog>
```

In the listing above, notice that in the Schemas and XBRL folders of the folder identified by the variable `%AltovaCommonFolder%` there are catalog files named `catalog.xml`. (The value of the `%AltovaCommonFolder%` variable is given in the table below.)

The catalog files in the Altova Common Folder map the pre-defined public and system identifiers of commonly used schemas (such as SVG and WSDL) and XBRL taxonomies to URIs that point to locally saved copies of the respective schemas. These schemas are installed in the Altova Common Folder when StyleVision is installed. You should take care not to duplicate mappings in these files, as this could lead to errors.

---

**CoreCatalog.xml, CustomCatalog.xml, and Catalog.xml**
In the RootCatalog.xml listing above, notice that CoreCatalog.xml and CustomCatalog.xml are listed for lookup:

- CoreCatalog.xml contains certain Altova-specific mappings for locating schemas in the Altova Common Folder.
- CustomCatalog.xml is a skeleton file in which you can create your own mappings. You can add mappings to CustomCatalog.xml for any schema you require but that is not addressed by the catalog files in the Altova Common Folder. Do this using the supported elements of the OASIS catalog mechanism (see below).
- There are a number of Catalog.xml files in the Altova Common Folder. Each is inside the folder of a specific schema or XBRL taxonomy in the Altova Common Folder, and each maps public and/or system identifiers to URIs that point to locally saved copies of the respective schemas.

Location of catalog files and schemas
The files RootCatalog.xml and CoreCatalog.xml are installed in the StyleVision application folder. The file CustomCatalog.xml is located in your MyDocuments/Altova/StyleVision folder. The catalog.xml files are each in a specific schema folder, these schema folders being inside the folders: %AltovaCommonFolder%\Schemas and %AltovaCommonFolder%\XBRL.

Shell environment variables and Altova variables
Shell environment variables can be used in the nextCatalog element to specify the path to various system locations (see RootCatalog.xml listing above). The following shell environment variables are supported:

```plaintext
%AltovaCommonFolder%
C:\Program Files\Altova\Common2015

%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 folder that stores administrative tools 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, consider the following SVG file:

```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">
```
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 StyleVision, 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
```

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.altova.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.altova.com/schemas/orgchart" uri="C:\MySchemas\OrgChart.xsd"/>
```

---

**The catalog subset supported by StyleVision**

When creating entries in `CustomCatalog.xml` (or any other catalog file that is to be read by StyleVision), 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.
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 `rewritesSystem` 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.

---

**More information**
For more information on catalogs, see the [XML Catalogs specification](https://www.w3.org/TR/xmlcatalogs/).

---

**See also**
- [Generated Files](#)
Chapter 8

SPS File: Content
8 SPS File: Content

This section describes in detail the core procedures used to create and edit SPS document components that are used to create locations in the document design for XML data content. The procedures are listed below and described in detail in the sub-sections of this section. These mechanisms are used to design any kind of template: main, global, or named.

- **Inserting XML Content as Text.** XML data can be inserted in the design by dragging the relevant nodes (element, attribute, type, or CDATA) into the design and creating them as (contents) or (rest-of-contents).
- **User-Defined Templates**
- **User-Defined Elements, XML Text Blocks**
- **Working with Tables.** Tables can be inserted by (i) the SPS designer, directly in the SPS design (static tables) or using XML document sub-structures, and (ii) the Authentic View user.
- **Creating Lists.** Static lists, where the list structure is entered in the SPS design, and dynamic lists, where an XML document sub-structure is created as a list, provide powerful data-ordering capabilities.
- **Using Graphics:** Graphics can be inserted in the SPS design using a variety of methods to determine the target URI (static, dynamic, a combination of both, and unparsed entity URIs).
- **Charts**
- **Using Data-Entry Devices (or Form Controls).** XML data can be input by the Authentic View user via data-entry devices such as input fields and combo boxes. This provides a layer of user help as well as input constraints. Individual nodes in the XML document can be created as data-entry devices.
- **Links**
- **Barcodes**
- **Layout Modules**
- **The Change-To Feature.** This feature enables a different node to be selected as the match for a template and allows a node to be changed to another content type.

- **See also**
  - SPS File Advanced Features
  - SPS File Additional Functionality
8.1 Inserting XML Content as Text

Data from a node in the XML document is included in the design by dragging the corresponding schema node from the Schema Tree window and dropping it into the design. When the schema node is dropped into the design, a menu pops up with options for how the node is to be created in the design (screenshot below).

**Types of schema nodes**
Schema nodes that can be dropped from the Schema Tree sidebar into the design are of three types: (i) element nodes; (ii) attribute nodes; and (iii) datatype nodes.

**Using the Insert Contents toolbar icon**
The Insert Contents icon in the Insert Design Elements toolbar also enables you to insert the contents of a node in the design. Insert contents as follows:

1. Select the Insert Contents icon.
2. Click the location in the design where you wish to insert contents. The Insert Contents Selector pops up (screenshot below).
3. The context of the insertion location in the design is displayed in the XPath Context field. Select the node for which you wish to create contents.

4. Click OK. The contents placeholder is created. If the node you selected is anything other than the context node, additional template tags with the path to the selected node will be created around the contents placeholder.

Outputting text content of nodes
To output the text contents of the node, the node should be created as contents. When a node is created as contents, the node will look something like this in the design document:

In the screenshot above, the Desc element has been created as contents. The output will display the text content of Desc. If Desc has descendant elements, such as Bold and Italic, then the text content of the descendant elements will also be output as part of the contents of Desc. Note that attribute nodes of Desc are not considered its child nodes, and the contents of the attribute nodes will therefore not be output as part of the contents of Desc. Attribute nodes have to be explicitly inserted in order to be processed.
CDATA sections
If CDATA sections are present in the XML document they will be output, and in Authentic View, are indicated with tags when markup is switched on (using the menu command Authentic | Markup). CDATA sections can also be inserted in the XML document when editing the document in Authentic View (via the context menu).

Note: In Authentic View, CDATA sections cannot be inserted into input fields (that is, in text boxes and multiline text boxes). They can only be entered within elements that are displayed in Authentic View as text content components.

In this section
In the sub-sections of this section, we describe other aspects of inserting XML content as text:

• How the text content of a node can be marked up with a predefined format directly when the node is inserted.
• How the structure of the source schema determines the effect of Authentic View usage.
• How descendant nodes not explicitly included within a node can be included for processing. See Rest-of-Contents.

Note: You can create an empty template rule by deleting the (content) placeholder of a node. An empty template rule is useful if you wish to define that some node not be processed, i.e. produce no output.

See also
• Design View Symbols
• Templates and Design Fragments
Inserting Content with a Predefined Format

The text content of a node can be directly inserted with the markup of one of StyleVision's predefined formats. To do this, drag the node from the Schema Tree window and drop it at the desired location. In the menu that pops up, select Create Paragraph (screenshot below).

The predefined format can be changed by selecting the predefined format tag and then choosing some other predefined format from the Format combo box in the toolbar (screenshot below) or using the menu command Insert | Format.

The predefined format can also be changed by changing the value of the paragraph type property of the paragraph group of properties in the Properties window, or by changing the paragraph type via the node-template's context menu command, Enclose With | Special Paragraph.

Each paragraph type has particular formatting features that can be used to advantage. Note that the pre format type enables carriage returns and linefeeds to be output as such instead of them being normalized to whitespace.

- See also
  - Design View Symbols
  - Predefined Formats
Adding Elements in Authentic View

When creating elements in the design, the way you create the elements determines how Authentic View will respond to user actions like pressing the Tab key and clicking the Add... prompt. The basic issue is what elements are created in Authentic View when an element is added by the user. For example, when the user adds an element (say, by clicking the Insert Element icon in the Elements sidebar), what child elements are created automatically?

The most important point to bear in mind is that Authentic View follows the structure specified in the underlying schema. In order to ensure that Authentic View implements the schema structure correctly there are a few design rules you should keep in mind. These are explained below.

Unambiguous content model

A content model is considered unambiguous when it consists of a single sequence (with maxOccurs=1) of child elements (with no choices, groups, substitutions, etc). In such cases, when the element is added, the sequence of child elements is unambiguously known, and they are automatically added. In the screenshot example below, the three child elements are all mandatory and can occur only once.

When the element parent is added in Authentic View, its child elements are automatically inserted (screenshot below). Pressing the tab key takes you to the next element in the sequence.

If the e2 element were optional, then, when the element parent is added in Authentic View, the elements e1 and e3 are automatically inserted, and the element e2 appears in the Elements sidebar so that it can be inserted if desired (screenshot below). Pressing the tab key in e1 takes the user to e3.

The above content model scenario is the only scenario Authentic View considers unambiguous. All other cases are considered ambiguous, and in order for Authentic View to disambiguate and efficiently display the desired elements the design must adhere to a few simple rules. These are explained below.
Ambiguous content model
For Authentic View to correctly and efficiently display elements while an XML document is being edited, the SPS must adhere to the following rules.

- Child elements will be displayed in the order in which they are laid out in the design.
- In order for Authentic View to disambiguate among sibling child elements, all child elements should be laid out in the design document in the required order and within a single parent node. If the sibling relationship is to be maintained in Authentic View, it is incorrect usage to lay out each child element of a single parent inside multiple instances of the parent node.

These two rules are illustrated with the following example.

We consider a content model of an element `parent`, which consists of a single sequence of mandatory child elements. This content model is similar to the unambiguous content model discussed above, with one difference: the single sequence is optional, which makes the content model ambiguous—because the presence of the sequence is not a certainty. If you create a design document as shown in the screenshot below, there will be ambiguity in Authentic View.

The Authentic View of the `parent` element will look like this (since the sequence is optional):

When clicking `add...` pops up a menu of the three child elements:

If you select one of these elements, it will be inserted (screenshot below), but since Authentic View cannot disambiguate the sequence it does not insert any of the remaining two elements, nor does it offer you the opportunity of inserting them:

The correct way to design this content model (following the rules given above) would be to explicitly create the required nodes in the desired order within the single parent node. The design document would look like this:
Note that all three child elements are placed inside a single parent node. The design shown above would produce the following Authentic View:

```
<parent>
  <e1>(content)<e1>
  <e2>(content)<e2>
  <e3>(content)<e3>
<parent>
```

The Authentic View user clicks the respective add element prompt to insert the element and its content.

**Note:**
- If an element can occur multiple times, and if the rules above are followed, then the element appears in the sidebar till the number of occurrences in Authentic View equals the maximum number of occurrences allowed by the schema (maxOccurs).
- Creating each child element inside a separate parent node (see screenshot below) not only creates isolated child–parent relationships for each child element so instantiated; it also increases processing time because the parent node has to be re-traversed in order to locate each child element.

```
<parent>
  <add e1>
  <add e2>
  <add e3>
<parent>
```

- **See also**
  - Inserting XML Content as Text
  - Authentic View Usage
Rest-of-Contents

The rest-of-contents placeholder applies templates to all the remaining child elements of the element for which the template has been created. As an example consider the following:

- An element parent has 4 child elements, child1 to child4.
- In the template for element parent, some processing has been explicitly defined for the child1 and child4 child elements.

This results in only the child1 and child4 child elements being processed. The elements child2 and child3 will not be processed. Now, if the rest-of-contents placeholder is inserted within the template for parent, then, not only will child1 and child4 be processed using the explicitly defined processing rules in the template. Additionally, templates will be applied for the child2 and child3 child elements. If global templates for these are defined then the global templates will be used. Otherwise the built-in default templates (for element, attribute, and text nodes) will be applied.

**Important:** It is important to note what nodes are selected for rest-of-contents.

- As described with the example above, all child element nodes and child text nodes are selected by the rest-of-contents placeholder. (Even invalid child nodes in the XML document will be processed.)
- Attribute nodes are not selected; they are not child nodes, that is, they are not on the child axis of XPath.
- If a global template of a child element is used in the parent template, then the child element does not count as having been used locally. As a result, the rest-of-contents placeholder will also select such child elements. However, if a global template of a child element is "copied locally", then this usage counts as local usage, and the child element will not be selected by the rest-of-contents placeholder.

**Note:** You can create an empty template rule by deleting the (content) placeholder of a node. An empty template rule is useful if you wish to define that some node not be processed, i.e. produce no output.

**See also**
- Inserting XML Content as Text
- Design View Symbols
- Templates and Design Fragments
8.2 User-Defined Templates

User-Defined Templates are templates for items generated by an XPath expression you specify. These items may be atomic values or nodes. In the screenshot below, which shows three User-Defined Templates, note the User-Defined Template icon on the left-hand side of the tags (a green person symbol). User-Defined Templates are very useful because they provide extraordinary flexibility for creating templates. Note, however, that content generated by User-Defined Templates cannot be edited in Authentic View.

The XPath expression of each of the three User-Defined templates shown in the screenshot above do the following:

- Selects a node in a source schema. By using an XPath expression, any node in any of the schema sources can be reached from within any context node. If StyleVision can unambiguously target the specified node, the template will be changed automatically from a User-Defined Template to a normal template, enabling Authentic View editing. If it is a User-Defined Template, this will be indicated by the green User-Defined Template icon on the left-hand side of the template tags.

- Selects a node that fulfills a condition specified by the for construct of XPath 2.0 and XPath 3.0. Such templates can never resolve to normal templates (but will remain User-Defined Templates) because the for construct does not allow StyleVision to unambiguously resolve the target from only the schema information it currently has at its disposal.

- Selects a sequence of atomic values \{1, 2, 3\}. While it is allowed to create a template for an atomic value, you cannot use the contents placeholder within such a template. This is because the xsl:apply-templates instruction (which is what the contents placeholder generates) can only be applied to node items (not atomic values). You could, however, use an Auto-Calculation in combination with some design element such as a list. For example, the User-Defined Template at left would generate the output at right.

Note: If the SPS uses XSLT 1.0, then the XPath expression you enter must return a node-set. Otherwise an error is reported.
**Advantage of using XPath to select template node**

The advantage of selecting a schema node via an XPath expression (User-Defined Templates) is that the power of XPath's path selector mechanism can be used to select any node or sequence of items, as well as to filter or set conditions for the node selection. As a result, specific XML document nodes can be targeted for any given template. For instance, the XPath expression `//Office/Department[@Location="NY"]` will select only those Department nodes that have a Location attribute with a value of NY. Also see the other examples in this section.

**Note:** If an XPath expression contains multiple location path steps, then it is significant—especially for grouping and sorting—whether brackets are placed around the multiple location path steps or not. For example, the XPath expression `/Org/Office/Dept` will be processed differently than `(/Org/Office/Dept)`. For the former expression (without brackets), the processor loops through each location step. For the latter expression (with brackets), all the Dept elements of all Office elements are returned in one undifferentiated nodeset.

<table>
<thead>
<tr>
<th>Brackets</th>
<th>Underlying XSLT Mechanism</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td><code>&lt;xsl:for-each select=&quot;Org&quot;&gt;</code>&lt;br&gt;<code>&lt;xsl:for-each select=&quot;Office&quot;&gt;</code>&lt;br&gt;<code>&lt;xsl:for-each select=&quot;Dept&quot;&gt;</code>&lt;br&gt;...&lt;br&gt;<code>&lt;/xsl:for-each&gt;</code>&lt;br&gt;<code>&lt;/xsl:for-each&gt;</code>&lt;br&gt;<code>&lt;/xsl:for-each&gt;</code></td>
<td>Each Office element has its own Dept population. So grouping and sorting can be done within each Office.</td>
</tr>
<tr>
<td>Yes</td>
<td><code>&lt;xsl:for-each select=&quot;/Org/Office/Dept&quot;&gt;</code>&lt;br&gt;...&lt;br&gt;<code>&lt;/xsl:for-each&gt;</code></td>
<td>The Dept population extends over all Office elements and across Org.</td>
</tr>
</tbody>
</table>

This difference in evaluating XPath expressions can be significant for grouping and sorting.

**Inserting a User-Defined Template**

To insert a User-Defined Template, do the following:

1. Click the **Insert User-Defined Template** icon in the Insert Design Elements toolbar and then click the design location where you wish to insert the template. Alternatively, right-click the design location where you wish to insert the template and, from the context menu that appears, select the **Insert User-Defined Template** command.
2. In the **Edit XPath Expression** dialog that pops up, enter the XPath expression you want, and click **OK**. Note that the context node of the XPath expression will be the node within which you have clicked. An empty node template will be created. Sometimes a joined node is created. When a node is joined, the targeted instance nodes are selected as if at a single level, whereas if a node is not joined (that is if it is split into multiple hierarchic
levels), then the node selection is done by looping through each instance node at every split level. The nodeset returned in both cases of selection (joined and split) is the same unless a grouping or sorting criterion is specified. For a discussion of the effect joined nodes have on the grouping and sorting mechanisms, see Node-Template Operations.

---

**Editing a Template Match**
The node selection of any node template (user-defined or normal) can be changed by using an XPath expression to select the new match expression. To edit the template match of a node template, right-click the node template, then select the **Edit Template Match** command. This pops up the Edit XPath Expression dialog, in which you enter the XPath expression to select the new node. Then click **OK**.

---

**Adding nodes to User-Defined Templates**
If a node from the schema tree is added to a User-Defined Template, the context for the new node will not be known if the User-Defined Template has been created for a node or sequence that cannot be placed in the context of the schema source of the SPS. You will therefore be prompted (**screenshot below**) about how the new node should be referenced: (i) by its name (essentially, a relative path), or (ii) by a full path from the root of the schema source.

Prompting for advice on how to proceed is the default behavior. This default behavior can be changed in the Design tab of the **Tool | Options dialog**.

---

- **See also**
  - SPS File: Content
  - Node-Template Operations
  - User-Defined Elements
8.3 User-Defined Elements, XML Text Blocks

User-Defined Elements and User-Defined XML Text Blocks enable, respectively, (i) any element, and (ii) any XML text block to be inserted into the design. The advantage of these features is that designers are not restricted to adding XML elements and design elements from source schemas and the palette of StyleVision design elements. They can create (i) templates for elements they define (User-Defined Elements), and (ii) independent and self-contained XML code (User-Defined Blocks) that creates objects independently (for example ActiveX objects).

There is one important difference between User-Defined Elements and User-Defined XML Text Blocks. A User-Defined Element is created in the design as a template node for a single XML element (with attributes). All content of this template must be explicitly created. This content consists of the various design elements available to the SPS. A User-Defined XML Text Block may not contain any design element; it is an independent, self-contained block. Since a User-Defined Element is created empty, it does not lend itself for the creation of an object requiring a number of lines of code. For the latter purpose, User-Defined XML Text Blocks should be used.

Note: User-Defined Elements and User-Defined Text Blocks are supported in Authentic View only in the Enterprise Editions of Altova products.

See also
- User-Defined Elements
- User-Defined XML Text Blocks
- User-Defined Templates
User-Defined Elements

User-Defined Elements are elements that you can generate in the output without these elements needing to be in any of the schema sources of the SPS. This means that an element from any namespace (HTML or XSL-FO for example) can be inserted at any location in the design. SPS design elements can then be inserted within the inserted element.

**Note:** User-Defined Elements are supported in Authentic View only in the Enterprise Editions of Altova products.

---

Inserting User-Defined Elements

The mechanism for using User-Defined Elements is as follows:

1. Right-click at the location in the design where you wish to insert the User-Defined Element.
2. From the context menu that appears, select **Insert User-Defined Item | User-Defined Element**.
3. In the dialog that appears (*screenshot below*), enter the element name, the desired attribute-value pairs, and, a namespace declaration for the element if the document does not contain one.

   ![Edit User XML Element](image)

   **Edit User XML Element**

   Enter the element’s name, followed by its attributes as a sequence of names and double-quoted values.

   For example, to generate an HTML ‘horizontal row’ element with 50% width, type:

   ```html
   hr width="50%"
   ```

   For dynamic attribute values, type an XPath expression enclosed in braces:

   ```html
   hr width="\{concat(width-in-pct, \"%\")\}"
   ```

   ![fo:leader xmlns:fo="http://www.w3.org/1999/XSL/Format" leader-length="100%" rule-style="”"]

   In the screenshot above an XSL-FO element called *leader* is created. It has been given a prefix of *fo;* which is bound to the namespace declaration `xmlns:fo="http://www.w3.org/1999/XSL/Format"`. The element has a number of attributes, including `leader-length` and `rule-style`, each with its respective value. The element, its attributes, and its namespace declaration must be entered without the angular tag brackets.

4. Click **OK** to insert the element in the design. The element is displayed in the design as an empty template with start and end tags (*screenshot below*).
5. You can now add content to the template as for any other template. The User-Defined Element may contain static content, dynamic content from the XML document, as well as more additional User-Defined Elements.

**Note:** A User-Defined Element that is intended for a particular output should be enclosed in a suitable output-based condition so as to avoid unexpected results in alternative outputs.

**See also**
- User-Defined XML Text Blocks
- User-Defined Templates
**User-Defined XML Text Blocks**

A User-Defined XML Text Block is an XML fragment that will be inserted into the XSLT code generated by the SPS. It is placed in the SPS design as a self-contained block to which no design element may be added. Such an XML Text Block should therefore be applicable as XSLT code at the location in the stylesheet at which it occurs.

The usefulness of this feature is that it provides the stylesheet designer a mechanism with which to insert XSLT fragments and customized code in the design. For example, an ActiveX object can be inserted within an HTML `SCRIPT` element.

**Note:** This feature will be enabled only in Enterprise editions of Authentic View (that is, in the Enterprise editions of StyleVision, Authentic Desktop, Authentic Browser, and XMLSpy).

---

**Inserting User-Defined XML Text Blocks**

To insert an XML Text Block, do the following:

1. Right-click at the location in the design where you wish to insert the User-Defined Block.
2. From the context menu that appears, select **Insert User-Defined Item | User-Defined Block**.
3. In the dialog that now appears (screenshot below), enter the XML Text Block you wish to insert. Note that the XML text block should be well-formed XML to be accepted by the dialog.

![Edit User XML Text](image)

In the screenshot above an XML Text Block is added that generates an HTML ordered list.

4. Click **OK** to insert the element in the design. The XML Text Block is displayed in the design as a text box.

**Note:** An XML Text Block that is intended for a particular output should be enclosed in a suitable output-based condition so as to avoid unexpected results in alternative outputs.
- **See also**
  - User-Defined Elements
  - User-Defined Templates
8.4 Tables

In an SPS design, two types of tables may be used: SPS tables and CALS/HTML tables. There are differences between the two types, and it is important to understand these. This section contains a detailed description of how to use both types of tables.

SPS tables

An SPS table is a component of an SPS design. It is structured and formatted in the design. It can be created anywhere in the design and any number of SPS tables can be created.

SPS tables are entirely presentational devices and are represented using the presentational vocabulary of Authentic View and the output format. The structure of an SPS table is not represented by nodes in the XML document—although the content of table cells may come from nodes in the XML document.

There are two types of SPS tables:

- **Static tables** are built up, step-by-step, by the person designing the SPS. After the table structure is created, the content of each cell is defined separately. The content of cells can come from random locations in the schema tree and even can be of different types. Note that the rows of a static table do not represent a repeating data structure. This is why the table is said to be static: it has a fixed structure that does not change with the XML content.
- **Dynamic tables** are intended for data structures in the XML document that repeat. They can be created for schema elements that have a substructure—that is, at least one child attribute or element. Any element with a substructure repeats if there is more than one instance of it. Each instance of the element would be a row in the dynamic table, and all or some of its child elements or attributes would be the columns of the table. A dynamic table's structure, therefore, reflects the content of the XML file and changes dynamically with the content.

CALS/HTML tables

The content model of a CALS table or HTML table is defined in the XML document—by extension in the DTD or schema—and follows the respective specification (CALS or HTML). In the SPS design you can then specify that CALS/HTML table/s are to be processed as tables. The XML data structure that represents the CALS/HTML table will in these cases generate table markup for the respective output formats. The formatting of CALS/HTML tables can be specified in the XML instance document or the SPS, or in both.

Shown below is the HTML Preview of an HTML table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Merrimack</td>
<td>6517890</td>
</tr>
<tr>
<td>Joe Concord</td>
<td>6402387</td>
</tr>
</tbody>
</table>
The HTML code fragment for the XML table shown in the illustration above looks like this:

```
<table border="1" width="40%">
  <tbody>
    <tr>
      <td>Name</td>
      <td>Phone</td>
    </tr>
    <tr>
      <td>John Merrimack</td>
      <td>6517890</td>
    </tr>
    <tr>
      <td>Joe Concord</td>
      <td>6402387</td>
    </tr>
  </tbody>
</table>
```

The original XML document might look like this:

```
<phonelist border="1" width="40%">
  <items>
    <person>
      <data>Name</data>
      <data>Phone</data>
    </person>
    ... 
    <person>
      <data>John Merrimack</data>
      <data>6517890</data>
    </person>
    <person>
      <data>Joe Concord</data>
      <data>6402387</data>
    </person>
  </items>
</phonelist>
```

Note that element names in the XML document do not need to have table semantics; the table structure, however, must correspond to the HTML or CALS table model. Also note the following:

- Note that only one XML element can correspond to the HTML column element `<td/>`.
- A CALS/HTML table can be inserted at any location in the XML document where, according to the schema, the element corresponding to the table element is allowed.
- In Authentic View, data is entered directly into table cells. This data is stored as the content of the corresponding CALS/HTML table element.
- The formatting properties of a CALS/HTML table could come from the XML document, or they could be specified in the SPS design.

---

**Summary for the designer**

From the document designer's perspective, the following points should be noted:

- **The structure** of an SPS table is defined in the SPS. The structure of a CALS/HTML table on the other hand is specified in the schema and must follow that of the CALS/HTML table model; the element names in the schema may, however, be different than those in the CALS or HTML table models.
- **Colspans and rowspans** in SPS tables are specified in the SPS. But in CALS/HTML tables, colspans and rowspans are specified in the XML instance document.
- **Table formatting** of SPS tables is specified in the SPS. The formatting of CALS/HTML
tables is specified in the XML instance document and/or the SPS.

- See also
  - Creating Static Tables
  - Creating Dynamic Tables
  - Conditional Processing in Tables
  - CALS/HTML tables
Static Tables

To create a static table, do the following:

1. Use one of the following commands: **Table | Insert Table** or **Insert | Table**, or click the [Insert Table] icon in the Insert Design Elements toolbar.

2. All of these commands pop up the Create Table dialog (screenshot below).

![Create Table Dialog](image)

Click **Static Table**.

3. The Insert Table dialog (screenshot below) pops up, in which you specify the dimensions of the table and specify whether the table should occupy the whole available width.

![Insert Table Dialog](image)

4. Click OK. An empty table with the specified dimensions, as shown below, is created.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5. You can now enter content into table cells using regular StyleVision features. Cell content could be text, or elements dragged from the schema tree, or objects such as images and nested tables. The figure below shows a table containing nested tables.

![Nested Tables](image)

Static SPS tables are especially well-suited for organizing XML data that is randomly situated in the schema hierarchy, or for static content (content not derived from an XML source).
Deleting columns, rows, and tables
To delete a column, row, or table, place the cursor in the column, row, or table to be deleted, and click the menu item **Table | Delete Column**, **Table | Delete Row**, or **Table | Delete Table**, respectively. If you have nested tables, these commands will apply, respectively, to the column, row, and table containing the cursor.

---

**Toolbar table editing icons**
The table editing icons, which are by default in the second row of the toolbar, are shortcuts to the **Table** menu commands. These commands allow you to insert, delete, edit the structure of, and assign formatting properties to the static table. These icons can also be used for dynamic SPS tables. They **cannot be used for CALS/HTML tables**, since **CALS/HTML tables** are not formatted in this way.

---

**Structure of static tables for PDF output**
To ensure that tables are correctly rendered in the PDF output, the number of columns specified for a table (in the FO document) must be correct. (The number of columns can vary from row to row when cells span columns.) The following points should be noted to ensure correct table definition for PDF output:

- The number of columns in a table is the maximum number of columns in any row when all rows are considered. For example, if a table has three rows, with Row1 having 6 columns, Row2 having 5 columns, and Row3 having 7 columns, the number of columns in the table is 7. It is best, therefore, to specify, when inserting the static table, that the table should have 7 columns. Cells in the table can subsequently be joined (using **Table** menu commands) so that they span columns. It is important to note that splitting a cell creates a new column in the graphic view but that is not added to the table's column count. In such an event, the table structure and dimensions will be incorrectly rendered in the PDF.
- If the width of columns is to be specified, the total of all columns should add up to 100% or to an absolute measurement that is smaller than the width of the body-area of the page.
- Make sure that widths are defined for each column, and not only for cells of a table. To check whether column widths have been correctly assigned, click the XSL-FO tab and check the widths of the **fo:table-column** child elements of the **fo:table** tag. For example, consider the table definition below:

  ```xml
  <fo:table width="100%" space-before.optimum="1pt" space-after.optimum="2pt">
    <fo:table-column column-width="proportional-column-width(15)" />
    <fo:table-column column-width="proportional-column-width(15)" />
    <fo:table-column column-width="proportional-column-width(20)" />
    <fo:table-column column-width="proportional-column-width(20)" />
    <fo:table-column column-width="proportional-column-width(30)" />
   </fo:table-body>
  </fo:table>
  ```

  In the listing above, the table is defined as having 5 columns, with the widths of each column being given in percentages.
- **See also**
  - Creating Dynamic Tables
  - Conditional Processing in Tables
  - SPS Tables in Design View
  - Formatting Static and Dynamic Tables
  - CALS/HTML tables
Dynamic Tables

To insert a dynamic table, do the following:

1. Use one of the following commands: **Table | Insert Table** or **Insert | Table**, or click the **Insert Table** icon in the Insert Design Elements toolbar.

2. All of these commands pop up the Create Table dialog (screenshot below). If you clicked the Insert Table icon in the toolbar, the Create Table dialog will pop up when you click at the location in the design where you want to insert the table.

   ![Create Table Dialog](screenshot)

   Click **Dynamic Table**.

3. In the XPath Selector dialog (screenshot below) that pops up, notice that the XPath Context is the context of the insertion location, and it cannot be changed in the dialog. Select the node that is to be created as the dynamic table. In the screenshot below, the context node is `n1:Department`, and the `n1:Person` node has been selected as the node to be created as a table.
If you select the User-defined XPath option, then you can enter an XPath expression to select the node to be created as the dynamic table.

4. Click OK. The Create Dynamic Table dialog (screenshot below) pops up.
5. The child elements and attributes of the element that has been dragged into the Design window are displayed in the "Select rows/columns" list and can be created as columns of the table. Deselect the child nodes that you do not want and select any attribute/element you want to include as columns. (In the figure above, the elements Shares, LeaveTotal, LeaveUsed and LeaveLeft have been deselected.) An explanation of the other options is given below. Click OK when done. Note that columns are created only for child elements and attributes, and for no descendant on a lower level.

**Note:** If you specified a User-defined X Path to select the node to be created as the dynamic table, then StyleVision will probably not know unambiguously which node is being targeted. Consequently, the Create Dynamic Table will, in such cases, not display a list of child attributes/elements to select as the fields (columns) of the table. The table that is created will therefore have to be manually populated with node content. This node content should be child attributes/elements of the node selected to be created as the table.

**Note:** Another way of creating a schema node as a table is to drag the node from the schema tree into the design and to specify, when it is dropped, that it be created as a table.

---

### Table grows down or right

When a table grows top-down, this is what it would look like:

<table>
<thead>
<tr>
<th>name</th>
<th>street</th>
<th>city</th>
<th>state</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;i po:name&gt; (contents)</td>
<td>&lt;i po:street&gt; (contents)</td>
<td>&lt;i po:city&gt; (contents)</td>
<td>&lt;i po:state&gt; (contents)</td>
<td>&lt;i po:zip&gt; (contents)</td>
</tr>
</tbody>
</table>

When a table grows left-right it looks like this:
Headers and footers

Columns and rows can be given headers, which will be the names of the column and row elements. Column headers are created at the top of each column. Row headers are created on the left hand side of a row. To include headers, check the Create Header check-box. If the table grows top-down, creating a header, creates a header row above the table body. If the table grows left-right, creating a header, creates a column header to the left of the table body.

To include footers, check the Create Footer check-box. Footers, like headers, can be created both for columns (at the bottom of columns) and rows (on the right hand side of a row). The footer of numeric columns or rows will sum each column or row if the Summary for Numeric Fields check box is checked.

Via the Table menu, header and footer cells can be joined and split, and rows and columns can be inserted, appended, and deleted; this gives you considerable flexibility in structuring headers and footers. Additionally, headers and footers can contain any type of static or dynamic content, including conditional templates and auto-calculations.

Note: Headers and footers must be created when the dynamic table is defined. You do this by checking the Create Header and Create Footer options in the Create Dynamic Table dialog. Appending or inserting a row within a dynamic table does not create headers or footers but an extra row. The difference is significant. With the Create Header/Footer commands, real headers and footers are added to the top and bottom of a table, respectively, and to the top and bottom of the table on new pages if the table runs on to a new PDF page. If a row is inserted or appended, then the row occurs for each occurrence of the element that has been created as a dynamic table.

Nested dynamic tables

You can nest one dynamic table within another dynamic table if the element for which the nested dynamic table is to be created is a child of the element that has been created as the containing dynamic table. Do the following:

1. Create the outer dynamic table so that the child element to be created as a dynamic table is created as a column.
2. In the dynamic table in Design View, right-click the child element.
3. Select Change to | Table. This pops up the Create Dynamic Table dialog.
4. Define the properties of the nested dynamic table.
To nest a dynamic table in a static table, drag the element to be created as a dynamic table into the required cell of the static table. When you drop it, select Create Table from the context menu that appears.

**Tables for elements with text content**

To create columns (or rows) for child elements, the element being created as a table must have a child element or attribute node. Having a child text node does not work. If you have this kind of situation, then create a child element called, say, Text, and put your text node in the TableElement/Text elements. Now you will be able to create TableElement as a dynamic table. This table will have one column for Text elements. Each row will therefore contain one cell containing the text node in Text, and the rows of the table will correspond to the occurrences of the TableElement element.

**Contents of table body cells**

When you create a dynamic table, you can create the node content as any one of a number of StyleVision components. In the examples above, the table body cells were created as contents; in the Create Dynamic Table dialog, the option for Display Cells As is contents. They could also have been created as data-entry devices. There are two points to note here:

- The setting you select is a global setting for all the table body cells. If you wish to have an individual cell appear differently, edit the cell after you have created the table: right-click in the cell and, in the context menu that appears, select "Change to" and then select the required cell content type.
- If you create cells as element contents, and if the element has descendant elements, then the content of the cell will be a concatenation of the text strings of the element and all its descendant elements.

**Deleting columns, rows, and tables**

To delete a column, row, or table, place the cursor in the column, row, or table to be deleted, and click the menu item Table | Delete Column, Table | Delete Row, or Table | Delete Table, respectively. If you have nested tables, the table immediately containing the cursor will be deleted when the Table | Delete Table command is used.

**Toolbar table editing icons**

The table editing icons in the toolbar are shortcuts to the Table menu commands. These commands allow you to insert, delete, edit the structure of, and assign formatting properties to the dynamic table. These icons can also be used for static tables. They cannot be used for CALS/HTML tables, since CALS/HTML tables are not formatted in this way. CALS/HTML tables
can only be enabled in StyleVision.

---

**Creating dynamic tables in global templates**

You can also create dynamic tables on elements inside global templates. The process works in the same way as for Main Template elements (described above). The important point to note is that, in a global template, a dynamic table can only be created for *descendant elements* of the global template node; it cannot be created for the global template node itself. For example, if you wish to create a dynamic table for the element *authors* within a global template, then this dynamic table must be created within the global template of the parent element of *authors*, say *contributors*. It cannot be created within the global template of the *authors* element.

---

**See also**

- Creating Static Tables
- Conditional Processing in Tables
- SPS Tables in Design View
- Formatting Static and Dynamic Tables
- CALS/HTML tables
Conditional Processing in Tables
Conditional processing can be set on individual columns and rows of static and dynamic tables, as well as on column and row headers, to display or hide the column, row, or header depending on the truth of the condition. If the condition evaluates to true, the column, row, or header is displayed. Otherwise it is not.

---

Adding and editing conditional processing
To add conditional processing to a column, row, or header, right click the respective design component and select Edit Conditional Processing. (In the screenshot below, the column-header design-component at top left is shown highlighted in blue; the second-column design-component is shown outlined in blue; the only row component is below the column-header design-component.)

Clicking the Edit Conditional Processing command pops up the Edit XPath Expression dialog, in which you enter the XPath expression of the condition. Here are some ways in which conditional processing could be used.

- On a column, row, or table, enter the XPath expression false() to hide the column, true() to display it.
- A column is output only if the sum of all the values in that column exceeds a certain integer value.
- A column or row is output only if no cell in that column or row, respectively, is empty.
- A column or row is output only if a certain cell-value exists in that column or row, respectively.

To edit an already created condition, right click the respective design component and select Edit Conditional Processing. In the Edit XPath Expression dialog that pops up, edit the XPath expression that tests the truth of the condition.

---

Removing conditional processing
To remove the conditional processing of a column, row, or header, right click the respective design component and select Clear Conditional Processing.

See also
- Creating Static Tables
- Creating Dynamic Tables
- SPS Tables in Design View
- Formatting Static and Dynamic Tables
- CALS/HTML tables
Tables in Design View

The main components of static and dynamic SPS tables are as shown in the screenshots below with the table markup (Table | View Table Markup) switched on.

The screenshot above shows a simple table that grows top-down and that has a header and footer.

- A column is indicated with a rectangle containing a downward-pointing arrowhead. Column indicators are located at the top of columns. To select an entire column—say, to assign a formatting property to that entire column—click the column indicator of that column.
- A row is indicated with a rectangle containing a rightward-pointing arrow. Click a row indicator to select that entire row.
- In tables that grow top-down (screenshot above), headers and footers are indicated with icons pointing up and down, respectively. In tables that grow left-right, headers and footers are indicated with icons pointing left and right, respectively (screenshot below).
- To select the entire table, click in the top left corner of the table (in the screenshots above and below, the location where the arrow cursor points).
- When any table row or column is selected, it is highlighted with a dark blue background. In the screenshot above, the footer is selected.
- In tables that grow top-down, the element for which the table has been created is shown at the extreme left, outside the column-row grid (screenshot above). In tables that grow left-right, the element for which the table has been created is shown at the top, outside the column-row grid (screenshot below).

After a column or row or table has been selected, styles and/or properties can be set for the selection in the Styles and Properties Windows.

Drag-and-drop functionality

The columns and rows of an SPS table (static or dynamic) can be dragged to alternative locations within the same table and dropped there.
Enclosing and removing templates on rows and columns

A row or column can be enclosed with a template by right-clicking the row or column indicator and, from the context menu that pops up (screenshot below), selecting Enclose With | Template or Enclose With | User-Defined Template. In the next step, you can select a node from the schema tree or enter an XPath expression for a User-Defined Template. A template will be created around the row or column.

A template that is around a row or column can also be removed while leaving the row or column itself intact. To do this, select the template tag and press the Delete key.

The enclosing with, and removing, templates feature is useful if you wish to remove a template without removing the contents of a row or column, and then, if required, enclosing the row or column with another template. Enclosing with a User-Defined Template also allows the use of interesting template-match results within the row or column (via Auto-Calculations, for example).

See also

- Table Menu
- Creating Static Tables
- Creating Dynamic Tables
- Formatting Static and Dynamic Tables
- CALS/HTML tables
- Node-Template Operations
- User-Defined Templates
Table Formatting

Static and dynamic tables can be formatted using:

- HTML table formatting properties (in the Properties sidebar)
- CSS (styling) properties (in the Styles sidebar).

Properties sidebar

The HTML table formatting properties are available in the Properties sidebar (screenshot below). These properties are available in the HTML group of properties for the table component and its sub-components (body, row, column, and cell).

Styles sidebar

The CSS table formatting properties are available in the Styles sidebar (screenshot below). CSS properties are available for the table component and its sub-components (body, row, column, and cell).
**Note:** If all table cells in a row are empty, Internet Explorer collapses the row and the row might therefore not be visible. In this case, you should use the HTML workaround of putting a non-breaking space in the appropriate cell/s.

---

**Vertical text**

Text in table cells can be rotated 90 degrees clockwise or anti-clockwise, so that the text is vertical, reading from top-to-bottom or bottom-to-top, respectively. To do this, in the design, select the content in the table cell that is to be rotated and, in the Properties sidebar (screenshot below), select tcell. In the *Table Cell* group of properties, select the required value for the *Orientation* property.

---

Note the following points:

- The rotation will be applied to the output, but will not be be displayed in the design.
- This property is intended to be applied to text and should not be used for other content.
• Besides being applicable to text in table cells, the property can also be applied to text in Text boxes.

---

**Table formatting via Properties and Styles**

Some formatting properties are available in both the Properties sidebar as well as in the Styles sidebar. The table below lists some of the more important table properties available in both sidebars.

<table>
<thead>
<tr>
<th>Table component</th>
<th>Properties sidebar</th>
<th>Styles sidebar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>border, frame, rules; cellpadding, cellspacing; bgcolor; height, width (overriden by height, width in Styles sidebar if the latter exist); align</td>
<td>borders and padding in Box styles; height, width in Details group (they override height and width in Properties sidebar); color, font, and text styles</td>
</tr>
<tr>
<td>Body</td>
<td>align, valign</td>
<td>height, vertical-align; color, font, and text styles</td>
</tr>
<tr>
<td>Column</td>
<td>align, valign</td>
<td>width, vertical-align; color, font, and text styles; box styles</td>
</tr>
<tr>
<td>Row</td>
<td>align, valign</td>
<td>height, vertical-align; color, font, and text styles; box styles</td>
</tr>
<tr>
<td>Cell</td>
<td>align, valign</td>
<td>height, width, vertical-align; color, font, and text styles; box styles</td>
</tr>
</tbody>
</table>

---

**Height and width**

The height and width of tables, rows, columns, and cells must be set in the Styles sidebar (in the Details group of styles). When a table, column, or row is resized in the display by using the mouse, the altered values are entered automatically in the appropriate style in the Styles sidebar. Note, however, that the height and width styles are not supported for cells that are spanned (row-spanned or column-spanned).

---

**Centering a table**

To center a table, set the **align** property in the HTML group of table properties to **center**. The **align** property can be accessed by selecting the table, then selecting the menu command **Table | Table Properties**. Alternatively, the property is available in the HTML group of properties in the Properties sidebar.

Centering the table in the PDF output will require additional settings according to the FOP...
processor you are using. According to the FO specification the correct way to center a table is to surround the `fo:table` element with an `fo:table-and-caption` element and to set the `text-align` attribute of the `fo:table-and-caption` element to `center`. Stylevision does not automatically create an `fo:table-and-caption` element when a table is inserted in the design, but you can add this element as a User-Defined Element. If you are using the Apache FOP processor, however, you should note that the `fo:table-and-caption` element might not be supported, depending on which FOP version you are using. In this case there is a simple workaround: Make the table a fixed-width table. Do this by specifying a length value, such as 4in or 120mm, as the value of the `width` property of the HTML group of table properties (accessed via the menu command Table | Table Properties).

---

**Giving alternating rows different background colors**

If you want alternating background colors for the rows of your dynamic table, do the following:

1. Select the row indicator of the row for which alternating background colors are required. Bear in mind that, this being a dynamic table, one element is being created as a row, and the design contains a single row, which corresponds to the element being created as a table.
2. With the row indicator selected, in the Properties sidebar, click the Properties for: `trow`.
3. Select the `bgcolor` property.
4. Click the XPath icon in the toolbar of the Properties window, and, in the Edit XPath Expression dialog that appears, enter an XPath expression similar to this:

   ```
   if ( position() mod 2 = 0 ) then "white" else "gray"
   ```

   This XPath expression specifies a `bgcolor` of white for even-numbered rows and a `bgcolor` of gray for odd-numbered rows.

You can extend the above principle to provide even more complex formatting.

---

**Numbering the rows of a dynamic table**

You can number the rows of a dynamic table by using the `position()` function of XPath. To do this, first insert a column in the table to hold the numbers, then insert an Auto-Calculation in the cell of this column with an XPath of: `position()`. Since the context node is the element that corresponds to the row of the dynamic table, the `position()` function returns the position of each row element in the set of all row elements.

---

**Table headers and footers in PDF output**

If a table flows over on to more than one page, then the table header and footer appear on each page that contains the table. The following points should be noted:

- If the footer contains Auto-Calculations, the footer that appears at the end of the table segment on each page contains the Auto-Calculations for the whole table—not those for only the table segment on that page.
• The header and footer will not be turned off for individual pages (for example, if you want a footer only at the end of the table and not at the end of each page).

In order to omit the header or footer being displayed each time the page breaks, use the `table-omit-header-at-break` and/or `table-omit-footer-at-break` properties (attributes) on the `table` element. These properties are available in the Styles sidebar, in the XSL-FO group of properties for the table. To omit the header or footer when the page breaks, specify a value of `true` for the respective attribute. (Note that the default value is `false`. So not specifying these properties has the effect of inserting headers and footers whenever there is a break.)

---

**Hyphenating content of table cells**

If you wish to hyphenate text in table cells, you must explicitly set the hyphenate option for the respective block/s.

---

### See also

- Creating Static Tables
- Creating Dynamic Tables
- SPS Tables in Design View
- CALS/HTML tables
Row and Column Display

For tables, the following row and column display options are available in the HTML output only. These features are not supported in Authentic View and they require XSLT 2.0 or XSLT 3.0 to be selected as the XSLT version of the SPS.

- Empty rows and columns can be automatically hidden.
- Each column can have a Close button, which enables the user to hide individual columns.
- Row elements with descendant relationships can be displayed with expand/collapse buttons.

Hiding empty rows and columns by default

To hide empty rows and/or columns in the HTML output, do the following:

1. In Design View, select the table or any part of it (column, row, cell).
2. In the Properties sidebar, select properties for Table, and the Table group of properties (screenshot below).
3. Select the required value for the Hide Columns and Hide Rows properties. The options for each of these two properties are the same: Never, If empty, and If body empty. The If empty option hides the column or row if the entire column/row (including header and footer) is empty. If body empty requires only that the body be empty.

Note: If a non-XBRL table has row or column spans (where cells of a row or a column have been joined), the hiding of empty rows and columns might not work.

User interaction to hide columns expand/collapse rows

It can be specified in the design that each table column contain a Close button in the HTML output (see screenshot below). The user can then hide individual columns by clicking the Close button. After the user hides a column, a plus symbol appears in the first column (see screenshot below). Clicking this symbol re-displays all hidden columns.
Also, row elements that have descendant elements can be displayed in the HTML output with an expand/collapse (plus/minus) symbol next to it (see screenshot above). Clicking these symbols in the HTML output expands or collapses that row element. In the design, you can specify indentation for individual rows using CSS properties.

The settings for these two features are made in the Interactive group of properties of the Table properties (screenshot below).

The options for both properties are Yes (to add the feature) and No (to not add the feature).

**See also**
- Creating Static Tables
- Creating Dynamic Tables
- SPS Tables in Design View
- Formatting Static and Dynamic Tables
- CALS/HTML tables
CALS/HTML Tables

A CALS/HTML table is a hierarchical XML structure, the elements of which: (i) define the structure of a CALS or HTML table, (ii) specify the formatting of that table, and (iii) contain the cell contents of that table. This XML structure must correspond exactly to the CALS or HTML table model.

To create a CALS/HTML table in the design, do the following:

1. Define the XML structure as a CALS/HTML table structure
2. Specify formatting styles for the table
3. Insert the CALS/HTML table in the SPS design

Enabling CALS/HTML table structures for output

An XML document may have a data structure that defines the structure and content of a table. For example, the following XML data structure corresponds to the HTML table model and in fact has the same element names as those in the HTML table model:

```xml
<table>
  <tbody>
    <tr>
      <td/>
    </tr>
  </tbody>
</table>
```

Alternatively, the XML data structure could have a structure corresponding to the HTML table model but different element names than in the HTML table model. For example:

```xml
<semester>
  <subject>
    <class>
      <student/>
    </class>
  </subject>
</semester>
```

This table structure, which is defined in the XML document, can be used to directly generate a table in the various output formats. To do this you need to define this XML data structure as a CALS or HTML table. If the XML data structure is not defined as a CALS or HTML (the default), the elements in the data structure will be treated as ordinary non-table elements and no table markup will be added to the output document.

To enable CALS/HTML table markup in the output do the following:

1. Select the command Table | Edit CALS/HTML Tables.
2. In the dialog that pops up (screenshot below), add an entry for the XML data structure you wish to use as a CALS/HTML table, according to whether the data structure follows the CALS or HTML table model. If the XML data structure is not defined as a CALS or HTML (the default), the elements in the data structure will be treated as ordinary non-table elements and no table markup will be added to the output document.
schema as a CALS or an HTML table element, click the Add CALS/HTML table button in
the top left part of the dialog and then select either the **Add CALS Table** command or
the **Add HTML Table** command. (In the screenshot below, the elements `table` and
`informaltable` have been enabled as CALS tables (as well as HTML tables).) Click **OK**
to confirm.

3. A dialog (Edit CALS Table or Edit HTML table) appears showing the elements of the table
   type you selected *(screenshot below)*. The element names that are listed in this dialog
   are, by default, the element names in the selected table model (CALS or HTML). If the
   SPS schema contains elements with the same names as the names of the CALS/HTML
   table model, then the names are shown in black *(as in the screenshot below)*. If a listed
   element name is not present in the SPS schema, that element name is listed in red. You
   can change a listed element name to match a schema name by double-clicking in the
   relevant **Element Name** field and editing the name.

4. Click **OK** to define this XML data structure as a CALS or HTML table.
5. You can add entries for as many XML data structures as you like *(see screenshot in Step
   2 above)*. The same main element can be used once each for CALS and HTML table
   types.
6. After you have finished defining the XML data structures you wish to enable as CALS/
HTML tables, click OK to finish.

If a CALS/HTML table has been defined and the XML data structure is correctly inserted as a CALS/HTML table, then the data structure will be sent to the output as a table. To remove a CALS/HTML table definition, in the Edit CALS/HTML table dialog select the definition you wish to delete and click the Delete button at the top right of the Define CALS/HTML Tables pane.

---

**Table formatting**

CALS/HTML tables receive their formatting in two ways:

1. From formatting attributes in the source XML document. The CALS and HTML table models allow for formatting attributes. If such attributes exist in the source XML document they are passed to the presentation attributes of the output's table markup.

2. Each individual element in the table can be formatted in the Styles column of the Edit CALS Table dialog or Edit HTML Table dialog (see screenshot below).

![Edit HTML Table](image)

To assign a style to a particular element, click the **Add Styles** button for that element and assign the required styles in the **Styles sidebar** that pops up. Each style is added as an individual CSS attribute to the particular element. Note that a style added via the **style** attribute will have higher priority than a style added as an individual CSS attribute (such as **bgcolor**). For example, in `<thead style="background-color: red" bgcolor="blue"/>` the **style="background-color: red"** attribute will have priority over the **bgcolor="blue"** attribute.

To remove a style that has been assigned to an element in the CALS/HTML table definition, select that element (for example in the screenshot above the **thead** element has been selected) and click the **Delete** button. The styles for that element will be removed.
Inserting a CALS/HTML table in the design

A CALS/HTML table structure can be inserted in the design in two ways:

1. The parent of the table element is inserted in the design as (contents). When the contents of the parent are processed, the table element will be processed. If CALS/HTML table output is enabled, then the element is output as a table. Otherwise it is output as text.

2. The table element can be dragged from the Schema Tree. When it is dropped at the desired location in the design, it can be created as a CALS/HTML table (with the Create CALS/HTML Table command). If the element has not been defined as a CALS/HTML table, the Insert CALS/HTML Tables dialog (screenshot below) pops up and you can define the element as a CALS or HTML table.

![Insert CALS/HTML Table dialog](screenshot below)

If the element has been created in the design as a CALS/HTML table, a placeholder for the CALS/HTML table design element is inserted at the location (screenshot below).

![Sample HTML table](screenshot below)

Global templates of table elements

If global templates of the following table elements are created they will be used in the CALS/HTML table output. For CALS tables: title and entry. For HTML tables: caption, th, and td.

Example files

Example files are in the Examples project folder (in the Project window of the GUI).
See also
- Creating Static Tables
- Creating Dynamic Tables
8.5 Lists

There are two types of lists that can be created in the SPS:

- **Static lists**, which are lists, the contents of which are entered directly in the SPS. The list structure is not dynamically derived from the structure of the XML document.
- **Dynamic lists**, which are lists that derive their structure and contents dynamically from the XML document.

How to create these two list types are described in detail in the sub-sections of this section.

- See also
  - Working with Tables
Static Lists

A static list is one in which list item contents are entered directly in the SPS. To create a static list, do the following:

1. Place the cursor at the location in the design where you wish to create the static list and select the Insert | Bullets and Numbering menu command (or click the Bullets and Numbering icon in the Insert Design Elements toolbar). This pops up a dialog asking whether you wish to create a static list or dynamic list (screenshot below).

2. Click Static List. This pops up the Bullets and Numbering dialog (screenshot below).

3. Select the desired list item marker and click OK. An empty list item is created.
4. Type in the text of the first list item.
5. Press Enter to create a new list item.

To create a nested list, place the cursor inside the list item that is to contain the nested list and click the Insert | Bullets and Numbering menu command. Then use the procedure described above once again.

Note: You can also create a static list by placing the cursor at the location where the list is to be created and clicking either the Bullets icon or Numbering icon in the Bullets or
Numbering icons in the Formatting toolbar. The first list item will be created at the cursor insertion point.

---

**Changing static text to a list**

To change static text to a list, do the following:

Highlight the text you wish to change to a list, select the command **Enclose With | Bullets and Numbering**, choose the desired marker type, and click **OK**. If the text contains a CR-LF, carriage-return and/or linefeed (inserted by pressing the **Enter** key), then separate list items are created for each text fragment separated by a CR-LF. If a text fragment within a line is highlighted, then that text is created as the list-item of a single-item list; you can add an unlimited number of additional list items by clicking **Enter** as many times as required. Note that the **Enclose With | Bullets and Numbering** command can also be accessed via the context menu.

- **See also**
  - Dynamic Lists
  - Bullets and Numbering
Dynamic Lists

Dynamic lists display the content of a set of sibling nodes of the same name, with each node represented as a single list item in the list. The element, the instances of which are to appear as the list items of the list, is created as the list. The mechanism and usage are explained below.

General usage mechanism

- Any element can be created as a list.
- When an element is created as a list, the instances of that element are created as the items of the list. For example, if in a department element, there are several person elements (i.e. instances), and you wanted to create a list of all the persons in the department, then you must create the person element as the list.
- Once the list has been created for the element, you can modify the appearance or content of the list or list item by inserting additional static or dynamic content such as text, Auto-Calculations, dynamic content, etc.

Creating a dynamic list

Create a dynamic list as follows:

1. Place the cursor at the location in the design where you wish to create the dynamic list and select the Insert | Bullets and Numbering menu command. This pops up a dialog asking whether you wish to create a static list or dynamic list (screenshot below).

2. Click Dynamic List. This pops up the XPath Selector dialog (screenshot below).
3. In the XPath Selector dialog, notice that the XPath Context is the context of the insertion location, and that it cannot be changed in the dialog. Select the node that is to be created as the dynamic list. In the screenshot below, the context node is n1:Department, and the n1:Person node has been selected as the node to be created as a list. This means that the content of each n1:Person node will be created as an item in the list.
If you select the User-defined XPath option, then you can enter an XPath expression to select the node to be created as the dynamic table. Clicking **OK** pops up the Bullets and Numbering dialog described in the next step.

4. In the Bullets and Numbering dialog, select the kind of list you wish to create. You can choose from a bulleted list (with a bullet, circle, or square as the list item marker), or a numbered list. Clicking **OK** creates the list with the type of list item marker you selected.
See also
- Static Lists
- Bullets and Numbering
8.6 Graphics

There are two ways in which graphics are used in an SPS:

- As images in the design document, and
- As Authentic View toolbar icons for applying markup to the XML document (custom toolbar buttons).

When inserting images in the design document, the location of the image can be specified directly in the SPS (by the SPS designer) or can be taken or derived from a node in the XML document. How to specify the location of the image is described in the section Image URIs. What type of images are supported in the various outputs are listed in the section Image Types and Output. The section Reference | Authentic Menu | Custom toolbar buttons describes how toolbar icons for Authentic View can be defined.

---

**Image properties**

Images can be set in the Properties window. Do this as follows. Select the image in the design. Then, in the Properties window, (i) select image in the Properties for column, (ii) select the required property group, and (iii) within the selected property group, select the the required property. For example, to set the height and width of the image, set the height and width properties in the HTML group of properties.

- See also
  - Insert Image
  - Blueprint images
Images: URIs and Inline Data

Images can be inserted at any location in the design document. These images will be displayed in Authentic View and the output documents; in Design View, inserted images are indicated with thumbnails or placeholders.

To insert an image, click the Insert | Image menu command, which pops up the Insert Image dialog (screenshot below).

Images can be accessed in two ways:

- The image is a file, which is accessed by entering its URI in the Insert Image dialog.
- The image is encoded as Base-16 or Base-64 text in an XML file.

Inserting an image file

An image file is inserted in the design by specifying its URI. This file is accessed at runtime and placed in the document. There are three ways in which the URI of the image can be entered in the Insert Image dialog (screenshot above):

- In the Static tab, the URI is entered directly as an absolute or relative URI. For example, nanonull.gif (relative URI; see section below), and C:/images/nanonull.gif (absolute URI).
- In the Dynamic tab, as an XPath expression that selects a node containing either (i) a URI (absolute or relative), or (ii) an unparsed entity name. For example, the entry image/@location would select the location attribute of the image element that is the child of the context node (that is, the node within which the image is inserted). The location node in the XML document would contain the image URI. How to use unparsed entities is described in the section Unparsed Entity URIs.
- In the Static and Dynamic tab, an XPath expression in the Dynamic part can be prefixed and/or suffixed with static entries (text). For example, the static prefix could be C:/XYZCompany/Personnel/Photos/; the dynamic part could be concat(First, Last); and the static suffix could be .png. This would result in an absolute URI something like: C:/XYZCompany/Personnel/Photos/JohnDoe.png.
Inserting an image that is encoded text

An image can be stored in an XML file as Base-16 or Base-64 encoded text. The advantage of this is that the image does not have to be accessed from a separate file (linked to it), but is present as text in the source XML file. To insert an image that is available as encoded text in the XML source, use the Inline Data tab of the Insert Image dialog (see screenshot below).

![Insert Image Dialog](image.png)

Use an XPath expression to locate the node in the XML document that contains the encoded text of the image. Select an option from the Image Format combo box to indicate in what format the image file must be generated. (An image file is generated from the encoded text data, and this file is then used in the output document.) In the Encoding combo box, select the encoding that has been used in the source XML. This enables StyleVision to correctly read the encoded text (by using the encoding format you specify).

The Image File Settings dialog (accessed by clicking the Image File Settings button) enables you to give a name for the image file that will be created. You can choose not to provide a name, in which case StyleVision will, by default, generate a name.

Accessing the image for output

The image is accessed in different ways and at different times in the processes that produce the different output documents. The following points should be noted:

- Note the output formats available for your edition: (i) HTML in Basic Edition; (ii) HTML and RTF in Professional; (iii) HTML, RTF, PDF, and Word 2007+ in Enterprise Edition.
- For Design View and Authentic View in StyleVision, as well as for Authentic View in Altova products, you can set, in the Properties dialog, whether relative paths to images should be relative to the SPS or to the XML file.
- For HTML output, the URI of the image is passed to the HTML file and the image is accessed by the browser. So, if the path to the image is relative, it must be relative to the location of the HTML file. For the HTML Preview in StyleVision, a temporary HTML file is created in the same folder as the SPS file, so, for rendition in HTML Preview, relative paths must be relative to this location.
- For RTF output, the URI of the image is passed as an object link to the RTF file and is accessed by the RTF application (typically MS Word) when the file is opened. If the URI is relative, it must be relative to the location of the RTF file. For the RTF Preview in StyleVision, a temporary RTF file is created in the same folder as the SPS file, so, for rendition in RTF Preview, relative paths must be relative to this location.
For PDF output, the URI is passed to the FO document and the image is accessed when the FO document is processed with the FO processor. This means that, if the URI is relative, it must be relative to the FO document at the time the FO document is processed. For the PDF Preview in StyleVision, a temporary FO file is created in the same folder as the SPS file, so, for rendition in PDF Preview, relative paths must be relative to this location.

Whether the URI is relative or absolute, the image must be physically accessible to the process that renders it.

Editing image properties
To edit an image, right-click the image placeholder in Design View, and select Edit URL from the context menu. This pops up the Edit Image dialog, which is the same as the Insert Image dialog (screenshot above) and in which you can make the required modifications. The Edit Image dialog can also be accessed via the URL property of the image group of properties in the Properties window. The image group of properties also includes the alt property, which specifies alternative text for the image.

Deleting images
To delete an image, select the image and press the Delete key.

See also
- Image Types and Output
- Unparsed Entity URIs
Image Types and Output

The table below shows the image types supported by StyleVision in the various output formats supported by StyleVision. Note that different editions of StyleVision support different sets of output formats: Enterprise Edition, HTML, Authentic, RTF, PDF, and Word 2007+; Professional Edition, HTML, Authentic, RTF; Basic Edition, HTML.

<table>
<thead>
<tr>
<th>Image Type</th>
<th>Authentic</th>
<th>HTML</th>
<th>RTF</th>
<th>PDF</th>
<th>Word 2007+</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GIF</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PNG</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BMP</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TIFF</td>
<td>Yes*</td>
<td>Yes*</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SVG</td>
<td>Yes*</td>
<td>Yes*</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>JPEG XR</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

* See notes immediately below

Note the following points:

- In Design View, images will be displayed only if their location is a static URL (i.e. directly entered in the SPS).
- For the display of TIFF and SVG images in Authentic View and HTML View, Internet Explorer 9 or higher is required.
- In RTF output, TIFF images can only be linked, not embedded. So re-sizing is not possible.
- SVG documents must be in XML format and must be in the SVG namespace.
- FOP reports an error if an image file cannot be located and does not generate a PDF.
- If FOP is being used to produce PDF, rendering PNG images requires that the JI MI image library be installed and accessible to FOP.
- For more details about FOP's graphics handling, visit the FOP website.

Example file

An example file, Images.sps, is located in the folder:

C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial\Images

Image resizing in RTF output

Resizing an image in the RTF output is only supported for JPG and PNG images. The following points should be noted:
- Resizing is supported only in designs that use XSLT 2.0 or XSLT 3.0, not XSLT 1.0.
- The `height` and `width` attributes of the image must be set in the Details group of the Styles sidebar. The `height` and `width` attributes in the HTML group of the Properties sidebar are not used.
- Only absolute units (px, cm, in, etc) are supported. Percentage values are not supported.
- JPG and PNG images are embedded in the RTF file. This embedding is implemented using a proprietary Altova XSLT 2.0 extension functionality.

Image embedding in RTF and Word 2007+
In the RTF and Word 2007+ output, images can either be embedded (when XSLT 2.0 is used) or linked. This setting is made for each SPS individually. To embed images, do the following:

1. With the required SPS active, open the Properties dialog (File | Properties).
2. Check the Embed Images check box (default setting is checked). Note that images will only be embedded if XSLT 2.0 or XSLT 3.0 is set as the XSLT version of the active SPS.
3. Click OK and save the SPS. The setting is saved for the active SPS.

To make this setting for another SPS, make this SPS active and repeat the steps listed above.

If the Embed Images check box is not checked, images will be linked according to the image file path specified in the images properties (select the image and select URL in the Properties sidebar). For information about how paths are resolved, see the section Image URLs.

**Note:** The RTF format supports embedded images only for EMF, JPG, and PNG files.

SVG images in HTML
When an external SVG file with code for mouse events is used as an image, the SVG file is rendered within the image and will no longer be interactive. This limitation can be overcome by using the external SVG image file as an object or by including the SVG code fragment as a User-Defined XML Block.

Given below are the three ways in which SVG images can be included in a web page.

1. **External SVG inserted as image:** This generates an `<img>` in the generated HTML file, and interactivity will be lost.
2. **External SVG inserted as an object via the User-Defined Element feature** (see screenshot below). Be sure to insert the `type` attribute correctly: `ke type="image/svg+xml"`. When inserted in this way, the SVG object is still interactive and the mouse hover-functionality will work.
3. Inline SVG inserted via a User-Defined XML Block. See screenshot below for an example of an SVG code fragment. In this case, interactivity will work. Note that the svg element does not need to be in the SVG namespace if the output method is HTML 4.0 or 5.0, but the namespace is required if the output method is XHTML.

---

See also

- Image URIs
Example: A Template for Images

The StyleVision package contains an SPS file that demonstrates the use of images in StyleVision. This file is in the (My) Documents folder: C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial\Images\Images.sps. The Images document (Images.xml and Images.sps) consists of three parts:

- The first part shows how text state icons can be used in Authentic View (can be created in Enterprise and Professional editions only). When you open the file Images.xml in the Authentic View of XMLSpy, Authentic Desktop, or Authentic Browser, you can try out the use of text state icons. Note that text state icons are not available in the Authentic Preview of StyleVision, so you cannot try out this feature in StyleVision. How to create text state icons is described in the Reference | Authentic Menu | Custom toolbar buttons section of this user manual.

- The second part contains a table showing which image formats are supported in the various StyleVision output formats. In Design View, only images with static URIs will be displayed. All the image formats listed in the table are displayed in Part 3 of the Images document.

- In Part 3, all the popular image formats supported by StyleVision are displayed one below the other. After opening the file Images.sps in StyleVision, you can switch among the various previews of StyleVision to see how each image is displayed in that preview. Since the location of the image is in an XML node, you can also enter the location of your own images in Authentic View and test their appearances in the preview windows.

See also
- Image URIs
- Image Types and Output
- Custom Toolbar Buttons
8.7 Charts

Charts in the SPS provide a graphical representation of data in the source XML document. In an SPS design that uses XSLT 2.0 or 3.0, a chart is set up by specifying a sequence of items for each axis of the chart. StyleVision then automatically generates the chart in the various output formats.

The table below shows the types of charts that can be created in an SPS design and the kind of data items that are required for each of the chart's axes. The dataset to be used for each axis is specified separately with an XPath expression.

<table>
<thead>
<tr>
<th>Chart type</th>
<th>X-Axis (Category)</th>
<th>Y-Axis (Value)</th>
<th>Number of Series (on Z-Axis)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pie Charts (2D, 3D)</td>
<td>Text</td>
<td>Numeric</td>
<td>1</td>
</tr>
<tr>
<td>Bar Charts Ungrouped (2D, 3D)</td>
<td>Text</td>
<td>Numeric</td>
<td>1</td>
</tr>
<tr>
<td>Bar Charts Grouped (2D, 3D)</td>
<td>Text</td>
<td>Numeric</td>
<td>&gt; 1</td>
</tr>
<tr>
<td>Category Line Graphs</td>
<td>Text</td>
<td>Numeric</td>
<td>1 line = 1 series</td>
</tr>
<tr>
<td>Value Line Graphs</td>
<td>Numeric</td>
<td>Numeric</td>
<td>1 line = 1 series</td>
</tr>
<tr>
<td>Area and Stacked Area Charts</td>
<td>Text</td>
<td>Numeric</td>
<td>1 area = 1 series</td>
</tr>
<tr>
<td>Candlestick Charts</td>
<td>Text</td>
<td>Numeric</td>
<td>3 or 4</td>
</tr>
<tr>
<td>Gauge</td>
<td>—</td>
<td>Numeric</td>
<td>1</td>
</tr>
<tr>
<td>Overlay Charts</td>
<td>Text</td>
<td>Numeric</td>
<td>= 1 or &gt; 1 per chart</td>
</tr>
</tbody>
</table>

**Note:** Pie charts and gauge charts have a single nominal series, which requires no name. So if a series name is entered in the data selection, it is ignored. For ungrouped bar charts, however, the name of the single series, if present, is used for the legend. For gauge charts, in addition to any Series Name entry being ignored, the X-Axis data selection is also ignored; only the Y-Axis selection is used for gauge charts.

Inserting a Chart

The XML Schema node within which a chart is created in the design will be the context node of the XPath expressions to select the chart's axes and other chart data. A chart must therefore be inserted within a suitable context node: one from which correct XPath expressions to select the required data can easily be constructed.

To insert a chart, right-click at the required location and select Insert Chart from the menu that pops up (or use the menu command Insert | Insert Chart). Alternatively, select the Insert Chart icon in the toolbar of StyleVision and then click in the design at the location where you wish to insert the chart. These insertion methods pop up the Chart Configuration dialog (screenshot below).
The Chart Configuration dialog has three parts:

- **Single or Multiple layers**: Multiple layers can be selected to create overlay charts, in which multiple charts can be presented, one overlaid upon the other in a sequence of layers.
- **Chart Settings**: To select the chart type and define the appearance of the chart
- **Chart Data Selection**: To select the data for the various axes of the chart using either the simple option or flexible option

On clicking **OK** in the Chart Configuration dialog, the chart is inserted in the design.
**Editing chart settings and data selection**

If you wish to change chart settings or a chart's data selection after a chart has been created, right-click the chart in the design and select **Edit Chart Settings**. This pops up the Chart Configuration dialog (*screenshot above*) of that chart. You can edit the chart's settings or data selection in the dialog, then click **OK** to finish.

- **See also**
  - Chart Settings
  - Chart Data Selection: Simple
  - Chart Data Selection: Flexible
Chart Basics

In this section we present simple examples to illustrate how charts work. These charts are in the SPS file SimpleCharts.sps. The SPS design is based on the XML Schema YearlySales.xsd and it uses YearlySales.xml (screenshot below) as its Working XML File. All three files are located in the (My) Documents folder, C:\Documents and Settings\<username>\My Documents \Altova\StyleVision2015\StyleVisionExamples\Tutorial\Charts.

XML file used in charts example: YearlySales.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="YearlySales.xsd">
  <ChartType>Pie Chart 2D</ChartType>
  <Region id="Americas">
    <Year id="2005">30000</Year>
    <Year id="2006">90000</Year>
    <Year id="2007">120000</Year>
    <Year id="2008">180000</Year>
    <Year id="2009">140000</Year>
    <Year id="2010">100000</Year>
  </Region>
  <Region id="Europe">
    <Year id="2005">50000</Year>
    <Year id="2006">60000</Year>
    <Year id="2007">80000</Year>
    <Year id="2008">100000</Year>
    <Year id="2009">95000</Year>
    <Year id="2010">80000</Year>
  </Region>
  <Region id="Asia">
    <Year id="2005">10000</Year>
    <Year id="2006">25000</Year>
    <Year id="2007">70000</Year>
    <Year id="2008">110000</Year>
    <Year id="2009">125000</Year>
    <Year id="2010">150000</Year>
  </Region>
</Data>
```

After opening the SPS file in StyleVision, check the chart settings to see how data has been selected for the two axes. Do this by right-clicking the charts and selecting the Edit Chart Settings command. This pops up the Chart Configuration dialog, at the bottom of which is the Chart Data Selector pane with fields for entering the XPath expressions for data selection (screenshot below). To view the charts, switch to Authentic Preview.
The four XPath expressions in the Chart Data Selector pane work together and do the following:

- **For-Each XPath expression**: (i) Sets the context for the other three XPath expressions; (ii) Sets the number of items in the returned sequence as the number of ticks on the X-Axis. In the case of the screenshot above, the `Region[1]/Year` expression returns six node items; so there will be six ticks on the X-Axis (see screenshot below).

- **X-Axis XPath expression**: The items in the returned sequence provide the label text for the corresponding ticks on the X-Axis. In the example shown above, the `@id` expression returns the id attribute value of each Year element. These values become the labels of the corresponding ticks (see screenshot below). Since we have specified that this will be a bar chart, bars are drawn at the ticks.

- **Y-Axis XPath expression**: The Y-Axis can display multiple series, each of which is defined in one row of the Y-Axis table. Each series is defined by two XPath expressions: one for the series value, the other for the series name. In our example, the `self::node()` XPath expression (indicated by its abbreviated form of a period) selects the current node, which is the Year element that is the context node. So, for each Year element (represented by a bar on the X-Axis), the Year element's content will be read as the Y-Axis value of that year, and therefore plotted as the height of the bar (see screenshot below). The screenshot further below shows a chart with multiple series on the Y-Axis.

- **Series Name XPath expression**: This expression provides the legend text for the series. In our example, the legend text (which appears at the bottom of the chart, screenshot below) is obtained from an XPath expression that is a text string (see screenshot above).
The screenshot above shows a bar chart with a single series, while that below is of a stacked bar chart with multiple series. In the latter example, the value of each series is stacked on to the bar.
See the chart configuration dialog of the example file SimpleCharts.sps to see how the chart settings were defined. The XPath expressions for this chart are shown in the screenshot below.

```
For each expression (XPath):
Region

Category / X axis:  ○ Auto Enumerated  ○ XPath  @id

Value/Y-Axis(XPath)      Series name(XPath)
Year(1)  2005
Year[5]  2009
Year[6]  2010
```

See also
- Charts
- Chart Settings
- Chart Data Selection: Simple
- Chart Data Selection: Flexible
Chart Settings: Quick Reference

Chart settings select the chart type and define the appearance of the chart. They are made via the Chart Settings pane (screenshot below) of the Chart Configuration dialog. There are two groups of settings: basic (which are made directly in the Chart Settings pane) and advanced (which are accessed via the All Settings and Dynamic XPath Settings buttons).

The Image File Settings button pops up a dialog in which the location and name of the image file can be specified. This location is used when a preview is generated or when the Save Generated Files command is used. Image files created for previews are deleted when StyleVision is closed.

This section contains the following parts:

- Basic chart settings, which are the chart settings available in the Chart Settings pane (see screenshot above) of the Chart Configuration dialog.
- Advanced chart settings, which can be accessed via the All Settings button in the Chart Settings pane (see screenshot above).
- Dynamic XPath settings, which can be accessed via the Dynamic XPath Settings button in the Chart Settings pane (see screenshot above).

Basic chart settings

The basic chart settings are: the chart type, chart title, chart width, and chart height.

To select the chart type, click the Change Type button and, in the Change Type dialog that pops up (screenshot below), select the chart type you want, and click OK.
The chart’s title, width, and height can be entered as static values directly in the Chart Settings pane. If you wish to obtain the chart's title by evaluating an XPath expressions, then the XPath expression for the title must be set in the Dynamic XPath Settings dialog (accessed with the Dynamic XPath Settings button).

The Pixel Density setting specifies the resolution of the chart image in the output. It is set as a percentage of the height and width setting. For example: If a chart is specified to have a height of 250px and a width of 250px, and the pixel density has been set to 200%, then a chart image file is created measuring 500px x 500px. In the output, this image file will be constrained to a width of 250px x 250px, thus producing an image of higher resolution.

**Advanced chart settings**

A chart's advanced settings are accessed by clicking the All Settings button in the Chart Configuration dialog (see screenshot above). This pops up the Change Appearance dialog for that particular chart type (the screenshot below shows the Change Appearance dialog of a pie chart).
The advanced settings are organized into the following common tabs:

- **General**: The chart title (see screenshot below) is the same as the basic setting (see above) and can be edited as an advanced setting also. Other settings in this dialog are the background color of the chart and the plot. In the screenshot below, the plot has been given a pale green background color. An image file can also be set as the background image of the chart and/or the plot. This image can be stretched to cover the entire area of the chart or plot; zoomed to fit so that the zoom matches one of the two dimensions (of chart/plot); centered; or tiled. The legend is the key to the color codes in the chart, and it can be turned on or off.
- **Color scheme:** Four predefined color schemes are available plus a user-defined color scheme. You can modify any of the color schemes by adding and/or deleting colors to a scheme. The color scheme selected in this tab will be used in the chart.
- **Sizes:** Sizes of various aspects of the chart can be set, either as pixels or as a percentage ratio.
- **Font:** The font properties of the chart title and of legends and labels can be specified in this tab. Sizes can be set as a percentage of the chart size or absolutely as points.
- **Load/Save button:** Settings can be saved to an XML file and can be loaded from an XML file having the correct structure. To see the structure, save the settings of a chart and then open the XML file. Clicking this button also gives you the option of resetting chart settings to the default.

Additionally, each type of chart has settings specific to its type. These are listed below:

- **Pie charts:** Settings for: (i) the angle from which the first slice should be drawn; (ii) the direction in which slices should be drawn; (iii) the outline color; (iv) whether the colors receive highlights (in 3D pie charts: whether dropshadows and transparency are used); (v) whether labels should be drawn; and (vi) whether values and percentages should be added to labels and how many decimal places should be added to the percentages.

- **Bar charts:** Settings for: (General) Drawing the X and Y axes exchanged generates a horizontal bar chart; (Bar) Bar outlines and dropshadows (dropshadows in 2D bar charts only); (X-Axis) Label and color of the x-axis, and vertical gridlines; (Y-Axis) Label and color of the y-axis, horizontal gridlines, the range of values to be displayed, and the tick marks on the y-axis; (Z-Axis, 3D only) Label and color of the z-axis; (3D) the vertical tilt, horizontal rotation, and the width of the view.

- **Line graphs:** Settings for: (General) Drawing the X and Y axes exchanged; (Line) including the plot points or not; (X-Axis) Label and color of the x-axis, and vertical gridlines; (Y-Axis) Label and color of the y-axis, horizontal gridlines, the range of values to be displayed, and the tick marks on the y-axis.

- **Gauge:** Settings for: (i) the angle at which the gauge starts and the angular sweep of the scale; (ii) the range of the values displayed; (iii) the interval and color of major and minor ticks; (iv) colors of the dial, the needle, and the border.

- **Area:** The transparency of areas can be set as a value from 0 (no transparency) to 255 (maximum transparency). In the case of non-stacked area charts transparency makes
parts of areas that lie under other areas visible top the viewer. Outlines for the areas can also be specified.

- **Candlestick**: The fill color can be specified for the two situations: (i) when the closing value is greater than the opening value, and (ii) when the opening value is greater than the closing value. In the latter case, the Series color is also available as an option. The Series color is specified in the Color Schema tab of the Change Appearance dialog.

---

### Dynamic XPath Settings

Dynamic XPath Settings are very useful if you wish to use dynamic data from the XML document in the settings of the chart. For example, the title of a chart about a *Region* element might need to have data about that *Region* element (such as its name) in the title of the chart. If there are several Region elements, any one of which will be used at a time for the chart, then the data for the chart title can only be obtained dynamically via an XPath expression. The Dynamic XPath Settings dialog enables such data to be accessed via XPath expressions.

The Dynamic XPath Settings dialog (*screenshot below*) is accessed by clicking the **Dynamic XPath Settings** button in the Chart Settings pane of the **Chart Configuration dialog**. A number of chart settings can be entered in this dialog.

![Dynamic XPath settings for Pie chart](image)

Note the following points:

- All entries in this dialog are evaluated as XPath expressions, so string literals must be enclosed in quotes. For example: "Tahoma", '1', '18pt', and '#3366FF'.
- On hovering over a setting or its value, a tooltip appears giving information about enumerations and formats.
- The Dynamic XPath Settings have precedence over settings made in the Chart Configuration dialog or Change Appearance dialog. For example, a chart title made as a dynamic XPath setting has precedence over one set in the Chart Configuration dialog.
- The colors of the color schema will be used when the User-Defined color schema is
selected in the Change Appearance dialog as the color schema to use. Colors are set in the RGB hexadecimal format: #RRGGBB. So an XPath expression to specify the color red would be: '#FF0000'.

See also
- Chart Settings
- Chart Data Selection: Simple
- Chart Data Selection: Flexible
Chart Settings and Appearance

Chart settings are organized as follows:

- **Basic Chart Settings**, which enable you to select the type of chart and its title. Basic chart settings are defined in the Chart Settings pane of the Chart Configuration dialog (see screenshot below).

- **Advanced Chart Settings**, which enable you to change the appearance of a chart (its title, legend, colors, fonts, etc). Advanced settings are defined in the Change Appearance dialog. To access this dialog, click **All Settings** in the Chart Configuration dialog (see screenshot below).

- **Dynamic XPath Settings**, which can be accessed by clicking **Dynamic XPath Settings** in the Chart Settings pane (see screenshot below).
Basic Chart Settings

This section:

- Setting the chart type
- List of chart types
- Other basic settings

Setting the chart type

The most basic chart setting is the chart type. To select the chart type, click Change Type in the Chart Settings pane of the Chart Configuration dialog.

The various types of charts that are available are listed below. In the Change Type dialog (screenshot above), select the chart type you want and click OK.

- Pie charts
  
  In pie charts, one column/axis provides the values, another column/axis provides labels for
these values. The labeling column/axis can take non-numeric values.

Bar charts can have two sets of values used along two axes (below).

They can also use three sets of values, as in the example below: (i) continent, (ii) year, (iii) sales volume. Bar charts can be displayed in 2D (below) or 3D (above).
A three-axis bar chart can also be stacked if you need to show totals. Compare the stacked chart below with the chart above. The stacked chart shows the total of sales on all continents.

\[ \text{Yearly Sales: Regional and Total} \]

\[ \text{Line charts} \]
The difference between a line chart (below left) and a value line chart (below right) is that value line charts only take numerical values for the X-axis. If you need to display line charts with text values on the X-axis, use line charts.
**Area charts**

Area charts are a variation of line charts, in which the areas below the lines are also colored. Note that area charts can also be stacked (see bar graphs above).
Candlestick charts
A candlestick chart can be used to depict price movements of securities, commodities, currencies, etc over a period of time. The chart indicates not only how prices developed over time, but also the daily close, high, low, and (optionally) open. The Y-axis takes three or four series (close, high, low, and (optionally) open). The screenshot below shows a four-series candlestick chart.

Gauge charts
Gauge charts are used to illustrate a single value and show its relation to a minimum and a
maximum value.

Other basic settings
In the Chart Settings pane, you can also set the title of the chart (see screenshot below).
Additionally:

- The chart's width and height can be entered as static values directly in the Chart Settings pane.
- The *Pixel Density* setting specifies the resolution of the chart image in the output. It is set as a percentage of the height and width setting. For example: If a chart is specified to have a height of 250px and a width of 250px, and the pixel density has been set to 200%, then a chart image file is created measuring 500px x 500px. In the output, this image file will be constrained to dimensions of 250px x 250px, thus producing an image of higher resolution.
- Click the **Image File Settings** button to specify the location and name of an image file for the chart. This image file is used when a preview is generated or when the Save Generated Files command is used. Image files created for previews are deleted when StyleVision is closed.
Advanced Chart Settings

This section:

- Accessing the advanced settings
- Overview of advanced settings
- Loading, saving, resetting chart settings

Accessing the advanced settings
To access a chart's advanced settings do the following: Click All Settings in the Chart Configuration dialog. This displays the Change Appearance dialog for that particular chart type (the screenshot below shows the Change Appearance dialog of a pie chart).

Overview of advanced settings
The advanced settings are organized into tabs that are common to all chart types and those that are specific to a single chart type.

Common chart settings
- General
The chart title (see screenshot below) is the same as the basic setting (see above) and can be edited as an advanced setting also. Other settings in this dialog are the background color of the chart and the plot. In the screenshot below, the plot has been given a pale green background color. An image file can also be set as the background image of the chart and/or the plot. This image can be stretched to cover the entire area of the chart or plot; zoomed to fit so that the zoom matches one of the two dimensions (of chart/plot); centered; or tiled. The legend is the key to the color codes in the chart, and it can be turned on or off.

![Pie Chart Example](image)

**Color scheme**

Four predefined color schemes are available plus a user-defined color scheme. You can modify any of the color schemes by adding colors to and/or deleting colors from a scheme. The color scheme selected in this tab will be used in the chart.

**Sizes**

Sizes of various aspects of the chart can be set, either as pixels or as a percentage ratio.

**Font**

The font properties of the chart title and of legends and labels can be specified in this tab. Sizes can be set as a percentage of the chart size or absolutely as points.

**Load/Save button**

Settings can be saved to an XML file and can be loaded from an XML file having the correct structure. To see the structure, save the settings of a chart and then open the XML file. Clicking this button also gives you the option of resetting chart settings to the default.
**Type-specific chart settings**

**Pie charts**
Settings for: (i) the angle from which the first slice should be drawn; (ii) the direction in which slices should be drawn; (iii) the outline color; (iv) whether the colors receive highlights (in 3D pie charts: whether dropshadows and transparency are used); (v) whether labels should be drawn; and (vi) whether values and percentages should be added to labels and how many decimal places should be added to the percentages.

**Bar charts**
Settings for: (General) Drawing the X and Y axes exchanged generates a horizontal bar chart; (Bar) Bar outlines and dropshadows (dropshadows in 2D bar charts only); (X-Axis) Label and color of the x-axis, and vertical gridlines; (Y-Axis) Label and color of the y-axis, horizontal gridlines, the range of values to be displayed, and the tick marks on the y-axis; (Z-Axis, 3D only) Label and color of the z-axis; (3D) the vertical tilt, horizontal rotation, and the width of the view.

**Line graphs**
Settings for: (General) Drawing the X and Y axes exchanged; (Line) including the plot points or not; (X-Axis) Label and color of the x-axis, and vertical gridlines; (Y-Axis) Label and color of the y-axis, horizontal gridlines, the range of values to be displayed, and the tick marks on the y-axis.

**Gauge charts**
Settings for: (i) the angle at which the gauge starts and the angular sweep of the scale; (ii) the range of the values displayed; (iii) the interval and color of major and minor ticks; (iv) colors of the dial, the needle, and the border.

**Area charts**
The transparency of areas can be set as a value from 0 (no transparency) to 255 (maximum transparency). In the case of non-stacked area charts transparency makes parts of areas that lie under other areas visible to the viewer. Outlines for the areas can also be specified.

**Candlestick charts**
The fill color can be specified for the two situations: (i) when the closing value is greater than the opening value, and (ii) when the opening value is greater than the closing value. In the latter case, the Series color is also available as an option. The Series color is specified in the Color Schema tab of the Change Appearance dialog.
Loading, saving, resetting chart settings
Chart settings that are different from the default settings can be saved in an XML file. These settings can subsequently be loaded as the settings of a chart, which can help you save time and effort. The **Load/Save** button (see first screenshot in this section) provides the following options when clicked:

- **Set to default**: Rejects changes made to the settings, and restores the default settings to all settings sections.
- **Load from file**: Enables settings to be imported that have been previously saved in an XML file (see next command). The command displays the **Open** dialog, in which you enter the location of the required file.
- **Save to file**: Opens the **Save As** dialog box. You can specify an XML file in which to save the settings. This file lists those settings that are different from the default settings.

General
The General section of the **Change Appearance** dialog box lets you define the title of the chart, add or remove a legend, and define background pictures and colors and—for bar, line, area, and candlestick charts—orientation of the chart.

---

**Chart**
Enter a descriptive title for your chart into the **Chart Title** field and select a background color for the entire chart from the drop-down list. You can choose a solid background, vertical gradient, or horizontal gradient and define start and end colors for the gradient, if applicable. In addition, or instead of a colored background, you can also define a background image and choose one of the available display options from the drop-down list:
- **Stretched**: the image will be stretched to the height and width of the chart.
• Zoom to Fit: the image will be fit into the frame of the chart and the aspect ratio of the image will be maintained
• Center: the image will be displayed in its original size in the center of the chart
• Tiled: if the image is smaller than the chart, duplicates of the image will be displayed to fill the background area

The Draw Legend check box is activated by default, clear the check box if you do not want to display a legend in your chart.

---

**Plot**

The Plot is the area where the actual data of the chart is displayed. You can draw a border around the plot and specify a different background color and/or image for the plot area. In the screenshot below, the background color of the chart has been changed to gray (vertical gradient) whereas the plot is still white, a red border has been drawn around it, and a background image has been added.

---

**Orientation**

If you have a small series of large values it may be convenient to swap the X and Y axis for a better illustration. Note that in the screenshot below also the background color of the plot has been set to “Transparent” and the background image has been applied to the chart.
Note that this option is not available for pie and gauge charts.

Type-Related Features

For each of the chart types, and even for the various sub-types, the **Change Appearance** dialog box provides a section where you can define the type-related features of the chart.

### Pie chart

Most settings are the same for the 2d and 3d versions. In 2d pie charts, you can additionally draw highlights.

- **Start Angle:** 0°
- **Labels**
  - Show Labels
  - Add Value to Labels
  - Add Percent to Labels
  - Decimal Digits: 0
- **Draw Outline**
- **Clockwise**
- **Draw Highlights**

In 3d pie charts, you can display drop shadows, add transparency and define the 3d tilt.
The Start Angle value defines where the first row of the selected column will be displayed in the chart. An angle of 0 degrees corresponds to 12 o'clock on a watch.

You can show labels in addition to, or instead of, the legend, add values and/or percentage to the labels, and define for the percentage values the number of decimal digits to be displayed.

The color that you can select next to the Draw Outline check box is used for the optional border drawn around the chart and the individual pie segments. The Clockwise check box allows you to specify whether the rows should be listed clockwise or counter-clockwise.

In 3d pie charts, you can draw a drop shadow and define its color, add transparency to the chart, and define the 3d tilt. In 2d charts, the Draw Highlights option adds additional structure to the chart.

---

**Bar chart**

For bar charts, you can add an outline to the bars and define its color. In 2d bar charts, you can also draw a drop shadow and define its color (this option is not available for 3d bar charts). By default, the shape of the bars resembles a cylinder, however you can also choose "Vertical Gradient" or "Solid" from the Fill style drop-down list.

---

**Line chart**
To draw connection shapes that mark the values in line charts, you need to activate at least one check box in the Draw Connection Shapes group box. You can use five different shapes to mark a series: square, rhomb, triangle, inverted triangle, and circle. If there are more than five series in your chart you can combine the connection shapes by selecting more than one option in the Draw Connection Shapes group box. In the screenshot below, both Filled and Slashed have been selected and the Slashed type is used for the sixth series and beyond.

Connection shapes are available for both line charts and value line charts.

### Area chart

Among the properties that you can change for area charts is transparency; this way you can prevent that one series is hidden by another series in the chart. In addition, you can add an outline to the individual data areas and define its color (see screenshot below).

### Candlestick chart
If both opening and closing value are defined as series, you can choose the colors and whether or not the candle should be filled if the closing value is greater that the opening value.

---

**Gauge chart**

<table>
<thead>
<tr>
<th>Angles</th>
<th>Start: 225</th>
<th>Sweep: 270</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value Range</td>
<td>Start: 0</td>
<td>End: 100</td>
</tr>
<tr>
<td>Major Ticks</td>
<td>Interval: 10</td>
<td>Color:</td>
</tr>
<tr>
<td>Minor Ticks</td>
<td>Interval: 5</td>
<td>Color:</td>
</tr>
<tr>
<td>Colors</td>
<td>Dial fill:</td>
<td>Border:</td>
</tr>
<tr>
<td></td>
<td>Needle:</td>
<td>Needle Base:</td>
</tr>
</tbody>
</table>

The **Start** value in the Angles group box defines the position of the 0 mark and the **Sweep** value is the angle that is used for display. In the Value Range group box you can define the minimum and maximum values to be displayed. Tick marks are displayed with (major ticks) or without (minor ticks) the corresponding value; you can define separate colors for them. In the Colors group box you can define colors for the dial fill, needle, needle base ( hides the first part of the needle in the center of the chart), and the border that surrounds the chart.

**Colors**

Depending on the chart type you have selected, StyleVision provides two different sections for the definition of colors to be used in charts:

- Color Schema for pie, bar, line, area, and candlestick charts
- Color Range for gauge charts
Color Schema

The Color Schema section of the Change Appearance dialog box provides four predefined color schemas (i.e., default, grayscale, colorful, and pastel) that can be customized; in addition you can also define your own color schema from scratch.

The top color will be used for the first series, then the second color and so on. You can change the order or the colors by selecting a color and dragging it to its new positions with the mouse. To add a new or delete an unwanted color, click the corresponding button. In candlestick charts, only the first color will be used.

If you have appended one or several layers of overlay charts to a Charts window, the Color Schema section of the Change Appearance dialog box contains the additional radio button Use subsequent colors from previous chart layer which is activated by default.

When the radio button is activated, the color schema from the previous layer will be used and you cannot choose a separate color schema for the overlay. The series of the active layer will be displayed using subsequent colors from the color schema of the previous layer. This way, all series of the Charts window have different colors and can therefore be distinguished more easily.

You can break this link on any additional layer that you add and choose a different color schema that then can also be re-used in subsequent layers.
**Color Range**

In gauge charts, you can customize the appearance of the gauge by applying colors to certain value ranges.

The definition shown in the screenshot above will appear in the gauge charts as follows:

X-Axis

In the X-axis section of the Change Appearance dialog box, you can enter a label for the axis, and define colors for the axis and the grid lines (if displayed). You can also define whether or not you want to display tick marks and axis values. This section is the same for all bar, line, area, and candlestick charts.

In Value Line Charts however, you can also define the value range, and define at what interval tick marks should be displayed.
**Label**
The text entered into the Label field will be printed below the axis as a description of the X-axis.

**Range**
By default, the Auto radio button is selected in the Range group box. If you want to display a fragment of the chart in greater detail, activate the Manual radio button and enter minimum and maximum values into the respective fields. If the column that is used for the X-axis does not include zero, you can deactivate the Include zero check box and the X-axis will start with the minimum value that is available in the series.

**Line**
The axis is displayed in the color that you choose from the Line drop-down-list. You can use one of the preselected colors, or click the Other color... button to choose a standard color or define a custom color. Click the Select... button on the Custom tab and use the pipette to pick a color that is displayed somewhere on your screen.

**Grid lines**
If the Show Grid lines check box is activated, you can choose a color from the corresponding drop-down list box.

**Tick Interval**
If you are not satisfied with the default tick marks, you can activate the Manual radio button in the Tick Interval group box and enter the difference between the individual tick marks into the corresponding field.
**Tick Drawing**
You can switch the display of tick marks on the axis and/or axis values on or off.

**Axis Position**
From the drop-down list, you can choose the position where the axis is to be displayed. When selecting "At Value / On Category Number", you can also position the axis anywhere within the plot.

**Y-Axis**
In the Y-axis section of the Change Appearance dialog box, you can enter a label for the axis, define colors for the axis and the grid lines (if displayed), define the value range, and decide if and where tick marks should be displayed and whether or not you want to show the axis values. This section is the same for all bar and line charts.

- **Label**
  The text entered into the **Label** field will be printed to the left of the axis as a description of the Y-axis.

- **Range**
  By default, the **Auto** radio button is selected in the Range group box. If you want to display a fragment of the chart in greater detail, activate the **Manual** radio button and enter minimum and maximum values into the respective fields.
  If the column that is used for the X-axis does not include zero, you can deactivate the **Include zero** check box and the X-axis will start with the minimum value that is available in the series.
**Line**
The axis is displayed in the color that you choose from the Line drop-down-list. You can use one of the preselected colors, or click the Other color... button to choose a standard color or define a custom color. Click the Select... button on the Custom tab and use the pipette to pick a color that is displayed somewhere on your screen.

**Grid lines**
If the Show Grid lines check box is activated, you can choose a color from the corresponding drop-down list box.

**Tick Interval**
If you are not satisfied with the default tick marks, you can activate the Manual radio button in the Tick Interval group box and enter the difference between the individual tick marks into the corresponding field.

**Tick Drawing**
You can switch the display of tick marks on the axis and/or axis values on or off.

**Axis Position**
From the drop-down list, you can choose the position where the axis is to be displayed. When selecting "At Value / On Category Number", you can also position the axis anywhere within the plot.

**Z-Axis**
In the Z-axis section of the Change Appearance dialog box, you can enter a label for the axis, define colors for the axis, and decide whether or not you want to show tick marks on the axis. This section is the same for all 3d bar charts (Bar Chart 3d and Bar Chart 3d Grouped).

**Label**
The text entered into the Label field will be printed to the right of the axis as a description of the Z-axis.

**Line**
The axis is displayed in the color that you choose from the Line drop-down-list. You can use one of the preselected colors, or click the Other color... button to choose a standard color or define a custom color. Click the Select... button on the Custom tab and use the pipette to pick a color that is displayed somewhere on your screen.

**Tick Drawing**
You can switch the display of tick marks on the axis on or off.
3D Angles

In 3d bar charts you can customize the 3d appearance of the chart in the 3d Angles section of the **Change Appearance** dialog box.

![Camera settings](image)

The **Tilt** value determines the rotation around the X-axis, whereas the **Rotation** value defines the rotation around the Y-axis. You can automatically adapt the size of the chart axis to the Chart window width by activating the corresponding check box.

If the **Automatic Chart Axis Size** check box is activated, StyleVision will automatically calculate the optimum size of the X-axis as well as the Y-axis for the current Chart window size. The width and height of the chart will change dynamically when you resize the Chart window.

Sizes

In the Sizes section of the **Change Appearance** dialog box, you can define different margins as well as the size of axis and gauge ticks. Note that not all the properties listed below are available for all chart types.

**General**
- **Outside margin**: Space between the plot and the edge of the Chart window.
- **Title to Plot**: Space between the chart title and the upper edge of the plot.
- **Legend to Plot**: Space between the lower edge of the plot and the legend.

**Pie**
- **Plot to Label**: In pie charts, the space between the most left and right edge of the pie and its labels.
- **Pie Height**: In 3d pie charts, the height of the pie.
- **Pie Drop Shadow**: In 3d pie charts, the length of the shadow (if it is activated in the Pie section).

**X-Axis**
- **X-Axis to Axis Label**: In bar and line charts, the space between the X-axis and its label.
- **X-Axis to Plot**: In 2d bar charts and line charts, the space between the X-axis and the plot.
- **X-Axis Tick Size**: In bar and line charts, the length of the ticks on the X-axis.

**Y-Axis**
- **Y-Axis to Axis Label**: In bar and line charts, the space between the Y-axis and its label.
- **Y-Axis to Plot**: In 2d bar and line charts, the space between the Y-axis and the plot.
- **Y-Axis Tick Size**: In bar and line charts, the length of the ticks on the Y-axis.

**Z-Axis**
- **Z-Axis to Axis Label**: In 3d bar charts, the space between the Z-axis and its label.
- **Z-Axis Tick Size**: In 3d bar charts, the length of the ticks on the Z-axis.

**Line Drawing**
- **Connection Shape**: In line charts, the size of the squares that mark the values in the chart.
Size

3d Axis Sizes
- Manual X-Axis Size of Base: In 3d bar charts, defines the relation between the length of the X-axis and the Chart window size. Please note that the Automatic Chart Axis Size check box in the 3d Angles section must be deactivated, otherwise the size will still be calculated automatically.
- Manual Y-Axis Size of Base: In 3d bar charts, defines the relation between the length of the Y-axis and the Chart window size. Please note that the Automatic Chart Axis Size check box in the 3d section must be deactivated, otherwise the size will still be calculated automatically.
- Z-Axis Series Margin: In 3d bar charts, the distance on the Z-axis between the individual series.

Gauge
- Border Width: In round gauge charts, the width of the border around the gauge.
- Border to Tick Distance
- Major Tick Length: In round gauge charts, the length of the major ticks (i.e., ticks that show a label).
- Major Tick Width: In round gauge charts, the width of the major ticks (i.e., ticks that show a label).
- Minor Tick Length: In round gauge charts, the length of ticks that do not have a value displayed.
- Minor Tick Width: In round gauge charts, the width of ticks that do not have a value displayed.

Gauge Needle
- Needle Length: In round gauge charts, the length of the needle. (Note that the percentage is calculated from the diameter of the gauge, so if you choose a value greater than 50%, the needle will point to somewhere outside the gauge!)
- Needle Width at Base
- Needle Base Radius: In round gauge charts, the radius of the base that covers the center of the gauge.

Gauge Color Range
- Border to Color Range Distance
- Color Range Width: In round gauge charts, the width of the customizable color range. (Note that the percentage is calculated from the diameter of the gauge!)

Fonts
The Fonts section of the Change Appearance dialog box lets you configure fonts for objects in the Chart window.
Font settings
You can choose the font face, size, and style for the individual elements displayed in the Chart window. You can define the size as a percentage of the chart size and define a minimum size in points, or specify an absolute value (in points). To apply the same font and/or size to all text elements, activate the respective Use the same for all check box. The element names in the list box are defined as follows:

- **Title**: The name of a chart
- **Legend**: The key to the colors used in the chart
- **Labels**: The designation of the pies of a pie chart
- **Axis Title**: The name of the X, Y, and Z axis in a bar or line chart
- **Axis Values**: The units displayed on an axis in a bar or line chart
- **Tick Values**: The units displayed on a gauge chart
Dynamic XPath Settings

Dynamic XPath Settings are very useful if you wish to use dynamic data from the XML document in the settings of the chart. For example, the title of a chart about a Region element might need to have data about that Region element (such as its name) in the title of the chart. If there are several Region elements, any one of which will be used at a time for the chart, then the data for the chart title can only be obtained dynamically via an XPath expression. The Dynamic XPath Settings dialog enables such data to be accessed via XPath expressions.

To access the chart's Dynamic XPath Settings dialog, click Dynamic XPath Settings in the Chart Settings pane of the Chart Configuration dialog (see screenshot below).

Dynamic XPath Settings

The Dynamic XPath Settings dialog (screenshot below) is accessed by clicking the Dynamic XPath Settings button in the Chart Settings pane of the Chart Configuration dialog. A number of chart settings can be entered in this dialog.

Note the following points:
All entries in this dialog are evaluated as XPath expressions, so string literals must be enclosed in quotes. For example: 'Tahoma', '1', '18pt', and '#FF3366'.

On hovering over a setting or its value, a tooltip appears giving information about enumerations and formats.

Dynamic XPath settings have precedence over settings made in the Chart Configuration dialog or Change Appearance dialog. For example, a chart title made as a dynamic XPath setting has precedence over one set in the Chart Configuration dialog.

The colors of the color schema will be used when the user-defined color schema is selected in the Change Appearance dialog as the color schema to use. Colors are set in the RGB hexadecimal format: #RRGGBB. So an XPath expression to specify the color red would be: '#FF0000'.

See also

- Chart Settings and Appearance
- Chart Data Selection: Simple
- Chart Data Selection: Flexible
Chart Data Selection: Simple

In the Chart Data Selector pane of the Chart Configuration dialog, the Simple option enables the data selection to be visualized as a table. We will use the XML document below to explain the visualization.

XML file used in charts example: YearlySales.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="YearlySales.xsd">
    <ChartType>Pie Chart 2D</ChartType>
    <Region id="Americas">
        <Year id="2005">30000</Year>
        <Year id="2006">90000</Year>
        <Year id="2007">120000</Year>
        <Year id="2008">180000</Year>
        <Year id="2009">140000</Year>
        <Year id="2010">100000</Year>
    </Region>
    <Region id="Europe">
        <Year id="2005">50000</Year>
        <Year id="2006">60000</Year>
        <Year id="2007">80000</Year>
        <Year id="2008">100000</Year>
        <Year id="2009">95000</Year>
        <Year id="2010">80000</Year>
    </Region>
    <Region id="Asia">
        <Year id="2005">10000</Year>
        <Year id="2006">25000</Year>
        <Year id="2007">70000</Year>
        <Year id="2008">110000</Year>
        <Year id="2009">125000</Year>
        <Year id="2010">150000</Year>
    </Region>
</Data>
```

In the SPS design we insert a chart inside the Data node. In the Chart Data Selector pane (screenshot below) we make the data selection as shown in the screenshot. Since the chart has been inserted within the Data node, the context node for the For-Each expression is the Data node.
The chart data table can be visualized as the table below. What happens is that for each Region[1]/Year element a row is created and the X-Axis and Y-Axis XPath expressions are evaluated within the respective Region[1]/Year element's context.

<table>
<thead>
<tr>
<th>For-Each XPath</th>
<th>X-Axis</th>
<th>Y-Axis for Series</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>@id</td>
<td>Americas</td>
</tr>
<tr>
<td>Region[1]/Year</td>
<td></td>
<td>Europe</td>
</tr>
<tr>
<td>Region[1]/Year</td>
<td></td>
<td>Asia</td>
</tr>
<tr>
<td>Region[1]/Year</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Region[1]/Year</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Region[1]/Year</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Region[1]/Year</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- The For-Each expression Region[1]/Year returns six nodes (which become the rows of the table). The number of items in the sequence returned by the For-Each expression determines the number of ticks on the X-Axis.
- The XPath expression for the X-Axis returns the @id attribute value of each Region[1]/Year element. These values will be the labels of the X-Axis ticks. If there are more labels than ticks then extra ticks will be generated so that all labels are plotted. If there are fewer labels than ticks, then the latter ticks (for which no corresponding labels exist) will be unlabelled. The Auto-Enumerated option generates a sequence of integers starting with 1, and assigns each integer sequentially to an X-Axis tick.
- The XPath expression for the Americas series (text()) returns the content of each of the Region[1]/Year elements. This expression could also have been one similar to that for the Europe and Asia series (explained below)—as long as it efficiently returns the values we want.
- The XPath expression for the Europe series is: for $i in @id return //Region[2]/
Year[@id={$i}]. This expression does the following: (i) looks for the current Region[1]/Year/@id attribute value, (ii) returns the content of the Region[2]/Year element that has the same @id value as the @id value of the current Region[1]/Year element.

- The XPath expression for the Asia series works in a similar way to the XPath expression for the Europe series.

The bar chart generated with this data selection would look something like this:

![Yearly Sales: Regional Comparison](chart1)

And the line graph chart for this data selection would look like this:

![Chart 4: Simple Selection B](chart2)

The data selection described above is that for one of the line graphs contained in the example SPS file YearlySales.sps. The SPS design is based on the XML Schema YearlySales.xsd and it uses YearlySales.xml (XML file screenshot above) as its Working XML File. All three files
are located in the (My) Documents folder, C:s|\Documents and Settings\<username>\My
Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial\Charts.

If the For-Each expression returns items that are not nodes
Since the number of X-Axis ticks is primarily dependent on the number of items returned by the
For-Each XPath expression, the XPath expression in the screenshot below (distinct-values(//
Year/@id)), which returns the six unique year values, will also generate six ticks on the X-Axis.
The items returned by the sequence, however, are atomic values, not nodes. Consequently,
although they can be used as context items, they cannot be used as context nodes for locating
nodes in the XML tree. They can, however, be used to locate nodes on the basis of the equality of
values—which is how we will use them.

![Chart Data Selector]

In the data selection shown in the screenshot above, note the following:
- The X-Axis and Y-Axis data selections use the atomic values returned by the For-Each
eexpression, respectively, as direct output and as filter test values.
- Location steps in XPath expressions start at the document node (the $XML in $XML//
Region...). This is necessary because the atomic values provide no locational context.

The chart data table would evaluate to the following:

<table>
<thead>
<tr>
<th>For-Each XPath</th>
<th>X-Axis</th>
<th>Y-Axis for Series</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>2005</td>
<td>XPath-1</td>
</tr>
<tr>
<td>2006</td>
<td>2006</td>
<td>XPath-1</td>
</tr>
</tbody>
</table>

Americas  | Europe | Asia
---       | ---    | ---
XPath-1   | XPath-2| XPath-3
<table>
<thead>
<tr>
<th>Year</th>
<th>Year</th>
<th>XPath-1</th>
<th>XPath-2</th>
<th>XPath-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>2007</td>
<td>XPath-1</td>
<td>XPath-2</td>
<td>XPath-3</td>
</tr>
<tr>
<td>2008</td>
<td>2008</td>
<td>XPath-1</td>
<td>XPath-2</td>
<td>XPath-3</td>
</tr>
<tr>
<td>2009</td>
<td>2009</td>
<td>XPath-1</td>
<td>XPath-2</td>
<td>XPath-3</td>
</tr>
<tr>
<td>2010</td>
<td>2010</td>
<td>XPath-1</td>
<td>XPath-2</td>
<td>XPath-3</td>
</tr>
</tbody>
</table>

See also
- [Charts](#)
- [Chart Settings](#)
- [Chart Data Selection: Flexible](#)
Chart Data Selection: Flexible

In the Chart Data Selector pane (screenshot below) of the Chart Configuration dialog, the Flexible option enables the Series Axis (Z-Axis), X-Axis, and Y-Axis data to be selected freely using XPath expressions. The XPath expression for an axis returns the sequence of items that are to be plotted on that axis. These sequences (of items) for the axes are then collated to generate the chart.

Note the following points:

- A series refers to a series of values plotted for a set of X-Axis (Category Axis) ticks. A second series would plot a second set of values on the same X-Axis ticks. For example, if the X-Axis represented the years 2008, 2009, and 2010, and the Y-Axis represented sales turnover, then Series 1 could represent America (sales in America for those three years) while Series 2 could represent Europe (sales in Europe for those three years). If this data were selected for a bar chart, then for each year (2008, 2009, 2010) on the X-Axis, there would be two bars (America and Europe), one for each series. In the case of pie charts and single-bar charts, only one series is possible. See the chart type table for more information about each chart type.
- Each row in the Chart Data Selector pane represents a series.
- The XPath expression in the For-Each column provides the context for the evaluation of each of the other three XPath expressions. The For-Each XPath expression is itself evaluated in the context of the node in the design within which it was inserted.

The following example illustrates important points to consider when selecting data for the axes. We will use the XML document below.

XML file used in charts example: YearlySales.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="YearlySales.xsd">
  <ChartType>Pie Chart 2D</ChartType>
</Data>
```
One row, one series

Say we wish to generate a 2D bar chart for each Region element (there are three such elements: for the Americas, Europe, and Asia). Let us create the chart in the design within the Region element node. The context node for the For-Each XPath expressions in the Chart Data Selector will therefore be the Region element.
In the chart data selection shown in the screenshot above, the For-Each expression returns the current node (which is the Region element), so the Region element will be the context node for all the other three XPath expressions (series, X-Axis, and Y-Axis). Since there is only one series in this chart, we do not need a series name and so we leave this column blank. The X-Axis selection returns six values. Six will therefore be the number of ticks on the X-Axis and the six items of the sequence will be the respective labels of the X-Axis ticks. The Y-Axis selection also returns six items, each of which is plotted on the Y-Axis for its corresponding X-Axis tick. Since the chart was created within the Region element, the XSLT engine will iterate over the three Region elements to produce three charts, one for each Region element. For each chart, that particular Region element’s descendant nodes will be used.

The chart for the Americas region would look something like this in the output:
Three rows, three series, category values not merged

To create multiple series, additional rows can be added to the chart data selection, as shown in the screenshot below.

The important points to note about the data selection above are:

- Each row defines a series and all rows have the Region element as its context node (since the chart is created inside a Region node).
- The first row is set to define the Americas series and is given a string expression as its series name. The X-Axis values are selected using the Year/@id values of the Europe
region (it doesn't matter which region is selected since all have the same Year/@id values). The Y-Axis values of the first (Americas) series are selected for the Americas region using a predicate filter.

- The second and third series follow the same pattern as the first series. Note, however, that the X-Axis selection for each series is identical. But since the Merge Category Values check box is not checked, the second and third expressions are ignored. (Even if the values were merged, it would not make a difference because the values of each series are identical; only new distinct values would be added to the category values.)

The chart generated with the data selection above would look something like this:

![Yearly Sales: Regional Comparison](image)

Three rows, three series, category values merged
The data selection in this example (see screenshot below) is different from the previous example in two respects: (i) the X-Axis selection for the third series has an extra item (2011) added to the series, and (ii) the Merge Category Values check box has been checked, and (iii) the Y-Axis tick interval has been manually set to 20000.
The effect of this change is to add one new item (2011) to the X-Axis result sequence. The chart would look something like this:

![Yearly Sales: Regional Comparison](image)

**One row, three series**

The chart in this example is created, not within the Region node as in the previous examples, but one node higher, in the Data node (see XML document above). Only one row is used for data selection, but it generates three series. This is because the XPath expression in the For-Each column returns a sequence of three items, thus implicitly creating three series.
For each series, the series name, X-Axis, and Y-Axis selections will correspond to the different regions because each series has a different `Region` element as its context node. The chart for this data selection will look something like this:

**Yearly Sales: Regional Comparison**

![Yearly Sales Chart]

### Rules for chart data selection
The following points should be noted when using the Chart Data Selector to select data for the various chart axes:

1. The number of bars (or pie chart slices, etc) is equal to the number of items in the larger of the X-Axis or Y-Axis sequences of a single data row selection. So if the X-Axis (which
gives labels) has five items and the Y-Axis (which gives values) has six items, then six bars will be plotted with the last one being unlabelled. If the X-Axis has six items and the Y-Axis five items, then six bars will be plotted with the last one being labeled but having a value of zero.

2. The number of series is equal to the cumulative number of items in all the sequences returned by expressions in the For-Each column.

3. The name of a series is selected with the XPath expression of the Z-Axis (or Series Name Axis). If, in a data selection row, this XPath expression is left empty, then a no-name series is created. Also if the XPath expression returns a sequence with a lesser number of items than the number of series, then some series will have no name.

For more examples of how data is selected with the Flexible option, see the examples in the following sections. These examples include:

- The use of editable Authentic variables for data selection, also in Gauges
- XPath expressions for complex data selection
- Using the Series Axis to generate grouped bars
- Using the Series Axis to generate line graphs
- Using literal sequences from an XPath expression as values
- Using Java and .NET functions

See also
- Charts
- Chart Settings
- Chart Data Selection: Simple
Example: Pie Charts

In this section, we discuss the 2D and 3D pie charts contained in the SPS file `YearlySales.sps`. The SPS design is based on the XML Schema `YearlySales.xsd` and it uses `YearlySales.xml` (screenshot below) as its Working XML File. All three files are located in the (My) Documents folder, `C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial\Charts`.

XML file used in charts example: `YearlySales.xml`

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="YearlySales.xsd">
  <ChartType>Pie Chart 2D</ChartType>
  <Region id="Americas">
    <Year id="2005">30000</Year>
    <Year id="2006">90000</Year>
    <Year id="2007">120000</Year>
    <Year id="2008">180000</Year>
    <Year id="2009">140000</Year>
    <Year id="2010">100000</Year>
  </Region>
  <Region id="Europe">
    <Year id="2005">50000</Year>
    <Year id="2006">60000</Year>
    <Year id="2007">80000</Year>
    <Year id="2008">100000</Year>
    <Year id="2009">95000</Year>
    <Year id="2010">80000</Year>
  </Region>
  <Region id="Asia">
    <Year id="2005">10000</Year>
    <Year id="2006">25000</Year>
    <Year id="2007">70000</Year>
    <Year id="2008">110000</Year>
    <Year id="2009">125000</Year>
    <Year id="2010">150000</Year>
  </Region>
</Data>
```

After opening the SPS file in StyleVision, switch to Authentic Preview, and, in the Chart Type combo box (screenshot below), select Pie Chart 2D and Pie Chart 3D to see the two kinds of chart.
For each chart type (2D pie charts and 3D pie charts), there are four charts. The first three charts of each chart type show the yearly sales of the three sales regions. Each of these pie charts (see screenshot of a 3D pie chart below) represents a single region, and the slices of the pie chart represent the region's sales per year for the six years from 2005 to 2010.

To view the chart settings of these charts, switch to Design View, select the appropriate branch...
(Pie Chart 2D or Pie Chart 3D) in the first condition of the design, right-click the chart, and select Edit Chart Settings. This pops up the Chart Configuration dialog.

Note the following points about the Regional Yearly Sales charts:

1. The charts have been inserted within the Region element, so the context node is the Region element. A chart is therefore created for each iteration of the XSLT Engine over the Region element.
2. The title of each chart contains the value of the Region/@id attribute. Click the Dynamic XPath Settings button to see how this is done.
3. Since no more than one series is required, we can leave the Z-Axis or Series Name Axis empty. This results in a default no-name series being created.
4. The X-Axis selects the labels to be used for the slices of the pie chart, in this case the years 2005 to 2010, which are obtained with the XPath expression Year/@id (see screenshot below).

5. The Y-Axis selects the values that appear for each of the slices of the pie chart. In our example, these are the contents of the Year element children of the Region element.
6. The number of slices in the pie chart is determined according to the Chart Data Selector Rules. So, if the selection were made as in the screenshot below, then another slice for the year 2011 would be added with a value of 60000.

7. The items of the sequence selected for the X-Axis will be assigned, in corresponding order, as the label for each slice of the pie chart. Each X-Axis item also generates a legend item.
8. The number of X-Axis and Y-Axis items in each series should be the same (see screenshot above), though it is not an error if they are not the same. If there are fewer X-Axis items than Y-Axis items, then the latter slices of the pie chart (which do not have a corresponding X-Axis item) will be unlabelled. If there are more X-Axis items than Y-Axis items, then the extra X-Axis items will be ignored for label-generation, but they will be generated as legend items.
Using variables in XPath expressions that select axes data

The 2D Pie Chart example has a fourth pie chart at the bottom of the page: the Yearly Market Share pie chart (screenshot below).

In Authentic Preview, when the year selection is changed in the Select Year combo box, the pie chart changes to show the correct values for the selected year. This is achieved by (i) storing the user selection in an editable variable for Authentic ($SelectYear), and (ii) using this editable variable in an XPath predicate to determine which years are selected for the Y-Axis: `for $i in //Year[@id=$SelectYear] return $i`. In Design View, open the Chart Configuration dialog to see the data selection settings. Also, the labels of the X-Axis contain the percentage share of the pie of that X-Axis tick; the percentage can be toggled on and off in the Pie settings of the advanced settings of the chart. In these settings you can also specify whether the Y-Axis value should be shown in the labels or not.
Using XPath capabilities for complex data selection

The important thing to bear in mind is that the data selection for each axis (X and Y) must return the sequence of required atomic items. The last 3D pie chart shows the yearly worldwide sales. The XPath expressions for the data selection are as follows:

X-Axis: \( \text{for } \$i \text{ in distinct-values}(/\text{Year}@id) \text{ return } \$i \)  
The selection could as easily have been: 2005, 2006, 2007, 2008, 2009, 2010

Y-Axis: \( \text{for } \$i \text{ in distinct-values}(/\text{Year}@id) \text{ return sum}(/\text{Year}@id=\$i) \)  
The selection returns, for each year, the sum of worldwide sales for that year, that is, for a given year, the sum of sales in all three regions.

See also
- Charts
- Chart Settings
- Chart Data Selection: Simple
- Chart Data Selection: Flexible
Example: Bar Charts

In this section, we discuss the 2D and 3D bar charts (including grouped bar charts and stacked bar charts) contained in the SPS file YearlySales.sps. The SPS design is based on the XML Schema YearlySales.xsd and it uses YearlySales.xml (screenshot below) as its Working XML File. All three files are located in the (My) Documents folder, C:\Documents and Settings \<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples/Tutorial/Charts.

XML file used in charts example: YearlySales.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="YearlySales.xsd">
  <ChartType>Pie Chart 2D</ChartType>
  <Region id="Americas">
    <Year id="2005">30000</Year>
    <Year id="2006">90000</Year>
    <Year id="2007">120000</Year>
    <Year id="2008">180000</Year>
    <Year id="2009">140000</Year>
    <Year id="2010">100000</Year>
  </Region>
  <Region id="Europe">
    <Year id="2005">50000</Year>
    <Year id="2006">60000</Year>
    <Year id="2007">80000</Year>
    <Year id="2008">100000</Year>
    <Year id="2009">95000</Year>
    <Year id="2010">80000</Year>
  </Region>
  <Region id="Asia">
    <Year id="2005">10000</Year>
    <Year id="2006">25000</Year>
    <Year id="2007">70000</Year>
    <Year id="2008">110000</Year>
    <Year id="2009">125000</Year>
    <Year id="2010">150000</Year>
  </Region>
</Data>
```

After opening the SPS file in StyleVision, switch to Authentic Preview, and, in the Chart Type combo box (screenshot below), select, one after the other, Bar Chart 2D, Bar Chart 3D, Bar Chart 3D Grouped, and Bar Chart Stacked to see the four kinds of chart.
The first three charts of the 2D bar charts page and 3D bar charts page show the yearly sales of
the three sales regions. Each of these bar charts (see screenshot of a 3D bar chart below)
represents a single region, and the bars represent the region's sales per year for the six years
from 2005 to 2010.

To view the chart settings of these charts, switch to Design View, select the appropriate branch
(Bar Chart 2D or Bar Chart 3D) in the first condition of the design, right-click the chart, and
select Edit Chart Settings. This pops up the Chart Configuration dialog.

Note the following points about the Regional Yearly Sales charts:

1. The charts have been inserted within the Region element, so the context node is the
   Region element. A chart is therefore created for each iteration of the XSLT Engine over
   the Region element.
2. The title of each chart contains the value of the Region/@id attribute. Click the Dynamic
   XPath Settings button to see how this is done.
3. Since no more than one series is required, we can leave the Z-Axis or Series Name Axis empty. This results in a default no-name series being created.

4. The X-Axis selects the labels to be used for the bars of the bar chart, in this case the years 2005 to 2010, which are obtained with the XPath expression `Year/@id` (see screenshot below).

5. The Y-Axis selects the values of each bar of the bar chart. The number of bars in the chart is equal to the number of items in the sequence returned by the XPath expression for the Y-Axis. In our example, the XPath expression returns a sequence of items that are the contents of the `Year` element children of the `Region` element. The vertical number scale of the Y-Axis is automatically generated on the basis of the items selected for the Y-Axis.

6. The number of bars in the bar chart is determined according to the Chart Data Selector Rules. So, if the selection were made as in the screenshot below, then another bar for the year 2011 having a value of 60000 would be added to the chart.

7. The items of the sequence selected for the X-Axis will be assigned, in corresponding order, as the label for each bar of the bar chart.

8. The number of X-Axis and Y-Axis items in each data selection row of the Chart Data Selector pane (see screenshot above) must be the same, though it is not an error if they are not the same. If there are fewer X-Axis items than Y-Axis items, then the latter bars (which do not have a corresponding X-Axis item) will be unlabelled. If there are more X-Axis items than Y-Axis items, then the extra X-Axis items will be ignored for label-generation.

---

**Using the Series Axis to generate grouped bars and stacked bars**

If we have the years on the X-Axis, it would be convenient to group region bars for a single year together, as in the screenshot below. Such grouped charts are present in the example file: the last chart, respectively, in the Bar Chart 2D (screenshot below) and Bar Chart 3D pages. The one chart in the Bar Chart 3D Grouped page is also a grouped chart. Stacked bar charts use the same data selection principles as grouped bar charts, but are different from them in appearance. They are discussed farther below.
The grouping of bars in these charts is achieved with the Z-Axis or Series Name Axis. For the Series Name Axis, we select the names of the three regions. To do this, we create the chart, not within a Region element, but one level higher (within the Data element; refer to XML document above), so that we can now target all three Region elements easily. Our XPath expressions for the three axes would be as in the screenshot below.

Note the following points about these charts that group all the regions by year:

- The context node of the chart is one level higher than the Region element: It is the Data element.
- The Z-Axis (Series Name Axis) selects the name of the region (Region/@id).
- As with the bar charts showing regional yearly sales, the X-Axis selects the names of the years (Region/Year/@id) and the Y-Axis selects the sales for each year (content of the
What the data selection above does is this: For each Region element, it selects a series name (the value of the Region element’s id attribute), then generates the X-Axis for the Region (using the sequence obtained with the values of the Year/@id attributes). It then generates the Y-Axis values (using the sequence obtained with the content of the Year elements). This is done for all three Region elements, in sequence, generating the chart shown above.

For more information about data selection for the various axes, see the section [Rules for Chart Data Selection](#).

The same effect is obtained with the data selection shown in the screenshot below. In this data selection, each region and its axes are selected individually, with one region following on another. Note that, since the X-Axis items of the second and third series will be ignored, it is safe to leave these fields in the dialog box empty.

Yet another way to select the same data would be with the XPath expressions in the screenshot below.

Notice that the X-Axis expression in all three data selection rows is identical. What is required is that the XPath data selection returns the correct sequence of items. The sequence can also be entered directly in the XPath expression. For example: "2005, 2006, 2007, 2008, 2009,"
Stacked bar charts

Stacked bar charts, such as the stacked bar chart in the Bar Chart Stacked page of the example file, YearlySales.xml (screenshot below), use the same data selection principles as those for grouped charts. All three axes are selected exactly as they are selected for the grouped charts described above. However, instead of the bars of a group being located side-by-side, they are stacked on top of each other, so that for each X-Axis tick, each member of the series at that X-Axis tick is represented within a single stacked bar (screenshot below).

Stacked bar charts provide a good view of the break-up of a composite. For example, in the screenshot above, each bar is a composite that represents total sales in a specific year and it is composed of the yearly sales of each of the regions.

See also

- Charts
- Chart Settings
- Chart Data Selection: Simple
- Chart Data Selection: Flexible
Example: Line Charts

In this section, we discuss the line graphs (or category line graphs) contained in the SPS file YearlySales.sps. The SPS design is based on the XML Schema YearlySales.xsd and it uses YearlySales.xml (screenshot below) as its Working XML File. All three files are located in the (My) Documents folder, C:\Documents and Settings\username\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial\Charts.

XML file used in charts example: YearlySales.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="YearlySales.xsd">
    <ChartType>Pie Chart 2D</ChartType>
    <Region id="Americas">
        <Year id="2005">30000</Year>
        <Year id="2006">90000</Year>
        <Year id="2007">120000</Year>
        <Year id="2008">180000</Year>
        <Year id="2009">140000</Year>
        <Year id="2010">100000</Year>
    </Region>
    <Region id="Europe">
        <Year id="2005">50000</Year>
        <Year id="2006">60000</Year>
        <Year id="2007">80000</Year>
        <Year id="2008">100000</Year>
        <Year id="2009">95000</Year>
        <Year id="2010">80000</Year>
    </Region>
    <Region id="Asia">
        <Year id="2005">10000</Year>
        <Year id="2006">25000</Year>
        <Year id="2007">70000</Year>
        <Year id="2008">110000</Year>
        <Year id="2009">125000</Year>
        <Year id="2010">150000</Year>
    </Region>
</Data>
```

After opening the SPS file in StyleVision, switch to Authentic Preview, and, in the Chart Type combo box (screenshot below), select Line Charts to see the line charts.
The line graph charts show the yearly sales of each region (see screenshot below). The X-axis of the chart shows the years, the Y-axis shows the units sold. Each line graph shows the sales of a region over the period 2005 to 2010. The legend key at the bottom indicates which line represents which region.

This chart was created within the Data node, which is the parent of the Region nodes (refer to XML file above). The data selection is as shown in the screenshot below.
Note the following points about this data selection:

- The Z-Axis (Series Axis) selects the name of the region (Region/@id).
- The X-Axis selects the names of the years (Region/Year/@id) and the Y-Axis selects the sales for each year (content of the Region/Year element).

What the data selection above does is this: For each Region element, it selects a series name (the value of the Region element's id attribute), then generates the X-Axis for the Region (using the sequence obtained with the values of the Year/@id attributes). It then generates the Y-Axis values (using the sequence obtained with the content of the Year elements). This is done for all three Region elements, in sequence, generating the line graphs shown above.

For more information about data selection for the various axes, see the section Rules for Chart Data Selection.

See also
- Charts
- Chart Settings
- Chart Data Selection: Simple
- Chart Data Selection: Flexible
Example: Value Line Charts

The difference between value line graphs and category line graphs is that, in the former, the X-Axis takes numeric values that are the domain of the function whose range is represented on the Y-Axis. For example, for the function $y=3x$, the Y-Axis plots values that are three times the value of corresponding X-Axis values, for example: $(-1, -3)$ $(0, 0)$ $(1, 3)$ $(2, 6)$. Category line graphs on the other hand take X-Axis values that are text values and their Y-Axis values are not functions of X-Axis values.

The value line chart examples in this section are located in the (My) Documents folder, C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples/Tutorial/Charts. In the first example (see screenshot below), TeamTrials.sps (using TeamTrials.xsd and TeamTrials.xml), the X-Axis show time in hours. The Y-Axis values show distance traveled in time and they are a function of X-Axis values: $distance\ travelled = speed \times time$. 
The XML file contains the speeds of the various teams. The context node for the chart is the TeamTrials element in the XML document. The Z-Axis (Series Axis) selects the names of the teams (Team/@name). The X-Axis is a sequence of integers generated by an XPath expression.
The Y-Axis value of each graph (one per team) is derived by multiplying the X-Axis value (hours) by the speed of that team ($Team/@averagespeed$). The screenshot below shows the XPath expressions that were used to select data for the three axes.

More value line graphs
The second example, ValueLineGraphs.sps (using ValueLineGraphs.xsd and ValueLineGraphs.xml), contains more line graphs (screenshot below). The XPath expressions (flexible) to select values for the various graphs are shown in the second screenshot below.
Note the following points about the XPath expressions shown above:

- For the first graph, the X-Axis values are generated with the XPath expression for the X-Axis (Category) data selection.
- For all the other graphs except the first graph, the X-Axis values are generated with an XPath expression that outputs the context node.
- Java functions are called with `java:java.lang.Math`. We have used two Java functions: `cos()` and `sqrt()`.
- .NET functions are called with `clitype:System.Math`. We have used one .NET function: `Sin()`.
- The X-Axis and Y-Axis ticks have been set manually (in the Change Appearance dialog) at intervals of 2 units and 1 unit, respectively.

See also

- Charts
- Chart Settings
- Chart Data Selection: Simple
- Chart Data Selection: Flexible
Example: Area Charts

In this section, we discuss the area charts contained in the SPS file YearlySales.sps. The SPS design is based on the XML Schema YearlySales.xsd and it uses YearlySales.xml (screenshot below) as its Working XML File. All three files are located in the (My) Documents folder, C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2015 \StyleVisionExamples\Tutorial\Charts.

XML file used in charts example: YearlySales.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="YearlySales.xsd">
  <ChartType>Pie Chart 2D</ChartType>
  <Region id="Americas">
    <Year id="2005">30000</Year>
    <Year id="2006">90000</Year>
    <Year id="2007">120000</Year>
    <Year id="2008">180000</Year>
    <Year id="2009">140000</Year>
    <Year id="2010">100000</Year>
  </Region>
  <Region id="Europe">
    <Year id="2005">50000</Year>
    <Year id="2006">60000</Year>
    <Year id="2007">80000</Year>
    <Year id="2008">100000</Year>
    <Year id="2009">95000</Year>
    <Year id="2010">80000</Year>
  </Region>
  <Region id="Asia">
    <Year id="2005">10000</Year>
    <Year id="2006">25000</Year>
    <Year id="2007">70000</Year>
    <Year id="2008">110000</Year>
    <Year id="2009">125000</Year>
    <Year id="2010">150000</Year>
  </Region>
</Data>
```

After opening the SPS file in StyleVision, switch to Authentic Preview, and, in the Chart Type combo box (screenshot below), select Area Charts to see the two kinds of area chart: (i) standard overlay, and (ii) stacked.
Selecting data for area charts

If we have the years on the X-Axis and the unit sales on the Y-Axis, then the yearly sales of each region can be plotted as an area, with each subsequent area overlaying the previous areas (screenshot below). Area charts are selected via the Change Type dialog. For area charts in which some areas overlay parts of other areas, it is advisable to set transparency for the areas—an option available for area charts via their Chart Settings.

The data selection for area charts is done as follows: For the Series Name Axis, we select the names of the three regions. To do this, we create the chart, not within a Region element, but one level higher (within the Data element; refer to XML document above), so that we can target all three Region elements easily. Our XPath expressions for the three axes would be as in the screenshot below.
Note the following points about this data selection:

- The context node of the chart is one level higher than the Region element: It is the Data element.
- The for-each iteration is created on the Region element.
- The Z-Axis (Series Name Axis) selects the name of the region (Region/@id).
- As with the bar charts showing regional yearly sales, the X-Axis selects the names of the years (Region/Year/@id) and the Y-Axis selects the sales for each year (content of the Region/Year element).

What the data selection above does is this: For each Region element, it selects a series name (the value of the Region element’s id attribute), then generates the X-Axis for the Region (using the sequence obtained with the values of the Year/@id attributes). It then generates the Y-Axis values (using the sequence obtained with the content of the Year elements). This is done for all three Region elements, in sequence, generating the chart shown above.

For more information about data selection for the various axes, see the section Rules for Chart Data Selection.

The same effect is obtained with the data selection shown in the screenshot below. In this data selection, each region and its axes are selected individually, with one region following on another. Note that, since the X-Axis items of the second and third series will be ignored, it is safe to leave these fields in the dialog box empty.
Yet another way to select the same data would be with the XPath expressions in the screenshot below.

Notice that the X-Axis expression in all three data selection rows is identical. What is required is that the XPath data selection returns the correct sequence of items. The sequence can also be entered directly in the XPath expression. For example: "2005, 2006, 2007, 2008, 2009, 2010". Something similar has actually been done for the Series Axis.

**Stacked area charts**

Stacked area charts (screenshot below) use the same data selection principles as for the standard overlay area charts. All three axes are selected exactly as described above. However, instead of areas being superimposed on the same coordinate grid, the areas are stacked along the Y-Axis so that Y-Axis values are cumulative (screenshot below).
Stacked area charts enable multiple areas to be compared. For example, in the screenshot above, the sales of the three regions (Americas, Europe, and Asia) can be compared by observing the sizes of their areas over a particular period. In the chart, notice that the plotted Y-Axis value of a series in a given year is the sum of that series’ original Y-Axis value plus the Y-Axis values of all the series plotted below it. So Europe in 2005 is plotted at 80,000 (50,000 + 30,000 (of Americas)) while Asia is plotted at 90,000 (10,000 + 50,000 (Europe) + 30,000 (Americas)).
Example: Candlestick Charts

Candlestick charts (screenshot below) are most commonly used to show the movement of stock prices. They provide at-a-glance information about the opening and closing prices, and the highest and lowest prices during a trading day.

The screenshot above, for example, shows candlesticks of the share prices of a fictitious company, MyStock, for the trading days from 11 April 2011 to 20 April 2011. The highest and lowest points of the vertical line in each candlestick indicates the day's highest and lowest trading prices, respectively (see table below for prices). A color-filled candlestick body indicates a drop in the price over the day's trading, from the opening price to the closing price, an unfilled body indicates a price rise. So, on 11 April, we see that the price dropped from 210 to 203, whereas on 12 April the price rose from 205 to 218.

A second type of candlestick chart (screenshot below) shows three values: (i) the highest price traded on the day, (ii) the lowest price, and (iii) the closing price (indicated by the short horizontal line).
Candlestick chart example files
Files for the candlestick chart example discussed in this section are located in the (My) Documents folder, C:\Documents and Settings\<username>\My Documents\Altova \StyleVision2015\StyleVisionExamples\Tutorial\Charts. The SPS file Candlestick.sps (see screenshot below) uses Candlestick.xsd and Candlestick.xml. In Authentic View (screenshot below), the user can select what should be displayed in the candlestick chart. The selection options are: (i) the company (from two companies); (ii) the period for which prices are to be displayed; (iii) the extent of the Y-Axis; and (iv) the interval of Y-Axis ticks. Furthermore, the table below the chart allows new trading data to be entered for the two companies (right-click in a row and append or insert a Day element).
Stock Prices

Select the stock you wish to view and the period for which you wish to view its price movement. You can also modify the value-range of the Y-Axis.

Stock: MyStock
Start date: 2011-Jan-03
End date: 2011-Jun-28
Y-Axis start: 80 (Lowest value in selected period: 88)
Y-Axis end: 480 (Highest value in selected period: 460)
Y-Axis tick interval: 20

Notice that the top part of the design contains combo boxes and input fields in which the Authentic user can specify the parameters of the chart that is displayed below. These input specifications are stored within the XML file and are used to calculate some key chart settings.

XML file
The allowed structure of the XML document is as shown in the listing below. The document element Trades contains one Selection element and an unlimited number of Stock elements. Each Stock element can contain an unlimited number of Day elements.

<Trades xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
The Selection element contains information supplied by the Authentic user and this information is used to select what data is displayed in the chart. The data selection for the chart is therefore based on information in the Selection element. The exact details are explained in the next section. The data about the price movements of each stock on a given day is contained in the Stock/Day element of that stock on that given day.

Data selection for the candlestick chart
The data selection for the chart is specified in the Chart Configuration dialog (see screenshot below), which is accessed by right-clicking the chart and selecting Edit Chart Settings. The data selection was done using the Simple method.
The settings are made as follows:

- **In the design, the chart is inserted within a Stock element.** This Stock element has a filter to select the Stock element with an @id attribute equal to the content of the Selection/Stock element. So the context node within which the chart is created is the Stock element that the Authentic user selects in the Stock combo box.

- **The For-Each XPath expression selects all Day elements that have an @id attribute which greater than or equal to the StartDate the Authentic user specifies and greater than or equal to the users EndDate.** This ensures that data for the dates selected by the Authentic user will now be processed. The context node for all the other XPath expressions used for data selection in this dialog will therefore be the Day element (see example structure below).

```xml
<Day id="20110104" year="2011" month="Jan" week="01" date="04">
  <Open>100</Open>
  <High>112</High>
  <Low>92</Low>
  <Close>109</Close>
</Day>
```

- **On the X-Axis of the chart, we wish to show the dates.** So we use the XPath expression: `concat(@date, '-' @month)`. This expression serves to collect the date from the attributes of the respective Day elements.

- **The Y-Axis plots the stock prices.** We use four series for our candlestick chart: (i) the opening prices, (ii) the highest price of the day, (iii) the lowest price of the day, and (iv) the closing price. For each of these series, we identify, respectively, the Day/Open, Day/High, Day/Low, and Day/Close elements.

**Important:** The order in which the series are specified is important, and must be as follows:

- **Four-series charts:** (i) Open, (ii) High, (iii) Low, (iv) Close.
- **Three-series charts:** (i) High, (ii) Low, (iii) Close.

The nodes that contain the values that will be used for the various series may have any valid XML name (they need not be called Open, High, etc) but these nodes must contain number values.

---

**Other settings in the example file**

Some other crucial chart settings were obtained as explained below. Many are obtained dynamically from the XML file and needed to be set using Dynamic XPath settings.
• Since the name of the chart needs the Authentic user's date specifications, the chart title is set dynamically.
• The Y-Axis range and interval is also determined by the user and therefore requires dynamic XPath settings.

**See also**

- [Charts](#)
- [Chart Settings](#)
- [Chart Data Selection: Simple](#)
- [Chart Data Selection: Flexible](#)
Example: Gauge Charts

In this section, we discuss the gauge charts contained in the SPS file YearlySales.sps. The SPS design is based on the XML Schema YearlySales.xsd and it uses YearlySales.xml as its Working XML File. All three files are located in the (My) Documents folder, C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial/Charts.

After opening the SPS file in StyleVision, switch to Authentic Preview, and, in the Chart Type combo box (screenshot below), select Gauge to see the gauge charts.

On the gauge charts (screenshot below), selecting options in the Select Region and Select Year combo boxes causes the needle in each of the two gauges (round and bar) to point to the correct value on the gauge scale. The screenshot below shows sales in Asia in 2009, as an absolute value (125000) and as a percentage of total sales in that year (34.7%).
The settings for both charts are similar. Here we consider the first gauge chart: the round gauge chart. To generate a gauge chart a single value is needed, and this is provided by the Y-Axis entry. In the round gauge chart this value is the yearly sales of a user-selected region in a user-selected year. The Y-Axis must accordingly select this data. The X-Axis and Series Name Axis entries are ignored. The XPath expression to select the required data is created as follows:

1. The user-selected region is stored in an editable variable for Authentic ($SelectRegion), and the user-selected year is stored in an editable variable for Authentic ($SelectYear).
2. These two editable variables are used in predicates to filter which Region element and then which of its Year child elements is selected as the data for the Y-Axis (see the XPath expression in the screenshot below).
3. The following gauge settings (accessed by right-clicking the chart in the design, then selecting **Edit Chart Settings | All Settings | Gauge**) were modified. The scale was changed to a maximum of 200000, and the major and minor ticks, were set, respectively to 20000 and 10000. Major and minor ticks are the major and minor divisions on the scale (see screenshot above).

**Note:** The scale must be set to the correct range and ticks must be suitably adjusted in order for gauge charts to display correctly.

For more information about data selection for the various axes, see the section **Rules for Chart Data Selection**.

---

**See also**
- Charts
- Chart Settings
- Chart Data Selection: Simple
- Chart Data Selection: Flexible
**Example: Overlay Charts**

Overlay charts *(screenshot below)* enable you to combine multiple charts of different types. The following charts can be combined:

- 2D bar charts and stacked bar charts
- Line charts
- Area charts and stacked area charts
- Candlestick charts

---

**Yearly Sales by Region (Units Sold)**

*Chart type: Overlay Chart*

The chart below has three layers. The bottom layer is an area chart showing the total sales of all three regions (Americas, Europe, Asia). The next layer is a bar chart layer which shows the sales of the region selected in the combo box. The third and topmost layer is a layer with three line graphs, showing the sales of each of the three regions separately. The three Y-Axes, starting from left, correspond to the three chart layers starting from the bottom layer. In Authentic View, you can modify values in the regional tables below (not the summary table) to see the effect on the chart.

Region to be displayed in bar chart: Americas

---

**Yearly Sales: Regional and Total**

*Graph showing yearly sales with different regional and total categories*
An overlay chart (shown in the screenshot above) is contained in the SPS file *YearlySales.sps*. The SPS design is based on the XML Schema *YearlySales.xsd* and it uses *YearlySales.xml* as its Working XML File. All three files are located in the *(My) Documents folder*, `C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial\Charts`.

After opening the SPS file in StyleVision, switch to Authentic Preview, and, in the *Chart Type* combo box *(screenshot below)*, select *Overlay Charts* to see the overlay chart shown above.

---

### Creating the overlay chart

An overlay chart is created by selecting the *Multiple Layers* radio button in the *Chart Configuration dialog* *(see screenshot below)*. A main layer is automatically created. You can select the *type of chart* for the main layer and then make the *chart settings* and *data selection* in the usual way.

![Chart Configuration](chart_configuration.png)

### Managing the layers of the overlay chart

You can add new layers to an overlay chart and delete layers. To do this, click the *Manage* button in the Chart Configuration dialog *(see screenshot above)*. This pops up the Overlays dialog *(screenshot below)*. Click *Add* to add a new layer. To delete a layer, select a layer and click *Delete*.

![Overlays](overlays.png)

You should note that when a layer is placed over another, it will obscure the layers beneath it. Since only area charts can be made transparent, some layer arrangements might not be optimal. For example a bar chart that is layered over a line chart would obscure parts of the lines. You should keep this in mind when planning the layering order.
Data selection for the various charts
The data selection for each chart depends on the chart type and is done in the usual ways for each chart type. See the sections, Chart Data Selection: Simple and Chart Data Selection: Flexible for an overview of chart data selection, and the respective example sections for details about chart data selection for individual chart types.

The following general points relating to overlay charts are important and should be noted:

- The X-Axis for all chart layers is common and is the X-Axis specified in the main layer. If this X-Axis definition cannot be evaluated for another chart, then that chart will not be drawn. X-Axes specified in chart definitions of other layers are ignored.
- The Y-Axes series for each chart are selected separately. The Y-Axes of each layer is drawn parallel to each other, starting from the left for the bottom-most layer (the main layer) and proceeding to right with each layer. Note that a label can be specified for each Y-Axis, in its chart settings.

Important chart settings
The following chart settings are relevant:

- The title of the chart is specified in the Chart Title setting of the main layer.
- X-Axis settings are specified in the settings for the main layer.
- Y-Axis labels can be set for each chart separately.
- Y-Axis start points (minimum) and end points (maximum), as well as the tick intervals, can be specified for each chart separately. This enables axes to be calibrated relative to each other.
- The colors of series in each chart can be selected separately.

See also
- Charts
- Chart Settings
- Chart Data Selection: Simple
- Chart Data Selection: Flexible
8.8 Form Controls

Nodes in the XML document can be created as data-entry devices (such as input fields and combo boxes). Data-entry devices are intended for easier editing in Authentic View. For example, an input field makes it clear to the Authentic View user that input is expected in this location while a combo box lists, as well as restricts, the values a user can enter. When data is entered into a data-entry device, the data is inserted into the XML document as element content or as an attribute's value. In the HTML, RTF, PDF, and Word 2007+ output, the data-entry device is rendered as an object that is the same as that displayed in Authentic View, or a near-equivalent. Note that data-entry devices accept input to the XML document and, therefore, will not work in the HTML output.

General mechanism
Given below is a list of the data-entry devices available in StyleVision, together with an explanation of how data is entered in the XML document 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 is mapped to a value.</td>
</tr>
<tr>
<td>Check box</td>
<td>User selection is mapped to a value.</td>
</tr>
<tr>
<td>Radio button</td>
<td>User selection is mapped to a value.</td>
</tr>
<tr>
<td>Button</td>
<td>User selection is mapped to a value.</td>
</tr>
</tbody>
</table>

The text values entered in the input fields are entered directly into the XML document as XML content. For the other data-entry devices, the Authentic View user's selection is mapped to a value. StyleVision enables you to define the list of options the user will see and the XML value to which each option is mapped. Typically, you will define the options and their corresponding values in a dialog.

General usage
To create a data-entry device, do the following:

1. Drag a node from the Schema Tree window into Design View and drop it at the desired location.
2. From the context menu that appears, select the data-entry device you wish to create the node as.
3. For some data-entry devices, a dialog pops up. In these cases, enter the required information in the dialog, and click OK.

To reopen and edit the properties of a data-entry device, select the data-entry device (not the
node containing it), and edit its properties in the Properties sidebar.

**Note:**
- Data can be entered in data-entry devices only in Authentic View.
- Data-entry devices can also be created by changing the current component type of a node to a data-entry device. To do this right-click the node and select **Change to**.
- In the HTML, RTF, PDF, and Word 2007+ output, the entry selected by the user is displayed in the output. Changing the value of a data-entry device in the HTML document does not change the text value in either the XML document or HTML document.
- In the case of some data-entry devices, such as check boxes, where the device cannot correctly be rendered in print, an alternative rendition is implemented.

**See also**
- Input Fields, Multiline Input Fields
- Combo Boxes
- Check Boxes
- Radio Buttons, Buttons
Input Fields, Multiline Input Fields

You can insert an Input Field or a Multiline Input Field in your SPS when you drop a node from the Schema Sources window into Design View. The text that the Authentic View user enters into these fields is entered into the XML node for which the field was created. The content of that node is displayed in the input field or multiline input field.

Editing the properties of input fields

You can modify the HTML properties of input fields by selecting the input field and then modifying its HTML properties in the Properties sidebar (see screenshot below).

For example, with the input field selected, in the Properties window select editfield, select the HTML group of properties and the maxlength property. Then double-click in the Value field of maxlength and enter a value.

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.

See also
- Combo Boxes
- Check Boxes
- Radio Buttons, Buttons
Check Boxes

You can create a check box as a data-entry device. This enables you to constrain user input to one of two choices. In the Edit Check Box dialog (shown below), you specify the XML values to map to the checked and unchecked events.

In the above screenshot, an element called Name has been created as a check box. If the Authentic View user checks the check box, a value of true will be entered as the value of the element Name. If the value is unchecked, then the value false is entered as the XML value of Name (as defined in the dialog).

Note: When a new Name (or check box) element is created in Authentic View, its XML value is empty (it is not the Unchecked Value). The Unchecked Value is entered only after the check box has first been checked, and then unchecked. To have a default value in a node, create a Template XML file that contains the default value.

Accessing the Edit Check Box dialog

If you are creating a new check box, when you create the node as a check box, the Edit Check Box dialog pops up. To access the Edit Check Box dialog afterwards, do the following:

1. Select the check box in the design.
2. In the Properties sidebar, select the checkbox item and then the checkbox group of properties (see screenshot below).
3. Click the Edit button of the `check values` property. This pops up the Edit Check Box dialog.

**Note:** You can modify the HTML properties of a check box by selecting it and then modifying its HTML properties in the Properties sidebar.

---

- **See also**
  - [Input Fields, Multiline Input Fields](#)
  - [Combo Boxes](#)
  - [Radio Buttons, Buttons](#)
**Combo Boxes**

A combo box presents the Authentic View user with a list of options entries in a dropdown list. The selected option is mapped to a value that is entered in the XML document. The mapping of drop-down list entry to XML value is specified in the SPS.

Mappings can be made in the Edit Combo Box dialog in one of three ways:

- From the schema enumerations for the selected node. In this case, the visible entry (in the dropdown list) will be the same as the XML value.
- From a list defined in the Edit Combo Box dialog. You enter the visible entry and the corresponding XML value, which may be different.
- From the result sequence of an XPath expression relative to the current node. The items in the result sequence are displayed as the entries of the drop-down list, and the list entry selected by the Authentic View user is entered as the value of the node. This is a powerful method of using dynamic entries in the combo box. The node that you create as the combo box is important. For example, say you have a `NameList` element that may contain an unlimited number of `Name` elements, which themselves have `First` and `Last` children elements. If you create the `Name` element as a combo box, and select the `Last` child element for the list values, then, in Authentic View, you will get as many combo boxes as there are `Name` elements and each combo box will have the `Last` child as its dropdown menu entry. In order to get a single combo box with all the `Last` elements in the dropdown menu list, you must create the single `NameList` element as the combo box, and select the `Last` element in the XPath expression.

---

**Accessing the Edit Combo Box dialog**

If you are creating a new combo box, when you create the node as a combo box, the Edit Combo Box dialog pops up. You can also insert a combo box with the (Insert | Insert Form Controls | Combo Box) menu command. To access the Edit Combo Box dialog afterwards, do the following:

1. Select the combo box in the design.
2. In the Properties sidebar, select the combo box item and then the combo box group of properties (see screenshot below).

![Properties](image)

3. Click the Edit button of the content origin property. This pops up the Edit Combo Box dialog.
Using the Edit Combo Box dialog
The Edit Combo Box dialog is shown below.

To define the entries and values for the combo box, do the following:

1. Select the method with which you wish to define the entries and values by clicking the appropriate radio button: (i) schema enumerations, (ii) list of values, or (iii) XPath expressions to select values.

2. If you select Schema Enumerations, the enumerations assigned to that node in the schema are entered automatically as (i) the visible entries of the drop-down list of the combo box, and (ii) the corresponding XML values (screenshot below). Visible Entries are the entries in the drop-down list of the combo box. Each drop-down list entry has a corresponding XML value. The XML value corresponding to the visible entry that the Authentic user selects will be the XML value that is entered in the XML file. Both visible entries and XML values are grayed out in the list of values because they are both obtained
from the schema enumerations and cannot be edited.

If you select **Use List of Values**, you can insert, append, edit, and delete any number of entries for the drop-down list of the combo box as well as for the corresponding XML values. These edits are carried out in the pane below the **Use List of Values** radio button. You could also use an XPath expression to create the visible entries and XML values. The items in the sequence returned by the XPath expression will be used for visible entries and XML values. You can specify: (i) that the same XPath expression be used for visible entries and XML values, or (ii) that different XPath expressions be used. In the latter case, a one-to-one index mapping between the items of the two sequences determines the correspondence of visible entry to XML value. If the number of items in the two sequences are not equal, an error is reported.

3. If you wish to have the items that appear in the drop-down list of the combo box in **Authentic View** sorted, check the **Sort Values in Authentic** check box.

4. Click **OK** to finish.

**Note**
- Using an XPath expression to select the items of the combo box drop-down list enables you to create combo boxes with dynamic entries from the XML file itself.
- If the items in the drop-down list of the combo box are obtained from schema enumerations, they will be sorted alphabetically by default. If the items are obtained from an XML data file, they will appear in document order by default. If the items are obtained from a DB, the DB schema must be set as the main schema. If items are obtained from a DB that is not the main schema, a template for the DB row targeted by the XPath expression must be included in the design, even if the template must be empty. Additionally, in such cases, make sure that all instances of the targeted row are fetched.

**See also**
- Using Conditions
- Input Fields, Multiline Input Fields
- Check Boxes
- Radio Buttons, Buttons
Radio Buttons, Buttons

There are two types of button: radio buttons and buttons. Radio buttons allow the Authentic View user to enter data into the XML file. Buttons do not allow data-entry in Authentic View, but are useful for triggering events in the HTML output.

Radio buttons
Inserting radio buttons in the SPS allows you to give the user a choice among multiple alternatives. Each radio button you insert maps to one XML value. The user selects one radio button. The radio buttons for a node are mutually exclusive; only one may be selected at a time, and the associated XML value is entered as the value of the node. The way to use this feature is to create the node for which the data-entry is required multiple times as a radio button. For each radio button enter (i) some static text to indicate its value to the user, and (ii) an XML value for each radio button. To edit the XML value of a radio button, in the Properties sidebar, select the radio button item, then click the Edit button of the radio button property. This pops up the Edit Radio Button dialog.

Buttons
The button option allows you to insert a button and specify the text on the button. This is useful if you wish to associate scripts with button events in the generated HTML output. Note, however, that a button does not map to any XML value and does not allow data entry in Authentic View. However, the value of the parent node of a button can be selected by the Authentic View user, and the SPS can be designed to modify presentation based on what the Authentic View user selects.

Note: You can modify the HTML properties of a radio button or button by selecting it and then modifying its HTML properties in the Properties sidebar.

See also
- Input Fields, Multiline Input Fields
- Combo Boxes
- Check Boxes
8.9 Links

Links (or hyperlinks) can be created to bookmarks located in the document as well as to external resources like Web pages. Links can also be created to dynamically generated anchors. StyleVision offers considerable flexibility in the way target URIs for hyperlinks can be built.

The section, Bookmarks and Hyperlinks, describes how to create static and dynamic bookmarks in the document and how to link to bookmarks as well as to external documents.

- See also
  - Bookmarks and Hyperlinks
8.10 Barcodes

The Barcode design element is supported in **XSLT 2.0 or XSLT 3.0 mode** (not XSLT 1.0) and enables barcodes (*screenshot below*) to be generated in the output document. At the location in the design document where you wish to enter the barcode, insert the Barcode design element and specify its properties.

![Barcode Image]

**Important:** For barcodes to work, a Java Runtime Environment must be installed. This must be version 1.4 or later in a bit version that corresponds to the bit version of the StyleVision package installed on your system: 32-bit or 64-bit.

---

**Inserting a barcode**

To insert a barcode in your design, do the following:

1. At the location where you wish to insert the barcode, right-click and select the command **Insert Barcode**. Alternatively, select the command **Insert | Insert Barcode** or click the Barcode icon in the toolbar and click the location in the design where you wish to insert the barcode. You can also drag and drop an element from the Schema Tree into the Design View and then select 'Create Barcode'. The Insert Barcode dialog pops up (*screenshot below*).
2. Two properties, Type and Text, are mandatory; the others are optional and/or have appropriate default values. The Type property, the value of which can be selected from a dropdown list (see screenshot above), specifies the type of the barcode, for example EAN-13 (which includes ISBN barcodes) and UPC-A. The Text property specifies the value that will generate the barcode, for example, an ISBN number. The various barcode properties are described below. Set the required properties and any other properties that you want. Note that, if you wish to use a value in the XML file as the value of a property, you can enter an XPath expression to locate the XML node you wish to access. Do this as follows: Select the property, toggle on the XPath button in the toolbar of the Properties dialog, and then enter the XPath expression in the Edit XPath Expression dialog. The XPath expression will be evaluated within the current context node.

3. After setting the properties, click OK. The barcode image will be inserted. The generated barcode (see screenshot below) can be immediately viewed in any of the output previews.

Note: Barcode images are generated as PNG files.
Barcode properties
The following barcode properties can be specified. The Type and Text properties must be set; the other properties are optional. Note that different properties are available for different barcode types.

- **Type**: The barcode system under which the message will be interpreted, such as EAN and UPC.
- **Text**: The value that will be used to generate the barcode pattern.
- **SetModuleWidth**: The width of the bars in the code.
- **SetBarHeight**: The height of the bars.
- **SetHeight**: The height of the barcode graphic.
- **DoQuietZone**: Yes or No values determine whether the "quiet zone" (or padding) around the barcode, which is specified in the SetQuietZone and SetVerticalQuietZone properties, will be implemented.
- **SetQuietZone**: Sets the "quiet zone" (or padding) around the barcode. In the case of one-dimensional barcodes, the value specified here is applied to the horizontal dimension. In the case of two-dimensional barcodes, the value is applied to both horizontal and vertical dimensions. The value of the vertical dimension can be overridden by the value specified in the SetVerticalQuietZone property. A length unit of millimeters (mm) is required. Example: 2mm.
- **SetVerticalQuietZone**: Sets the "quiet zone" (or padding) for the vertical dimension on two-dimensional barcodes. A length unit of millimeters (mm) is required. Example: 2mm.
- **SetMsgPosition**: Specifies where the message text appears relative to the barcode. Values are top, bottom, and none (no message is generated).
- **SetPattern**: Sets a pattern for the message text so that the text is readable. A long string of numbers, for example, would be difficult to read. The syntax for patterns is given below.
- **SetFontName**: The font in which text should appear.
- **SetFontSize**: The font-size in which text should appear.
- **SetChecksumMode**: The following values are available: (i) Add: the checksum is automatically added to the message; (ii) Check: the checksum is checked while rendering the barcode (assumes the checksum is present); (iii) Ignore: no checksum processing is done; (iv) Auto: enables the barcode type's default behaviour.
- **Orientation**: Whether the barcode should be rotated. The options are in steps of 90 degrees counter-clockwise.
- **PixelDensity**: Specifies the density of the pixels in the barcode image. Higher pixel density provides sharper images.
- **GeneratedImageSettings**: Enables you to set a name for the generated barcode image file. If no name is specified, a name is generated automatically by StyleVision.

Pattern syntax
Patterns are used to make the input message string more readable in the barcode. In the pattern, each character of the input message text is represented by the underscore " _ ". Any other characters included in the pattern are inserted at the corresponding locations in the output message text. The backslash " \ " is an escape symbol. So, the combination of " \? " will insert the character '?' in the output message text, where '?' can be any character. The character ' ' can be used to delete a character from the original message. These points of pattern syntax are illustrated with the examples below.

<table>
<thead>
<tr>
<th>Input message text</th>
<th>Pattern</th>
<th>Output message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Generating output files

The barcode image files that are generated for the output are saved to locations that are specified in the Paths tab of the Properties dialog (screenshot below), which is accessed with the menu command File | Properties.

Barcode image files for previews may be created in the same directory as the SPS file or as the Working XML File. These are temporary files, which are deleted when the SPS is closed. Barcode image files that are created when output is generated using the File | Save Generated File command can be created at any location. Their target location is specified in the pane, Location of Additionally Generated Files (see screenshot above).

See also
• SPS File: Content
8.11 Layout Modules

Layout Modules are objects containing a layout. The module as a whole is inserted in the SPS design and occurs as a block within the document flow. Within a Layout Module, multiple Layout Boxes, each containing standard SPS design elements, can be placed according to design requirements. Using Layout Modules, therefore, designers can create a layout just as they would using an artboard-based graphical design application.

The steps for creating a Layout Module are as follows:

1. Insert a Layout Container. The Layout Container can occupy the entire width of a page or can have any other dimensions you want. It can contain a blueprint of the design to serve as design guide and it can be formatted (in the Styles sidebar) using styles for the Layout Container.
2. Insert one or more Layout Boxes in the Layout Container. Layout Boxes can contain multiple design elements (including static text, schema nodes, Auto-Calculations, images, lists, etc), and they can be formatted (in the Styles sidebar) using styles for the Layout Box. Layout Boxes can also be moved relative to each other within the Layout Container and can be positioned in front of or behind each other.
3. Lines can be drawn, formatted, positioned and moved to the front or back of the stack of layout objects (Layout Boxes and other Lines).

Form-based designs

When you create a new SPS you are offered the choice of creating a free-flowing design or a form-based design. A form-based design is essentially an SPS design consisting of a Layout Container.

Note: Layout Modules are supported in Authentic View only in the Enterprise Editions of Altova products.

See also
- Creating the Design
- SPS File: Content
Layout Containers

A Layout Container has the following properties:

- It can be inserted within the flow of a document, that is, within a template. Or it can be inserted as the container within which the document design is created.
- It can have the same dimensions as the page dimensions defined for that section (the Auto-Fit to Page property of Layout Containers). Or it can have any other dimensions you specify. See the Layout Container size section below for details.
- A layout grid and a zoom feature make the positioning of objects in the Layout Container easier.
- It can have style properties, such as borders, background colors, font-properties for the whole container, etc.
- It can contain Layout Boxes and Lines, but no other design element. (All design elements must be placed within Layout Boxes.)
- It can contain a blueprint, which is an image placed on the artboard to serve as a reference template for the designer. The design can then be built to match the blueprint exactly.

Note: Layout Containers are supported in Authentic View only in the Enterprise Editions of Altova products.

Inserting a Layout Container

To insert a Layout Container, click the Insert Layout Container icon in the Insert Design Elements toolbar and click the location in the design where the Layout Container is to be inserted. A dialog appears asking whether you wish to auto-fit the Layout Container to the page. If you click Yes, the Layout Container will have the same size as the page dimensions defined in the page layout properties of that particular document section. If you click No, then a Layout Container with a default size of 3.5in x 5.0in is created.

Note that a Layout Container can also be created at the time you create an SPS.

Layout Container size

There are two sets of properties that affect the size of the Layout Container:

- The Auto-Fit Page Size property (Properties sidebar, screenshot below) can be set to yes to create a Layout Container having the same dimensions as those specified for pages in that document section. A value of no for this property creates a Layout Container with a customizable size.
The *height* and *width* properties of the Details group of Layout Container styles (in the Styles sidebar) specify the dimensions of the Layout Container. The dimensions can also be modified directly in the design by dragging the right and bottom margins of the Layout Container. Note that the *height* and *width* properties will take effect only when the *Auto-Fit Page Size* property has a value of *no*.

---

**Layout Container Grid**

The Layout Container has a grid to aid in spacing items in the layout. The following settings control usage of the grid:

- **Show/Hide Grid**: A toggle command in the Insert Design Elements toolbar switches the display of the grid on and off.
- **Grid Size**: In the Design tab of the Options dialog ([Tools] | Options) units for horizontal and vertical lengths can be specified. Note that if very large length units are selected, the grid might not be clearly visible.
- **Snap to Grid**: A toggle command in the Insert Design Elements toolbar enables or disables the Snap to Grid function. When the Snap to Grid feature is enabled, the top and left edges of Layout Boxes and the endpoints of Layout Lines align with grid lines and points, respectively.

---

**Zooming**

To help position objects more accurately, you can magnify the view. Do this by changing the Zoom factor in the Zoom combo box (in the Standard toolbar), or by pressing the Ctrl key and scrolling with the mouse.

---

**Layout Container style properties**

There are two types of style properties that can be applied to Layout Containers:

- Those applied to the Layout Container alone and which are not inheritable, such as the *border* and *background-color* properties.
- Those that are inheritable by the Layout Boxes in the Layout Container, such as font...
The style properties of a Layout Container are set in the Layout Container styles in the Styles sidebar (screenshot above).

Layout Container contents
The only design items that can be contained in a Layout Container are Layout Boxes and Lines. Additionally, a blueprint (which is not a design element) can be placed in the Layout Container as a design aid. All design elements must be placed in a Layout Box.

Blueprints
One blueprint can be placed in a Layout Container at a time to aid the designer in creating the SPS. The blueprint is an image file that can be placed to exactly fit the size of the Layout Container. Alternatively, if the blueprint image is smaller than the Layout Container, it can be offset to the desired location in the design (see Blueprint image properties screenshot below). The designer can use the blueprint by reproducing the SPS design over the blueprint design. In this way the designer will be able to place design elements in the layout exactly as in the blueprint. The blueprint will appear only in Design View, but not in any output view: this is because its purpose is only to aid in the design of the SPS.
The blueprint's properties can be controlled via the *Blueprint image* group of properties of the Layout Container properties (in the Properties sidebar, *screenshot below*).

The opacity of the blueprint in the Layout Container can be specified so that the blueprint does not interfere with the viewing of the design. The display of the blueprint image can also be switched off with the *Show Image* property if required.

- **See also**
  - [Layout Boxes](#)
  - [Lines](#)
Layout Boxes

Every design element in a layout (such as static text, schema nodes, Auto-Calculations, images, lists, etc) must be placed in a Layout Box. The Layout Boxes containing design elements are laid out as required in the Layout Container. Note that a design element cannot be placed directly in a Layout Container; it must be placed in a Layout Box.

This section describes how Layout Boxes are used and is organized into the following subsections:

- Inserting Layout Boxes
- Selecting and moving Layout Boxes
- Modifying the size of the Layout Box
- Defining Layout Box style properties
- Inserting content in the Layout Box
- Stacking order of Layout Boxes

---

Inserting a Layout Box

A Layout Box can be inserted only in a Layout Container. To add a Layout Box, first click the Insert Layout Box icon in the Insert Design Elements toolbar, then click on the location inside the Layout Container where you wish to insert the Layout Box. A Layout Box will be inserted, with its top left corner positioned at the point where you clicked. The box will be transparent, will have no borders, and will have default text.

---

Selecting and moving a Layout Box

To select a Layout Box, place the cursor over the left border or top border of the Layout Box so that the cursor becomes a crossed double arrow. When this happens, click to select the Layout Box. If you keep the mouse button depressed, you can move the Layout Box to another location within its Layout Container. You can also move a Layout Box left, right, up, or down by selecting it, and then pressing the cursor key for the required direction. When the Layout Box is selected, its properties and styles are displayed in the respective sidebars.

---

Layout Box size

Each Layout Box has a property called Auto-Resize (see screenshot below). When the value of this property is set to yes, the Layout Box automatically resizes to exactly accommodate any static content (including markup) that is inserted in it in the Design View. When the value of Auto-Resize is set to no, the size of the Layout Box does not automatically change when content is inserted in it.

To change the size of the Layout Box manually, drag its right border and bottom border. You can also change the size of a Layout Box by using the cursor keys to move the right and bottom borders of the box. To do this first select the Layout Box. Then do the following: (i) to move the right border, keep the Shift key depressed and press the right or left cursor key till the required
size is obtained; (ii) to move the bottom border, keep the Shift key depressed and press the top or bottom cursor key.

The Additional Height and Additional Width properties give the lengths that are additional to the optimal dimensions as determined by auto-resizing. The additional lengths are obtained when a Layout Box is manually resized. Conversely, by changing the values of these two properties, the size of the Layout Box can be changed.

**Note:** In a Layout Box a linefeed is obtained by pressing the Enter key. This is significant, because if content is added that does not contain a linefeed, then the length of the current line increases, thus increasing the optimal width of the Layout Box and—incidentally—affecting the Additional Width value, which is calculated with reference to the optimal width.

**Layout Box style properties**
The style properties of a Layout Box are set in the Layout Box styles in the Styles sidebar (screenshot below). The styles are displayed when the Layout Box is selected, and can then be edited.
Note: The background-color value of transparent can be selected in the dropdown list of the property's combo box (it is not available in the color palette). The significance of this value in a situation where the Layout Box is part of a stack is explained below.

Inserting content in a Layout Box
Any type of design element can be inserted in a Layout Box, and is inserted just as it normally would be in an SPS. Note, however, that neither a Layout Container nor a Layout Line can be inserted in a Layout Box. The following points should be noted:

- When design elements are inserted that require a context node, the current node will be taken as the context node. The current node is the node within which the Layout Module has been created.
- Text content in a layout box can be rotated 90 degrees clockwise or anti-clockwise, so that the text is vertical, reading from top-to-bottom or bottom-to-top, respectively. To do this, in the design, select the text that is to be rotated and, in the Properties sidebar (screenshot below), select LayoutBox. In the Layout Box group of properties, select the required value for the Orientation property.
Note the following points:

- The rotation will be applied to the output, but will not be displayed in the design.
- This property can also be applied to text in table cells.

---

**Stacking order of Layout Boxes**

Layout Boxes can be placed one over the other. When one Layout Box is placed on top of another, then, if it is opaque, it hides that part of the Layout Box which it covers. This behavior can be extended to a stack of several Layout Boxes. In such a stack, only the topmost Layout Box will be fully visible; the others will be partially or fully covered.

![Stacking order of Layout Boxes](image)

Layout Boxes can be sent backward or brought forward using the **Order** menu commands in the context menu of the selected Layout Box. Using these commands a Layout Box can be ordered:

(i) relative to its nearest neighbor on the stack (the **Bring Forward** and **Send Backward** commands), or
(ii) relative to the entire stack (the **Bring to Front** and **Send to Back** commands).

In the screenshot above, the stacking order from front to back is as follows:

- **Left stack**: orange, green, blue
- **Right stack**: blue, green, orange

Note that Layout Boxes with transparent backgrounds (the default background of Layout Boxes) might appear to not move relative to each other, especially if more than one box in the stack is transparent and if boxes have no borders. The screenshot below presents some ways in which transparency affects stacking.

![Transparency and stacking](image)

**Note**: **Layout Lines** can also be added to a stack of Layout Boxes, and each Line can be moved relative to other items in the stack.
See also

- Layout Containers
- Lines
Lines
Lines can be inserted in a Layout Container (but not in Layout Boxes), then selected, re-sized and moved around within the Layout Container, assigned properties, and moved backwards and forwards in a stack of layout items consisting of Layout Boxes and other Lines.

Inserting a Line
To add a Line to a Layout Container, do the following:
1. Click the Insert Line icon in the Insert Design Elements toolbar.
2. Click on the location inside the Layout Container where you wish to locate the start point of the line.
3. Without releasing the mouse button, draw the line from the start point to the desired end point. Then release the mouse button.
A black line will be inserted, with a dot at each end indicating the start and end points respectively.

Selecting, moving, and sizing a Line
In the Main Window, you can carry out the following-drag-and-drop functions:

- To select a Line, click any part of the Line (the cursor becomes a crossed double arrow when it is over the Line). Once a Line is selected, its properties are displayed in the Properties sidebar and can be edited there (see below).
- To move a Line, select it and drag it to the desired location. You can also move a line left, right, up, or down by selecting it, and then pressing the cursor key for the required direction.
- To graphically re-size or re-orient a Line, select either the start point or end point and re-position it to obtain a new size and/or orientation. You can also re-size or re-orient a Line by pressing Shift and the cursor keys: the right and left cursor keys move the right-hand endpoint right and left, the up and down cursor keys move the right-hand endpoint up and down, respectively.

Line properties
When a Line is selected its properties are displayed in the Properties sidebar (screenshot below), and the properties (listed below) can be edited in the sidebar. You can also right-click a Line to pop up the Properties sidebar with the properties of the Line in it.
The following Line properties can be edited in the Properties sidebar:

- **Color**: Specifies a color for the Line. The default is black.
- **Size and position**: The location of the start and end points of the Line can be specified using an x-y (horizontal-vertical) coordinate system. The reference frame is created with the top left corner of the Layout Container having the coordinates \((x=0, y=0)\).
- **Width**: Specifies the thickness of the Line.

---

**Lines and stacking order**

When a Line is in a stack consisting of Layout Boxes and other Lines, it can be sent backward or brought forward using the **Order** menu commands in the context menu of the selected Line. Using these commands a Line can be ordered: (i) relative to its nearest neighbor on the stack (the **Bring Forward** and **Send Backward** commands), or (ii) relative to the entire stack (the **Bring to Front** and **Send to Back** commands).

In the screenshot above, the stacking order from front to back is as follows: green box, red line, black line, blue box.
• Layout Containers
• Layout Boxes
8.12 The Change-To Feature

The Change-To feature is available when a template or the contents of a template are selected, and enables you to change: (i) the node for which that template applies, or (ii) how the node is created in the design.

What can be changed with the Change-To feature

Either a node or its contents can be changed. In the image below left, the node is selected. In the image at right, the node's contents are selected.

The n1:Name element in the screenshot above has been created as (contents), and so the node's contents are represented by the (contents) placeholder. Alternatively, the node could have been created as another type of content, for example, as an input field or combo box. Other types of content can also be selected.

The Change-To command

Access the change to command by right-clicking your selection. In the context menu that pops up, select Change To (screenshot below).
Changing template matches

If a template is selected, you can change the node for which that template applies. This is useful if, for instance, the name of an element has been changed in the schema. When you mouse over the Change To command and select Template from the sub-menu that pops up, you are presented with a list off all the nodes that may be inserted as a child of the selected node’s parent element. Click one of these nodes to make the template apply to that node.
If the selected node has a content model that does not match that described in the template, there will be structural inconsistencies. Such inconsistencies are errors and are indicated with red strikethroughs in the tags of nodes that are invalid.

You can also change the template-match to match, not a node, but a variable template.

Changing the content type of the node
If a template or its contents are selected, then you can change the type of content the node is created as. On hovering over the Change To command in the context menu, the type of content that the selected node can be changed to is displayed as options in the sub-menu that pops up (screenshot below).
The screenshot above has been taken with a combo box selected.

- **See also**
  - SPS File Content
  - Templates and Design Fragments
Chapter 9

SPS File: Structure
9 SPS File: Structure

The structure of an SPS document is both input- as well as output-driven, and it is controlled by:

- Schema sources
- Modular SPSs
- Templates and Design Fragments

**Input-driven structure: schemas and modular SPS files**

By input-driven, we mean that the source schemas of SPS files specify the structure of the input document/s and that this structure is the structure on which the SPS document is based. For example, if a source schema specifies a structure that is a sequence of Office elements, then SPS design could have a template for the Office element. At processing time this template will be applied in turn to each Office element in the source data document.

Another example of how the source document structure drives the design of the SPS file can be seen in the use of tables. Say that an Office element contains multiple Person element children, and that each Person element contains a set of child elements such as Name, Address, Telephone, etc. Then a template in the form of a table can be created for the Person element. Each Person element can be presented in a separate row of the table (*screenshot below*), in which the columns are the details of the Person (the child elements of the Person element).

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loby</td>
<td>Maise</td>
<td>Accounting Manager</td>
</tr>
<tr>
<td>Frank</td>
<td>Further</td>
<td>Accounts Receivable</td>
</tr>
<tr>
<td>Vernon</td>
<td>Callby</td>
<td>Office Manager</td>
</tr>
</tbody>
</table>

Such a template is possible because of the structure of the Person element and because the Person elements are siblings. In the table template a single row is designed for the Person element, and this processing (the row design) is applied in turn to each Person element in the source document, creating a new row for each Person element, with the child elements forming the columns of the table.

How to use various kinds of schema sources is described in the section, Schema Sources.

Additionally, StyleVision allows SPSs to be re-used as modules within other SPSs. In this way, modules can be included within a structure and can modify it. However, a schema structure contained in a module must fit in with the structure of the underlying schema of the containing SPS. How to work with modular SPSs is described in the section, Modular SPSs.

**Output-driven structure: templates and design fragments**

While the schema sources provide the structure of the input data document, the actual design of
the output document is what is specified in the SPS document. This design is contained in one document template called the main template. The main template typically contains several component templates and can reference global templates. Templates are described in the section, *Templates and Design Fragments*.

This composability (of multiple templates) is further enhanced by a StyleVision feature called Design Fragments, which enables specific processing to be assigned to a document fragment that can be re-used. A Design Fragment is different than a global template in that: (i) it can be composed of multiple templates; and (ii) identical content with different processing can be created in separate design fragments, either of which can be used in a template according to the situation. For example, in some processing situations, an `Email` node might be required as a link that opens an empty email; in other cases the `Email` element could be required in bold and in red. Two separate design fragments could provide the respective processing, and both can be re-used as required.

Design fragments are described in detail in the section, *Design Fragments*.

- **See also**
  - Usage Overview
9.1 Schema Sources

The schema sources are the starting point of the design, and design structure can be influenced by: (i) choices you make during schema selection, and (ii) the root elements you select in the schema.

Schema selection

The selection of the schema for a new SPS file can be done in the following ways:

1. Click File | New and directly select a schema source to add via one of the methods (except New (empty)) available in the menu that pops up.

2. Click File | New, select New (empty) from the menu that pops up. After the new SPS is created and displayed in the GUI, in the Design Overview sidebar, select the Add New Schema command. This pops up a menu listing the methods you can use to add different types of schemas (screenshot below). Each command in this menu is described in the sub-sections of this section.

The schema source can be selected from a file, from a DB, or be an XBRL taxonomy or user-defined. An important point to consider is whether you will be using global templates, and whether elements you wish to create as global templates are defined as global elements in the schema.

When adding a DTD from file, remember that all elements defined in the DTD are global elements. When adding an XML Schema from file, it is worth checking what elements are defined as global elements and, should you wish to make any change to the schema, whether this is permitted in your XML environment. When a DB is selected, during the import process, you can select what tables from the DB to import. This selection determines the structure of the XML Schema that will be generated from the DB.

Note: You can select multiple schemas, of which one schema is designated the main schema (right-click the schema in the Design Overview sidebar and select Set As Main Schema). The document node of the main schema is the entry point of the main template. The selection and presence of subsidiary schemas in the schema sources enable nodes from these schemas to be included in the design.

Note: If you wish to add a namespace to an SPS or to an XSLT stylesheet being generated from an SPS, the namespace must be added to the top-level schema element of the XML Schema on which the SPS is based.

Root elements
If a schema source has multiple global elements, then multiple root elements (document elements) can be selected for use in the design. This enables the SPS design to have templates that match multiple document elements. The advantage of this is that if an SPS, say UniversalSPS.sps, based on UniversalSchema.xsd has one template each for its two root elements, Element-A and Element-B, then this one SPS can be used with an XML instance document which has Element-A as its document element as well as with another XML instance document which has Element-B as its document element. For each XML instance, the relevant template is used, while the other is not used. This is because for the document element of each XML instance document, there is only one template in the SPS which matches that document element. For example, the document element /Element-A will be matched by the template which selects /Element-A but not by that which selects /Element-B. In this connection, it is important to remember that if multiple global elements are defined in the schema, an XML document with any one of these global elements as its document element is valid (assuming of course that its substructure is valid according to the schema).

To set up the SPS to use multiple root elements (document elements), click the ▼ button to the right of the /Root elements entry of the schema. The following dialog pops up.

The dialog lists all the global elements in the schema. Select the global elements that you wish to use as root elements (document elements) and click OK. The selected element/s will be available as root document elements and will be displayed in the Root Elements list. A template can now be created for each of these document elements. Each of these templates serves as an alternative root element template. When an XML document is processed with this SPS, only one of the alternative root element templates will be used: the one that matches the root (or document) element of the XML document.

So, when an XML document having Element-A as its document element is processed with this SPS, then the root template in the SPS that matches Element-A is triggered, while all the other root element templates in the SPS are ignored. If an XML document having Element-B as its document element is processed, then the root template in the SPS that matches Element-B is triggered, while all other root element templates in the SPS are ignored. In this way a single SPS
can be used to process two or more XML documents, each of which has a different root (or document) element.

- **See also**
  - [Schema Tree sidebar](#)
DTDs and XML Schemas

An SPS can be based on an XML Schema or DTD or have one or more XML Schemas or DTDs among its schema sources. An XML Schema or DTD can be created as a schema source in one of the following ways:

- The XML Schema or DTD is created as a schema source directly when the SPS is created (File | New | New from XML Schema / DTD / XML).
- The XML Schema or DTD is added to an empty SPS or to an SPS with existing schema sources (in the Design Overview sidebar).

The respective commands prompt you to browse for the XML Schema or DTD. If the schema is valid, it is created as a schema source in the Schema Sources tree of the Schema Tree sidebar. Alternatively, an XML file can be selected. If an XML Schema (`.xsd`) or DTD file is associated with the XML file, then the XML Schema or DTD file is loaded as the source schema and the XML file is loaded as the Working XML File. If no schema is associated with the XML file, a dialog pops up asking whether you wish to generate an XML Schema based on the structure and contents of the XML file or browse for an existing schema. If you choose to generate a schema, the generated schema will be loaded as the source schema, and the XML file will be loaded as the Working XML File.

---

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:
  - Checked in. Available for check-out.
  - Checked out by another user. Not available for check-out.
  - Checked out locally. Can be edited and checked-in.

- 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.

---

The **anyType** datatype of XML Schema

If an element in the XML Schema has been assigned the **anyType** datatype of XML Schema or if it has not been assigned any datatype, then the schema tree in the Schema Tree will show this element as having all the global elements of that schema as possible children. For example, if an element called **email** has not been assigned any datatype, then it will be displayed in the schema tree with all global elements as possible children, such as, for example: **person**, **address**, **city**, **tel**, etc. To avoid this, assign the **email** element a datatype such as **xs:string**.

---

**See also**

- Schema Tree sidebar
- Multiple Schema Sources
**DB Schemas**

An SPS can be based on a schema that is generated from a DB or have a DB-based schema (DB schema) as one of its schema sources. A DB schema can be created as a schema source in one of the following ways:

- The DB schema is generated for a new SPS when the SPS is created directly from a DB (File | New | New from DB or File | New | New from XML Column in DB Table).
- The DB schema is added to a new empty SPS that has no schema sources or to an SPS with existing non-DB schema sources (in the Design Overview sidebar).

The respective DB-schema-generation commands generate a temporary XML Schema from the selected DB and creates the schema as a schema source in the **Schema Tree sidebar**, or the command loads a DB-based schema (DB schema) from an XML DB. An element extraneous to the DB, called **DB**, is created as the document element, and the DB structure is created within this document element. During the schema creation process, you will be prompted to select which tables or data from the database you wish to import. These database tables will be created in the XML Schema as children of the **DB** element and also as items in the Global Templates list.

---

**Creating the XML Schema from a DB**

Creating the XML Schema from a DB consists of two steps:

- **Connecting to a Database.** This consists essentially of browsing for the DB file (if it is a MS Access DB), or building a connection string (all other DBs except MS Access).
- **DB Data Selection.** In this step, the table or data structure from the database is selected. In the case of non-XML DBs, the temporary XML Schema that is generated and the XML file that is created will be based on the selected data tables.

After the schema source appears in the Schema Tree window, you can start designing the SPS. A temporary XML File is created each time an output preview tab is clicked. The XML file is structured according to the generated XML Schema and contains data from the selected DB tables.

---

**See also**

- Schema Tree sidebar
- Multiple Schema Sources
- Connecting to a Database
- DB Data Selection
**XBRL Taxonomy**

An XBRL taxonomy can be used as a schema source in the following ways:

- A new SPS file can be created with an XBRL taxonomy as its schema source via the **File | New | New from XBRL Taxonomy**.
- In the Design Overview sidebar, clicking the Add Schema Source link pops up a menu in which you select the option **Add XBRL Taxonomy**. The taxonomy can be added to an empty SPS file (that is, one with no schema source) or to one already containing one or more schema sources.

On clicking either of these two commands, a file open dialog pops up, in which you can browse for the XBRL taxonomy. The taxonomy can also be selected via a URL or a global resource; how to do this is described below. Select the taxonomy and click **Open**. A dialog appears asking whether you wish to assign a Working XBRL file to the schema source. Browse for the file or global resource, or select **Skip** to not assign a Working XBRL File. (You will need a Working XBRL File in order to preview your output in the Output Views.)

The XBRL taxonomy will be loaded and will be available in the Schema Tree sidebar for dragging into the design. The Working XBRL file will be loaded and will provide the data for output previews.

---

**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:

  - Checked in. Available for check-out.
  - Checked out by another user. Not available for check-out.
  - Checked out locally. Can be edited and checked-in.

- 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.

**See also**
- Schema Tree sidebar
- Design Overview sidebar
- Multiple Schema Sources
User-Defined Schemas

You can quickly create a user-defined schema in the Schema Tree sidebar. This is useful if you have an XML document that is not based on any schema and you wish to create an SPS for this XML document.

To add and create a user-defined schema, in the Schema Tree sidebar, do the following:

1. In the Design Overview sidebar, click the Add New Source command (under the Sources heading), and select Add User-Defined Schema (screenshot below). The new schema is created and is indicated with the parameter $USER (screenshot below).

2. In the Root Elements tree, there is a single root element (document element) called UserRoot.

3. Double-click UserRoot and rename it to match the document element of the XML document for which you are building this schema.

4. To assign a child element or an attribute to the document element, select the document element (UserRoot), and click, respectively, (i) the drop-Append New Element icon in the toolbar of the Schema Tree sidebar; and (ii) the dropdown arrow of the Append New Element icon | the Append New Attribute command. Alternatively, you can right-click and select the required command from the context menu. When an element is selected, appending and inserting an element, adds the new element as a sibling element, respectively, after and before the selected element. You can also add a child element and a child attribute. When an attribute is selected, you can append or insert another attribute, respectively, after and before the selected attribute. After the new element or attribute is added to the tree, type in the desired name. You can also drag nodes to the desired location (described in the next step). In the screenshot below, the Article element is the document element. The elements Title, Para, Bold, and Italic, and the attributes ID and Author have been added at the child level of Article.
5. To move the elements **Bold** and **Italic**, and the attribute **ID** to the level of children of **Para**, select each individually and drag to the **Para** element. When a bent downward-pointing arrow \( \downarrow \) appears, drop the dragged node. It will be created as a "child" of **Para** (screenshot below).

6. When any element other than the document element is selected, adding a new element or attribute adds the new node at the same level as the selected element. Drag a node (element or attribute) into an element node to create it as a "child" of the element node.
Editing node names and deleting nodes
To edit the name of an element or attribute, double-click in the name and edit the name. To delete a node, select it and click the Remove icon \(\times\) in the toolbar. Alternatively, select Remove from the context menu.

- See also
  - Schema Tree sidebar
  - Design Overview sidebar
  - Multiple Schema Sources
Multiple Schema Sources

You can use multiple schema sources in your SPS. This enables you to combine data from different XML documents in the output. For example, in one XML document you can have the structure of an organization, with its various departments. In another XML document you can have the listing of personnel in the various departments. In the output, the organizational structure data (first XML document) and the personnel data (second XML document) can be combined.

Note: If, across the multiple schemas used, there are two or more global elements that have the same expanded QName, there will be a naming conflict and one or more of the involved schema documents will be rendered invalid. You must therefore ensure that no two global elements across the multiple schemas have the same expanded QName. Local elements across multiple schemas may have the same expanded QName as other local elements, and will not cause any naming conflict.

Adding multiple schemas

In order to combine the dynamic XML data from different documents, you first add to the SPS the different schemas that are involved. You do this in the Design Overview sidebar (screenshot below). This creates a tree for each schema in the Schema Tree sidebar (screenshot below). You can choose the Working XML File, Template XML File, and Root elements for each added schema (see screenshot below and the topic Design Overview sidebar). The global elements in all the added schemas are listed in the All Global Elements tree of the Schema Tree sidebar, and are available for creation as global templates.

Note that XML signatures can only be enabled for the main source.
Each schema source is identified by a parameter of the form $XML1 (DTD or XML Schemas) or
$DB1 (DB-based schema) or $USER1 (user-defined schema). These three kinds of schemas can be
combined in a single SPS (see screenshot below). Note, however, that, while any number of
DTDs and/or XML Schemas can be added to an SPS, only one each of a DB-based schema and
a user-defined schema can be added.

Selecting the main schema
Of all the schemas added to the SPS, one can be designated as the main schema, and, in the
Schema Tree sidebar, and in both the Design Overview and Schema Tree sidebars, this is
indicated with the the word (main) next to the main schema. The document node of the XML
document corresponding to the main schema will be the starting point of the transformation. To
select the main schema, right-click the schema in the Design Overview sidebar, and select Set
as Main Schema.

Adding nodes from different schemas to the design
Within the main template, nodes from different schemas are added to the design by dragging the nodes from their respective schema trees into the main template. A global template is created in the same way as with a single schema. It can be copied or used by a local template, or used instead of the built-in template when invoked via a rest-of-contents instruction from the main template.

---

**Using schema parameters in external transformations**

Each schema added to the SPS is identified by a parameter. For example, $XML (for DTDs or XML Schemas) or $DB (DB-based schema) or $USER (user-defined schema). In StyleVision, the Working XML File that is associated with each schema is known to the application, so the correct document nodes are accessed, thus enabling the display of data from the XML sources in Authentic View and output previews. However, when you generate XSLT stylesheets from the SPS, no XML file associations are contained in the stylesheet. In order to associate an XML file with each schema in the XSLT stylesheet, you must pass the location of each XML file as the value of the corresponding schema parameter. For example, the schema parameter, $XML, according to the screenshot above, would have a value of "[path]NanonullOrg.xml".

**Note:** The following points should be noted.

- For DB-based schemas, it is the the XML file generated from the DB (via File | Save Generated Files) that should be passed as the value of the $DB parameter.
- Any value passed for the main schema parameter will be ignored. The XSLT stylesheet will use as its starting point the document node of the XML document that is being transformed.

---

- **See also**
- Schema Tree sidebar
9.2 Merging XML Data from Multiple Sources

XML data from multiple source XML files can be merged when XSLT 2.0 or 3.0 is used as the XSLT version of the SPS.

Typically, the merging of data will be based on a common piece of data, such as an ID. For example, an employee in a company, who is identified by a personal ID number, can have his or her personal data stored in multiple XML files held by the personnel department: (i) personal details, (ii) payroll, (iii) work and leave, (iv) courses attended, etc. Data from these different files can be merged in a single output document using the personal ID number as a key.

Note: The Enterprise Edition enables you to include multiple schema sources, so XML nodes from other schemas can be selected using the parameter name for the corresponding schema (as is the case in the example below). In the Professional and Basic Editions, the doc() function of XPath 2.0 can be used to locate the required XML file and the XML node within that file. The doc() function of XPath 2.0 provides access to the document root of external XML documents, and thus enables node content from external XML documents to be inserted in the output. An Auto-Calculation that uses the doc() function can, therefore, also be used to merge XML data (see example below).

Example

The (My) Documents folder, C: \Documents and Settings\<username>\My Documents\Altova \StyleVision2015\StyleVisionExamples, contains an example SPS file (MergeData_2_Files.sps) that shows how data from different source XML files can be merged. The SPS selects data from an order (MergeOrder.xml, listed below) that a fictitious customer places.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="MergeOrder.xsd">
    <Item partNum="238-KK" quantity="3" shipDate="2000-01-07" comment="With no
    inclusions, please."/>
    <Item partNum="748-OT" quantity="1" shipDate="2000-02-14"
    comment="Valentine's day packaging."/>
    <Item partNum="229-OB" quantity="1" shipDate="1999-12-05"/>
    <Item partNum="833-AA" quantity="2" shipDate="1999-12-05" comment="Need this
    for the holidays!"/>
</Order>
```

The value of the /Order/Item/@partNum attribute in this file (see above) is used to select the ordered products from the catalog of articles stored in another file, MergeArticles.xml (see listing below).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Articles xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="MergeArticles.xsd">
    <Article PartNum="833-AA">
        <ProductName>Lapis necklace</ProductName>
        <Price>99.95</Price>
    </Article>
```

The Enterprise Edition enables you to include multiple schema sources, so XML nodes from other schemas can be selected using the parameter name for the corresponding schema (as is the case in the example below). In the Professional and Basic Editions, the doc() function of XPath 2.0 can be used to locate the required XML file and the XML node within that file. The doc() function of XPath 2.0 provides access to the document root of external XML documents, and thus enables node content from external XML documents to be inserted in the output. An Auto-Calculation that uses the doc() function can, therefore, also be used to merge XML data (see example below).
<Article PartNum="748-OT">
  <ProductName>Diamond heart</ProductName>
  <Price>248.90</Price>
</Article>

<Article PartNum="783-KL">
  <ProductName>Uncut diamond</ProductName>
  <Price>79.90</Price>
</Article>

<Article PartNum="238-KK">
  <ProductName>Amber ring</ProductName>
  <Price>89.90</Price>
</Article>

<Article PartNum="229-OB">
  <ProductName>Pearl necklace</ProductName>
  <Price>4879.00</Price>
</Article>

<Article PartNum="128-UL">
  <ProductName>Jade earring</ProductName>
  <Price>179.90</Price>
</Article>

...  

</Articles>

The way the merging of the data is done is to set up a user-defined template within the /Order/Item template (see screenshot below) that selects the corresponding Article element in the MergeArticles.xml file by using the part number of the ordered item to identify the article. The XPath expression (which is in the /Order/Item context) is: $Articles//Article[@PartNum=current()/@partNum]

This template will produce output something like that shown in the screenshot below.

You have ordered:

3 Amber ring
1 Diamond heart
1 Pearl necklace
2 Lapis necklace

Notice that while the quantity ordered of each item is taken from the file MergeOrder.xml, the name of the ordered article is taken from the file MergeArticles.xml. Also notice how the ProductName node is selected within the context of the /Articles/Article template.
The same result as that obtained above could also be achieved using an Auto-Calculation (see screenshot below). Drag the quantity attribute from the Schema Tree window and create it as contents. Then add an Auto-Calculation as shown in the screenshot and give the Auto-Calculation an XPath expression as described below.

```
You have ordered: $p

<Order>
   <Item>
      <div>
         <quantity>(content)<quantity> = (AutoCalc)$div
      </div>
   </Item>
</Order>
```

The XPath expression of the Auto-Calculation could target the required node using either the parameter of another schema source or the `doc()` function:

```
$Articles//Article[@PartNum=current()/@partNum]/ProductName
```

or

```
doc('MergeArticles.xml')//Article[@PartNum=current()/@partNum]/ProductName
```

Notice that, while the first XPath expression above uses a parameter to refer to another XML Schema (a feature available only in the Enterprise Edition), the second expression uses the `doc()` function of XPath 2.0 (a feature available in the Professional and Basic editions as well).

- See also
  - Auto-Calculations
  - Multiple Schema Sources
9.3 Modular SPSs

The global templates of an SPS, as well as Design Fragments, JavaScript functions, and page layout items can be used in the design of another SPS. This enables:

1. The re-use of global templates and other components across multiple SPSs, the main advantages of which are single-source editing and consistency of output.
2. SPSs to be modularized, and thus to be more flexibly structured.

In any given SPS, one or more SPSs can be added as modules. Some types of components (or objects) in these modules are then available to the importing (or referring) SPS.

---

Available module objects
The section, Available Module Objects, not only describes the extent to which, and conditions under which, the various components of an SPS are available to an importing SPS. It also lists those components that are not available to the importing SPS. You should note that if an added module itself contains modules, then these are added recursively to the referring SPS. In this way, modularization can be extended to several levels and across a broad design structure.

---

Creating a modular SPS
To build a modularized SPS, first add the required SPS to the main SPS as a module. All the JavaScript functions, global templates, Design Fragments, and XPath functions in the added module are available to the referring SPS. Each of the available objects is listed in the Design Tree, under its respective heading (screenshot below), and can be activated or deactivated, respectively, by checking or unchecking its check box.
These objects can then be re-used in the referring SPS according to their respective inclusion mechanisms. Global templates typically would need merely to be activated in order for them to be applied in the referring SPS. Design fragments have to be dragged from the Design Tree to the required location. JavaScript functions are assigned via the Property window as event handlers for the selected design component. And available (activated) XPath functions can be used in XPath expressions.

How to create and work with a modular SPS is described in the section, Creating a Modular SPS.

Terminology

When an SPS is used within another module it is said to be added to the latter, and we call the process adding. The two SPSs are referred to, respectively, as the added SPS module and the referring SPS module. When an SPS module is added, its objects are added to the referring SPS module. These objects are called module objects, and are of the following types: global templates; Design Fragments; JavaScript functions; and page layout items.

See also
- Design Overview
- Schema Sources
• Templates and Design Fragments
Available Module Objects

This section lists the objects in added SPS modules that are available to the referring SPS module. The listing explains in what way each object is available to the referring SPS module and how it can be used there. For a step-by-step approach to creating modular SPSs, see the next section, Creating a Modular SPS. The section ends with a list of objects in the added SPS that are not available to the referring SPS module; this will help you to better understand how modular SPSs work.

- Namespace declarations
- Global templates
- Design fragments
- Added modules
- Scripts
- CSS styles
- Page layouts
- Unavailable module objects

Namespace declarations

Each SPS stores a list of namespace URIs and their prefixes. When an SPS is added as a module, the namespaces in it are compared to the namespaces in the schema source/s of the referring SPS. If a namespace URI in the added SPS matches a namespace URI in the schema source/s of the referring SPS, then the prefix used in the schema source of the referring SPS is adopted as the prefix for that namespace in the added SPS. If a namespace URI in the added SPS cannot be matched with any namespace URI in the schema source/s of the referring SPS, then an error message indicating this is displayed.

The screenshot above shows the various namespaces in an SPS, together with their prefixes, in the Schema Tree sidebar. These namespaces come from the source schema/s and cannot be
Global templates
The global templates of the added SPS module are available to the referring SPS module and are displayed in the Design Tree sidebar (screenshot below). They are, by default, activated or deactivated (checked or unchecked), according to the respective activation status in the added module. If you wish to create a global template to override a global template from an added module, create the new global template by clicking the icon next to the Global Templates entry. In the Add New Global Template that pops up, select an element or attribute for which you wish to create the global template. Alternatively, enter an XPath expression that selects the required node in the schema. On clicking OK, you will be prompted as to whether the new global template should be activated in preference to the global template in the added module. The response you select activates either the newly created global template or the global template in the added module. You can switch your selection at any time by checking the other of the two global templates.

Note that the main template of added modules are not available. This means that if you plan to re-use a template via the modular approach, you must create it as a global template. If no global template is defined for a particular element and processing is invoked for that element, then the default processing for that element (XSLT's built-in templates) will be used.
Design fragments
Design fragments in the added SPS module are available to the referring SPS and are displayed in the Design Tree sidebar (screenshot above). When inserting a design fragment in the design, care should be taken to place the design fragment within the correct context node in the design.

Added modules
Each added SPS module also makes available to the referring SPS its own added modules, and their added modules, and so on. In this way, adding one module recursively makes available all modules that have been added to it, down multiple levels. Needless to say, these modules must together construct a content model that is valid according to the source schema/s of the referring SPS module. Modules are displayed and can be managed in the Design Overview sidebar.

Scripts
The scripts in all the added SPS modules are available for use in the referring SPS and are displayed in the Design Tree sidebar. In effect, the scripts of all the added modules are collected in a library that is now—in the referring SPS—available for selection in the Properties dialog.

CSS styles
The global styles present in added SPS modules are carried over to the referring SPS as global styles and the style rules are displayed in the Style Repository sidebar. The CSS files are also listed in the Design Overview sidebar. Similarly, external CSS files that were available to the added SPS module, are available to the referring SPS module.

Page layouts
The page layouts of an added module are available to the referring SPS and are displayed in the Design Tree sidebar.

Module objects that are not available to the referring SPS
The following objects of the added module are not available to the referring SPS:

- **Parameter definitions**: are ignored.
- **Schema sources**: The schema source on which the added SPS is based is ignored.

Bear in mind that the content model of the document element of the added SPS must be
contained within the content model of the referring SPS; otherwise it would not be possible to correctly use the added SPS as a module. If you wish, you could always add a secondary schema or a user-defined schema to the referring SPS. The additional schema could accommodate the content model of the added global template/s.

- **Working XML File and Template XML File**: References to these files are ignored. The referring SPS uses its own Working XML and Template XML Files.
- **XPath default namespaces**: If they have been set on a module that is imported then they are not carried through to the importing SPS.

See also
- Design Overview
- Creating a Modular SPS
- Schema Sources
- Templates and Design Fragments
- Using Scripts
Creating a Modular SPS

Creating a modular SPS consists of four broad parts:

1. Design and save the SPS module to be added.
2. Add the module to the SPS in which it is to be used (that is, to the referring SPS module).
3. Activate or deactivate the added object/s as required.
4. Apply the required object wherever required.

The SPS module to be added

There are two points to bear in mind when creating an SPS that will be added to another:

1. The templates that can be used in the referring SPS module can only be global templates. This means that the templates you wish to re-use must be created as global templates in the SPS module that is to be added.
2. The document structure defined in the SPS module to be added must be valid within the content model defined by the source schema/s of the referring SPS. If an added template is not contained in the content model defined by the main schema of the SPS, its content model, however, can still be defined in a secondary schema or a user-defined schema.

When creating the SPS module to be added, the schema on which you base the SPS could be one of the following:

- The main source schema of the referring SPS. In this case, when the SPS is added, the added global templates will be part of the content model of the referring SPS's main schema. The output of these global templates in Authentic View is, therefore, editable.
- A schema which defines a content model that is part of the content model defined by the main schema of the referring SPS. In this case, when the global templates are added, they will fit into the content model of the main schema of the referring SPS. The output of these global templates is editable in Authentic View.
- A schema which defines a content model that is not part of the content model defined by the main schema of the referring SPS. When this SPS module is added, its global templates will not be part of the content model of the main schema of the referring SPS. They can, however, be used to produce output if a secondary schema or a user-defined schema is used that defines a content model that contains the content model of the global template/s. In Authentic View, however, the output of these global templates cannot be edited.

When defining the content models in your schemas, you should pay close attention to the namespaces used since these determine the expanded names of nodes.

You could use a Working XML File to test the output of the SPS module to be added. The reference to this Working XML File will be ignored by the referring SPS.

Adding the SPS module

To add a module to an SPS, in the Design Overview (screenshot below), click the Add New
Module command, browse for the required SPS file in the dialog that appears, select it, and click **Open**.

The module is added to the SPS and is listed under the Modules heading in the Design Overview. In the screenshot above, the **BusinessAddressBook.sps** and **PersonalAddressBook.sps** modules have been added to the **AddressBook.sps** module (the active SPS). All the added module objects are listed in the Design Tree sidebar; added CSS files, though, are also also listed in the Design Overview. If the added modules themselves refer to modules, these latter, indirectly imported modules are listed under the Modules heading, but in gray. Information about which modules import an indirectly imported module is available in a pop-up that appears when you mouseover the indirectly imported module.

To open one of the added modules or indirectly imported modules quickly in StyleVision, right-click that module, and select **Open Defining Module** from the context menu that pops up.

**Order of added modules**

The order in which modules are added and listed is significant for the prioritizing of CSS styles. In keeping with the CSS cascade order, CSS style rules in a relatively later module (lower down the list) have priority over style rules defined in a relatively earlier module (higher up the list). CSS styles in the referring SPS module have priority over those in any added module. To change the relative position of an added module, right-click it in the Design Overview and click, as required, the **Move Up** or **Move Down** command in the context menu.

The module order is not significant for resolving conflicts among scripts, global templates, design fragments, and page layout items.

**File modification alerts**
If any added file (whether an SPS module, schema, or Working XML File) is modified after the referring SPS module has been opened, then a file modification pop-up will alert you to the change and ask whether the referring SPS module should be refreshed with the changes.

Activating/deactivating the added object
All module objects in all added modules (whether added directly or indirectly) are added to the referring SPS and are listed under the corresponding headings in the Design Tree: Scripts; Global Templates; Design Fragments; XSLT Templates; and XPath Functions. Next to each of these objects is a check box (see screenshot below), which you can check or uncheck to, respectively, activate or deactivate that object. When an object is deactivated, it is effectively removed from the SPS.

In the screenshot above, all the global templates used in the AddressBook.sps module are listed under the Global Templates heading. Those that have been added via other modules (whether directly or indirectly) are displayed in gray. Those that have been created directly in AddressBook.sps are displayed in black. The screenshot shows that only one global template, addr:Email, has been created in AddressBook.sps itself. All the other global templates have been added via other modules, and the file in which each of these is defined is listed next to its name.

Notice that there are two global templates for addr:Email, one created in the referring SPS (AddressBook.sps) itself, and the other created in the added module ContactPoints.sps. If more than one global template has the same (namespace-) expanded name, then only one of these will be active at a time. You can select which one by checking its check box. (Alternatively,
you activate the global template from its context menu in Design View.) This mechanism is useful if you: (i) wish to override an added global template with one that you create in the referring SPS module, or (ii) wish to resolve a situation where a global template for one element is defined in more than one added module.

A global template that has been defined in the current SPS can be deleted by selecting it and clicking the Remove button. However, global templates that have been defined in an added module cannot be removed from the referring SPS. They must be removed by opening the added SPS and removing the global template there.

Individual scripts, Design Fragments, and page layout items can be activated and deactivated in the same way.

---

**Applying or using modular objects**

In the referring SPS module, you design your templates as usual. Each different type of added object is used or applied differently. You should, of course, ensure that each module object you wish to apply has been activated.

**Global templates**

When you wish to use a global template from any of the added SPS modules, you must make sure that this global template is indeed applied. This can be done in one of two ways, according to which one is appropriate for your design:

- In the main template, specify that the element template either uses the global template for that element or copies that global template locally. These two commands are available in the context menu that appears when you right-click the element tag in the design.
- In the main template, the contents or rest-of-contents placeholders cause templates to be applied, leading to the relevant global templates being processed.

**Design Fragments**

To use a Design Fragment, drag it from the Design Tree to the desired location in the main template or a global template. Make sure that the location where the Design Fragment is dropped is the correct context node for that Design Fragment. For details, see Design Fragments.

**Scripts**

All JavaScript functions (whether in an added module or created in the referring SPS) are available as event handlers, and can be set for a particular event via the Properties sidebar.

---

- See also
  - Design Overview
  - Available Module Objects
  - Example: An Address Book
  - Schema Sources
  - Templates and Design Fragments
  - Using Scripts
Example: An Address Book

The (My) Documents folder, C:\Documents and Settings\<username>\My Documents\Altova \StyleVision2015\StyleVisionExamples\Tutorial\ModularSPS, contains examples of modular SPSs. The example files in this folder comprise a project in which an address book containing business and personal contacts is modularized. The example not only demonstrates the mechanisms in which modularization is implemented, but also illustrates the main reasons why one would modularize.

- The complete address book is composed of two modules: (i) a business address book, and (ii) a personal address book, each of which has a separate SPS defining different designs. The two modules together make up the composite address book. Modularization in this case is used to compose: the modules are the components of a larger unit.
- Although the content model of each module (business and personal address books) differs slightly from the other, both have a common module, which is the ContactPoints module, consisting of the core contact details: address, telephone, fax, and email. The ContactPoints module can therefore be shared between the two address books (business and personal). Modularization in this case enables a single module to be used as a common unit within multiple other units.
- Further, the ContactPoints module can be modularized to provide more flexibility. In the example project, we have created a separate Address module to contain the postal address, which may have one of three content models, depending on whether the address is in the EU, US, or elsewhere. The output for all three content models is defined in a single SPS. However, they could have been defined in separate SPSs, which would have provided finer granularity. In this case, modularization would provide more flexibility as modules could be re-used more easily.

The description of this project is organized into the following parts:

- The schema files
- The XML data sources
- The SPS files

The schema files

When creating schemas for modular SPSs, the most important thing to bear in mind is to create the elements that you wish to re-use as global elements. The schema for the address book is AddressBook.xsd. This schema has been constructed by importing the schemas for the business address book (BusinessAddressBook.xsd) and personal address book (PersonalAddressBook.xsd). The BusinessAddressBook.xsd schema provides a content model for companies, while the PersonalAddressBook.xsd schema provides a content model for persons (see screenshot below).
Both schemas import the `ContactPoints.xsd` schema (see screenshot below), which defines a content model for contact details.

Finally, the `ContactPoints.xsd` schema (screenshot below) includes the `Address.xsd` schema, which defines the three address-type content models: for EU, US, and other addresses.
Imports are used when the imported schema belongs to a different namespace than the importing schema. Includes are used when the included schema belongs to the same namespace as the including schema.

Note: The screenshots above are of the schema in the Schema View of Altova's XMLSpy.

The XML data sources
The XML data is contained in the file AddressBook.xml. This file is structured so that the AddressBook element contains the companies and persons elements as its children. The content models of these two elements are defined in the schema files, BusinessAddressBook.xsd and PersonalAddressBook.xsd, respectively.

There are two additional XML data files, which correspond to the BusinessAddressBook.xsd and PersonalAddressBook.xsd schemas. These two XML files, BusinessAddressBook.xml and PersonalAddressBook.xml, are used as the Working XML Files of the corresponding SPS files.

The three XML files are the Working XML Files of the following SPS modules:

- AddressBook.xml => AddressBook.sps, ContactPoints.sps, Address.sps
- BusinessAddressBook.xml => BusinessAddressBook.sps
- PersonalAddressBook.xml => PersonalAddressBook.sps

The SPS modules
The description of the SPS modules starts with the most basic module (Address.sps) and progresses in compositionally incremental steps to the complete address book (AddressBook.sps). All the SPS modules use AddressBook.xsd as its schema.

Address.sps
The key points to note are the use of the schema and the Working XML File.

- Address.sps uses AddressBook.xsd as its schema, but the schema could equally well
have been Address.xsd, ContactPoints.xsd, BusinessAddressBook.xsd, or PersonalAddressBook.xsd—since the Address element is present in all these schemas and would be available as a global element. When the SPS module is added to another SPS module, the schema of the imported module is ignored, so which one is used is not important when the SPS is added as a module.

- The Working XML File is AddressBook.xml. Note that the main template in Address.sps specifies that only the Address element should be processed, and that global templates for Address-EU, Address-US, and Address-Other have been defined.

**Main Template */
```
<sn:AddressBook>
   <sn:AddrContactPoints>
      <sn:AddrAddress>
         <sn:AddrTel>123-456-7890</sn:AddrTel>
         <sn:AddrFax>(555) 555-5555</sn:AddrFax>
         <sn:AddrEmail><EMAIL></sn:AddrEmail>
      </sn:AddrAddress>
   </sn:AddrContactPoints>
</sn:AddressBook>
```

Because only the Address element is processed, the output previews show only the output of Address. When Address.sps is used as a module, the global templates are added and the main template is ignored.

**ContactPoints.sps**
This SPS imports one module. Note the use of global templates within other global templates and the main template.

- ContactPoints.sps uses AddressBook.xsd as its schema and AddressBook.xml as its Working XML File.
- Address.sps is added as a module, thus making the global templates of the Address-EU, Address-US, and Address-Other elements available.
- Global templates for the ContactPoints and Email elements are defined. Note that the ContactPoints definition uses the global template of Email (screenshot below).

```
addr:ContactPoints (simple)
```
```
<addr:ContactPoints>
   <addr:AddrAddress>
      <addr:AddrTel>123-456-7890</addr:AddrTel>
      <addr:AddrFax>(555) 555-5555</addr:AddrFax>
      <addr:AddrEmail><EMAIL></addr:AddrEmail>
   </addr:AddrAddress>
</addr:ContactPoints>
```

- The main template—required for the previews—uses the global template of the ContactPoints element, thus enabling previews of the ContactPoints output.

**BusinessAddressBook.sps and PersonalAddressBook.sps**
These SPSs each import one module, which in turn imports another. Note that the main template simply applies global templates.

- Each of these two modules uses AddressBook.xsd as its schema. The Working XML
Files are, respectively, BusinessAddressBook.xml and PersonalAddressBook.xml.

- ContactPoints.sps is added as a module. This causes Address.sps to be indirectly imported. All the global templates in these two modules are available to the referring SPS module.
- In BusinessAddressBook.sps, global templates are defined for the Companies and Company elements. Note that the Company definition uses the global template of ContactPoints.
- In PersonalAddressBook.sps, global templates are defined for the Person and Persons elements. The Person definition uses the global template of ContactPoints.

**AddressBook.sps**
There are two global templates for the Email element; any one can be activated.

- AddressBook.sps uses AddressBook.xsd as its schema. The Working XML File is AddressBook.xml.
- BusinessAddressBook.sps and PersonalAddressBook.sps are added as modules, and this causes ContactPoints.sps and Address.sps to be indirectly imported.
- A global template is defined for the Email element. This means that there are now two global templates for Email, one in ContactPoints.sps and the other in AddressBook.sps (see screenshot below).

In the Global Templates list in the Design Tree (screenshot above), you can select which of the two global templates should be active. StyleVision allows only one to be active at a time. Whichever is active is used within the ContactPoints global template.

- The main template contains some static content for the output header.

**See also**
- Design Overview
- Available Module Objects
- Creating a Modular SPS
- Schema Sources
- Templates and Design Fragments
- Using Scripts
9.4 Templates and Design Fragments

The design document is composed of templates, and it is important to recognize the various types of templates that can be used.

- **Main templates and global templates**: The design document consists of one main template and, optionally, one or more global templates. Global templates can be referenced via the main template.

- **Node-templates and variable iterators**: These are the templates that constitute the main template and global templates. A node-template matches a node in a schema source.

- **Design fragments**: These are templates that are designed separately and re-used in various parts of the design (main template or global templates).

In this section, we describe the role that templates and design fragments play in the structure of the design. We are not concerned here with the presentation properties in the design, only the structure. In this section, we also do not consider additional structural items for paged media, such as cover pages, headers and footers. These are described in the section, Designing Print Output.

**Note**: In Design View, the SPS can have several templates: the main template, global templates, page layout templates, and Design Fragments. You can control which of these template types is displayed in Design View by using Template Display Filters, which are available as toolbar icons. These display filters will help you optimize and switch between different displays of your SPS.

---

**See also**

- Creating the Design
- Schema Sources
- Template Filters
Main Template

The main template determines the structure of the output. This means that the sequence in which the main template is laid out in the design is the sequence in which Authentic View and the output is laid out. In programming jargon, this is procedural processing. Processing starts at the beginning of the template and proceeds in sequence to the end. Along the way, nodes from the XML document are processed. The templates which process these nodes are called local templates. After a local template is processed, the processor moves to the next component in the main template, and so on. Occasionally, a node may reference a global template for its processing. In such cases, after the global template is executed for that node, the processor returns to the position in the main template from which it branched out and continues in sequence from the next component onwards.

The entry point for the main template is the document node of the main schema. The main schema is the schema designated by you as such (right-click the schema in the Design Overview sidebar and select Set As Main Schema Source). StyleVision offers the option of selecting multiple root elements (document elements). This means that within the main template, there can be local templates for each of the active document elements. The one that is executed during processing will be that for the element which is the document element of the XML instance document being processed.

See also

- Creating the Design
- Inserting XML Content as Text
- Global Templates
- Design Fragments
- Template Filters
Global Templates

A global template can be defined for any node or type in the schema, or for a node specified in an XPath pattern.

A global template specifies instructions for the selected node or type, and it is invoked by a call from the main template, design fragments, or other global templates. The processing model is similar to that of declarative programming languages, in that a single template is defined and invoked multiple times. In this way a single definition can be re-used multiple times. Global templates are invoked in two situations:

- When a node or type in the main template has been set to reference its global template (done by right-clicking the component in the design and selecting Make Global Template).
- When a (contents) or (rest-of-contents) is inserted within an element or type in a local template, and the rest of the content of that element or type includes a node or type for which a global template exists.

Global templates are useful if a node (or type) occurs within various elements or in various locations, and a single set of instructions is required for all occurrences. For example, assume that a para element must be formatted the same no matter whether it occurs in a chapter, section, appendix, or blockquote element. An effective approach would be to define a global template for para and then ensure, that in the main template the global template for the para element is processed wherever required (for example, by including //chapter/para in the main template and specifying that para reference its global template; or by including //chapter/title and then including (contents) or (rest-of-contents) so that the rest of the content of the chapter element is processed with the available global templates and default templates). Also, a global template can be defined for a complex type (for example, one that defines an address model) or even for a simple type (for example, xs:decimal). In such cases, all occurrences of the type (complex or simple) that invoke the global template for that type will be processed according to the rules in the global template.

Creating a global template

Global templates can be created for any node or type in the schema, or for a node specified in an XPath pattern, and are created from the Schema Tree sidebar (screenshot below).
A global template can be created in any of the following ways:

- Click the **Add New Global Template** button located at the right of the Global Templates item in the Schema Tree (see screenshot above). This pops up the Add New Global Template dialog (screenshot below). You can select an element, an attribute, or a type from the schema tree shown in the dialog, or you can enter an XPath pattern. (Note that global templates created for nodes selected with an XPath pattern are not supported in Authentic.) This selects the node that must be created as the global template. Click **OK** to finish. The template will be created and appended to the already existing templates in Design View and can then be edited. In the Schema Tree, the schema node or type will be marked with a plus sign icon in front of it.
Right-click the schema node or type component in the Schema Tree (under Root Elements, All Global Elements, or All Global Types, as appropriate), and select the command **Make/Remove Global Template**. The template will be created and appended to the already existing templates in Design View and can then be edited. In the Schema Tree, the schema node or type will be marked with a plus sign icon in front of it.

- Global templates can also be created from templates in the main template in Design View. Right-click the template (either in Design View or the Schema Tree sidebar) and select the command **Make Global Template**. A global template is created from the selected template (it is appended to the templates in Design View) and the template in the main template is automatically defined to **use** this global template (see below for an explanation of how global templates are **used**).

A global template is located in Design View below the main template. It is indicated by a mauve bar containing the name of the node for which the global template has been created, followed by its type: (**simple**) or (**complex**). A global template is shown in the screenshot below.

Note that the processing of the global template is user-defined and could include both static and dynamic components, as well as the whole range of processing options available for processing of the main template.
Using a global template
After a global template has been created, it can be used when a node having the same qualified
name is inserted into the document (When the node is dropped in the design, select the
command Use Global Template from the menu that pops up.) by dropping . Alternatively, if a
local template is present in the design and a global template exists for a node having the same
qualified name, then the global template can be used instead of the local template. To use a
global template for a local template, right-click the local template in Design View and select the
command Use Global Template. When a global template is used, its processing instructions
are called and used by the local template at runtime.

Wherever a global template is used in the design, an XPath pattern can be created on the global
template to filter the nodeset it addresses. To create such a filter, right-click the global template
tag in the design, and select Edit XPath Filter in the context menu that appears. This pops up
the Edit XPath Expression dialog, in which the required expression can be entered.

Recursive global templates
Global templates can be recursive, that is, a global template can call itself. However, to guard
against an endless loop in Authentic View, a property to limit the call-depth can be set. This
property, the Maximum Template-Call-Depth property, is available in the Authentic tab of the
Properties dialog of the SPS (File | Properties). It specifies the maximum number of template
calls that may be made recursively when processing for the Authentic View output. If the number
of template calls exceeds the number specified in the Maximum Template-Call-Depth property, an
error is returned.

Copying a global template locally
After a global template has been created, its processing instructions can be copied directly to a
template of the same qualified name in the main template. To do this, right-click the local
template and select the command Copy Global Template Locally. Copying the global template
locally is different than using the global template (at runtime) in that the processing instructions
are merely copied in a one-time action. The global template has no further influence on the local
template. Either, or both, the global template and local template can subsequently be modified
independently of each other, without affecting the other. On the other hand, if it is specified that a
global template should be used (at runtime) by a local template, then any modifications to the
global template will be reflected in the local template at runtime.

Activating and deactivating global templates
A global template can be activated by checking its entry in the global templates listing in the
Schema Tree sidebar. It can be deactivated by unchecking the entry. If a global template has
been activated (the default setting when the global template was created ), it is generated in the
XSLT stylesheet. If it has been deactivated, it is not generated in the XSLT stylesheet but is still
saved in the SPS design.
Any local template that uses a deactivated global template will then—since it is not able to reference the missing global template—fall back on the default templates of XSLT, which have the collective effect of outputting the contents of descendant text nodes.

The advantages of the activation/deactivation feature are: (i) Global templates do not have to be deleted if they are temporarily not required; they can be reactivated later when they are required; (ii) If there are name conflicts with templates from imported stylesheets, then the global template that is not required can be temporarily deactivated.

Removing a global template
To remove a global template, right-click the global template to be removed, either in Design View or the Schema Tree sidebar, and select the command Make/Remove Global Template.

Simple global templates and complex global templates
Global templates are of two types: simple and complex. Complex global templates are available for reasons of backward-compatibility. If a global template in an SPS created with a version of StyleVision prior to version 2006 contains a table or list, then that global template will typically be opened in StyleVision 2006 and later versions as a complex global template.

A complex global template is different than a simple global template in the way the node for which the global template was created is processed. When the first instance of the node is encountered in the document, the complex global template processes all subsequent instances of that node immediately afterwards. A simple global template, on the other hand, processes each node instance only when that node instance is individually encountered.

It is important to note that a simple global template will be automatically converted to a complex global template if a predefined format or newline is created around the element node for which the global template was created. This will result in the processing behaviour for complex global templates (described in the previous list item). To revert to the simple global template, the predefined format should be removed (by dragging the node outside the predefined format and then deleting the predefined format), or the newline should be removed (by deleting the item in the Design Tree sidebar), as the case may be. To avoid the automatic conversion from simple global template to complex global template, make sure that the predefined format or newline is added within the node tags of the element for which the simple global template was created.

Global templates in modular SPSs
When an SPS module is added to another SPS module, the global templates in the added module are available for use within the referring SPS. For more information about using modular SPSs, see the section Modular SPSs.
See also

- Inserting XML Content as Text
- Rest-of-Contents
- Main Template
- Design Fragments
- Template Filters
- Modular SPSs
User-Defined Templates

User-Defined Templates are templates for items generated by an XPath expression you specify. These items may be atomic values or nodes. In the screenshot below, which shows three User-Defined Templates, note the User-Defined Template icon on the left-hand side of the tags (a green person symbol). User-Defined Templates are very useful because they provide extraordinary flexibility for creating templates. Note, however, that content generated by User-Defined Templates cannot be edited in Authentic View.

The XPath expression of each of the three User-Defined templates shown in the screenshot above do the following:

- Selects a node in a source schema. By using an XPath expression, any node in any of the schema sources can be reached from within any context node. If StyleVision can unambiguously target the specified node, the template will be changed automatically from a User-Defined Template to a normal template, enabling Authentic View editing. If it is a User-Defined Template, this will be indicated by the green User-Defined Template icon on the left-hand side of the template tags.
- Selects a node that fulfills a condition specified by the \texttt{for} construct of XPath 2.0 and XPath 3.0. Such templates can never resolve to normal templates (but will remain User-Defined Templates) because the \texttt{for} construct does not allow StyleVision to unambiguously resolve the target from only the schema information it currently has at its disposal.
- Selects a sequence of atomic values \{1, 2, 3\}. While it is allowed to create a template for an atomic value, you cannot use the \texttt{contents} placeholder within such a template. This is because the \texttt{xsl:apply-templates} instruction (which is what the \texttt{contents} placeholder generates) can only be applied to node items (not atomic values). You could, however, use an Auto-Calculation in combination with some design element such as a list. For example, the User-Defined Template at left would generate the output at right.

\textbf{Note:} If the SPS uses XSLT 1.0, then the XPath expression you enter must return a node-set. Otherwise an error is reported.
**Advantage of using XPath to select template node**

The advantage of selecting a schema node via an XPath expression (User-Defined Templates) is that the power of XPath's path selector mechanism can be used to select any node or sequence of items, as well as to filter or set conditions for the node selection. As a result, specific XML document nodes can be targeted for any given template. For instance, the XPath expression `//Office/Department[@Location="NY"]` will select only those Department nodes that have a Location attribute with a value of NY. Also see the other examples in this section.

**Note:** If an XPath expression contains multiple location path steps, then it is significant—especially for grouping and sorting—whether brackets are placed around the multiple location path steps or not. For example, the XPath expression `/Org/Office/Dept` will be processed differently than `(/Org/Office/Dept)`. For the former expression (without brackets), the processor loops through each location step. For the latter expression (with brackets), all the Dept elements of all Office elements are returned in one undifferentiated nodeset.

<table>
<thead>
<tr>
<th>Brackets</th>
<th>Underlying XSLT Mechanism</th>
<th>Effect</th>
</tr>
</thead>
</table>
| No       | `<xsl:for-each select="Org">`  
           | `<xsl:for-each select="Office">`  
           | `<xsl:for-each select="Dept">`  
           | ...  
           | `</xsl:for-each>`  
           | `</xsl:for-each>`  | Each Office element has its own Dept population. So grouping and sorting can be done within each Office. |
| Yes      | `<xsl:for-each select="/Org/Office/Dept">`  
           | ...  
           | `</xsl:for-each>` | The Dept population extends over all Office elements and across Org. |

This difference in evaluating XPath expressions can be significant for grouping and sorting.

**Inserting a User-Defined Template**

To insert a User-Defined Template, do the following:

1. Click the **Insert User-Defined Template** icon in the Insert Design Elements toolbar and then click the design location where you wish to insert the template. Alternatively, right-click the design location where you wish to insert the template and, from the context menu that appears, select the **Insert User-Defined Template** command.
2. In the **Edit XPath Expression** dialog that pops up, enter the XPath expression you want, and click **OK**. Note that the context node of the XPath expression will be the node within which you have clicked. An empty node template will be created. Sometimes a joined node is created. When a node is joined, the targeted instance nodes are selected as if at a single level, whereas if a node is not joined (that is if it is split into multiple hierarchic levels), then the node selection is done by looping through each instance node at every
split level. The nodeset returned in both cases of selection (joined and split) is the same unless a grouping or sorting criterion is specified. For a discussion of the effect joined nodes have on the grouping and sorting mechanisms, see Node-Template Operations.

---

**Editing a Template Match**

The node selection of any node template (user-defined or normal) can be changed by using an XPath expression to select the new match expression. To edit the template match of a node template, right-click the node template, then select the **Edit Template Match** command. This pops up the Edit XPath Expression dialog, in which you enter the XPath expression to select the new node. Then click **OK**.

---

**Adding nodes to User-Defined Templates**

If a node from the schema tree is added to a User-Defined Template, the context for the new node will not be known if the User-Defined Template has been created for a node or sequence that cannot be placed in the context of the schema source of the SPS. You will therefore be prompted (screenshot below) about how the new node should be referenced: (i) by its name (essentially, a relative path), or (ii) by a full path from the root of the schema source.

![Create design element dialog](ui.png)

Prompting for advice on how to proceed is the default behavior. This default behavior can be changed in the Design tab of the **Tool | Options dialog**.
Variable Templates

A Variable Template is a template that targets a variable and, by default outputs its content. It is inserted with the Insert Variable Template or Enclose with Variable command, which inserts, at the cursor insertion point, a template for a variable defined in the SPS. The variable template (screenshot below) contains a content placeholder by default, and this serves to output the contents of the variable. You can insert additional content (static as well as dynamic) in the variable template as required, or modify it as you would any other template. A variable template is indicated with a dollar symbol in its start and end tags.

To insert a variable template, do the following:

1. Place the cursor in the design at the location where the template is to be inserted.
2. Click the Insert Variable Template command. This pops up the Insert Variable Template dialog (screenshot below).

3. The dialog contains a list of all the user-declared parameters and variables defined in the SPS. Select the variable for which you wish to add a variable template.
4. Click OK to finish.
Node-Template Operations

A node-template is a template in the design that specifies the processing for a node. In the design, node-templates are displayed with beige start and end tags (screenshot below). The type of node is indicated by a symbol inside the tags (For example: angular brackets for element nodes and equal-to signs for attribute nodes). The screenshot below contains two node-templates, both for elements: `metainfo` and `relevance`. Also see, Nodes in the XML document.

The operations that can be carried out on a node-template are accessible via the context menu of that node-template (accessed by right-clicking either the start or end tag of a node-template, see screenshot below).

The commands in this context menu are described below:

- Global templates
- Template match
- XPath filters
- Group by, Sort by, Define variables, Template serves as level

[Diagram of node-templates]

[Diagram of context menu with commands]

© 2015 Altova GmbH
Create Design Fragment
Remove Tag Only
Edit, Enclose with, Change to
Authentic properties

These menu commands are described below. Note that for a given node-template, some commands might not be available; these are grayed out in the context menu.

---

Global templates: make, use, copy locally
A node-template in the main template can be changed to or associated with a global template via the following commands:

- **Make global template:** This option is available if the node-template represents an element that is defined as a global element in the schema. A global template will be created from the node-template. The node-template in the main template will use this global template and its tags will then be displayed in gray (indicating its use of the global template).
- **Use global template:** If a global template of the same qualified name as the node-template has been defined, the node-template will use the processing of the global template. The tags of the node-template will become gray.
- **Copy global template locally:** The processing instructions of a global template of the same qualified name as the node-template are copied physically to the node-template. The node-template is independent of the global template. Subsequently, both it and the global template can be modified independently of each other. Since the node-template does not reference a global template, it retains its beige color.

For more information, see the section Global Templates.

---

Editing the template match
The node for which a template has been created can be changed by using this command. The Edit Template Match command pops up the Edit XPath Expression dialog, in which you can enter an XPath expression that selects another node in the schema. You can also enter any XPath expression to change the template to a User-Defined Template.

---

Edit/Clear XPath Filter
An XPath filter enables you to filter the nodeset on which a node-template is applied. XPath filters can also be applied to global templates.

By default, a node-template will be applied to nodes (elements or attributes) corresponding to the node for which the node-template was created (having the same name and occurring at that point in the schema hierarchy). For example, a node-template for the /Personnel/Office node will select all the /Personnel/Office elements. If an XPath filter with the expression 1 is now created on the Office element (by right-clicking the Office element and editing its XPath Filter),


this has the effect of adding a predicate expression to the `Office` element, so that the entire XPath expression would be: `/Personnel/Office[1]`. This XPath expression selects the first `Office` child of the `Personnel` element, effectively filtering out the other `Office` elements.

A filter can be added to any node-template and to multiple node-templates in the design. This enables you to have selections corresponding to such XPath expressions as: `/Personnel/Office[@country='US']/Person[Title='Manager']` to select all managers in the US offices of the company. In this example, a filter each has been created on the `Office` and on the `Person` node-templates, respectively.

Wherever a global template is used—that is, called—an XPath filter can be applied to it. So, for every instance of a global template that is used, an XPath filter can be applied to the global template in order to restrict the targeted nodeset.

To add an XPath Filter to a node-template, right-click the node-template and select **Edit XPath Filter**. Enter the XPath filter expression without quotes, square brackets, or delimiters of any kind. Any valid XPath expression can be entered. For example:

- `1`  
- `@country='US'`  
- `Title='Manager'`

After an XPath Filter has been created for a node-template, this is indicated by a filter symbol in the start tag of the node-template. In the screenshot below, the `synopsis` node-template has a filter.

```
<synopsis.tpl>(content)< synopsis
```

**Note:** Each node-template supports one XPath Filter.

---

**Group by, Sort by, Define variables, Template Serves as Level**

The mechanisms behind these commands are described in detail in their respective sections:

- The **Group by** command enables instances of the node represented by the selected node-template to be grouped. The grouping mechanism is described in the section, **Grouping.**
- The **Sort by** command enables instances of the node represented by the selected node-template to be sorted. The sorting mechanism is described in the section, **Sorting.**
- The **Define Variables** command enables you to define variables that are on scope on the selected node-template. How to work with variables is described in the section, **Variables.**
- The **Template Serves as Level** command is a toggle command that creates/removes a level on the node-template. Levels can be specified at various levels in order to structure the document into a hierarchy. This structure can then be used to generate a table of contents (TOC), automatic numbering, and text references. These features are described in detail in the section, **Table of Contents (TOC) and Referencing.**
Create Design Fragment
Creates a Design Fragment template from the selected template. The resulting Design Fragment template is added to the Design Fragment templates at the bottom of the design, and added to the Design Tree and Schema Tree. The Design Fragment is also applied at that point in the design where it was created.

Remove (Template or Formatting) Tag Only
This command removes the selected template or formatting tag only. It does not remove any descendant nodes or formatting tags. This command is useful for removing a formatting tag or a parent element tag without removing all that is contained within the tag (which is what would happen if the Delete operation is carried out with a tag selected). Note, however, that removing a parent element might render descendant nodes of the deleted element invalid. In such cases, the invalid nodes are indicated with a red strike-through.

Edit, Enclose with, Change to
These commands are described below:

- **Edit**: Pops out a submenu with the familiar Windows commands: cut, copy, paste, and delete.
- **Enclose with**: The node-template can be enclosed within the following design components, each of which is described in a separate section of this documentation: paragraph, special paragraph, Bullets and Numbering, Hyperlink, Condition, TOC Bookmark and Level.
- **Change to**: The Change-To feature enables you to change: (i) the node for which that template applies, or (ii) how the node is created in the design. It is described in detail in the section, The Change-To Feature.

Edit Authentic Properties
Selecting this command pops up the Properties sidebar, in which certain properties (Authentic Properties) of the instantiated node in Authentic View can be defined. Authentic Properties are described in detail in the section, Authentic Node Properties.
Design Fragments

Design Fragments are useful for creating parts that can be re-used at different locations in the document, similar to the way functions are re-used. The usage mechanism is as follows:

1. Create the Design Fragment in the design
2. Fill out the contents of the Design Fragment
3. Insert the Design Fragment at a location in a template.

Creating a Design Fragment

To create a Design Fragment do the following:

1. In the Design Tree or Schema Tree, click the Add New Design Fragment icon, which is located to the right of the Design Fragments item in the tree (see screenshot below). This adds a Design Fragment item in the Design Fragments list of the tree. (Also see note below.)

![Design Tree Screenshot](image)

Notice that a Design Fragment template is created in the SPS design. This template is appended to the templates already in the design and indicated with a green header. (If you wish to see only the Design Fragments that are in the design, hide the main template and global templates by clicking their Show/Hide icons in StyleVision’s Design Filter toolbar.) Additionally, the Design Fragment templates are also listed in the schema tree for ready access from there.

2. Double-click the Design Fragment item (either in the design tree or the schema tree) so as to edit its name. Name the Design Fragment as required and press Enter. The edited name is entered in the Design Tree (screenshot below) and in the template in the design.

![Design Fragment Name Edited Screenshot](image)

3. In the design, create the contents of the Design Fragment template. How to do this is described in the next section.

Note: If you wish to create a Design Fragment from an already existing template, right-click that template and select the command Create Design Fragment from the context menu that pops up. This creates a Design Fragment template from the selected template at that
Creating the contents of a Design Fragment

The contents of the Design Fragment template are created as for any other template. To insert static content, place the cursor in the Design Fragment template and insert the required static content. To insert dynamic content, drag the required schema node into the Design Fragment template.

When dragging a node from the schema source you can drag the node either: (i) from the Global Elements tree, or (ii) from the Root Elements tree. The difference is significant. If a node is dragged from the Global Elements tree, it is created without its ancestor elements (in the screenshot below, see the EmailPerson Design Fragment) and, therefore, when used in a template, it will have to be used within the context of its parent. On the other hand, if a node is dragged from the Root Elements tree, it is created within a structure starting from the document node (in the screenshot below, see the EmailDocNode Design Fragment), and can therefore be used anywhere in a template.

The screenshot above shows two Design Fragment templates that produce identical output for the Person element. In the EmailPerson Design Fragment template, the Person node has been created by dragging the global element Person into the EmailPerson template. In the EmailDocNode Design Fragment template, the Person node has been dragged from the Root Elements tree, and is created with an absolute path (from $XML, the document node).

When these Design Fragment templates are inserted in the main template, care must be taken that the EmailPerson template is called from within a context that is the parent of the Person
node. You can experiment with these Design Fragments. They are in the example file Email.sps, which is in the (My) Documents folder, C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial\DesignFragments.

You can also define a parameter with a default value on the Design Fragment. The parameter can be assigned a different value in every Design Fragment instance. See Parameters for Design Fragments for details.

After you have completed the design, notice that the components of the design are also graphically depicted in the Design Tree.

---

**Inserting a Design Fragment in a template**

To insert a Design Fragment, drag the Design Fragment from the Design Tree or Schema Tree to the required location. The location at which the Design Fragment is dropped should be such that it provides a correct context. If the contents of the Design Fragment were created from a global element, then the correct context in the main template would be the parent of the node dragged into the Design Fragment. See Creating the contents of a Design Fragment above.

Alternatively, right-click at the location where the Design Fragment is to be inserted and select Insert Design Fragment from the context menu.

**Note:** If a Design Fragment is referenced in the main template and if the name of the Design Fragment is changed subsequently, then the reference in the main template will no longer be correct and an XSLT error will result. In order to correct this, delete the original reference in the main template and create a fresh reference to the newly named Design Fragment.

---

**Recursive design fragments**

Design fragments can be recursive, that is, a design fragment can call itself. However, to guard against an endless loop in Authentic View, a property to limit the call-depth can be set. This property, the Maximum Call-Depth property, is available in the Authentic tab of the Properties dialog of the SPS (File | Properties). It specifies the maximum number of template calls that may be made recursively when processing for the Authentic View output. If the number of template calls exceeds the number specified in the Maximum Call-Depth property, an error is returned.

---

**Deleting a Design Fragment**

To delete a Design Fragment, select it in the Design Tree and click the Remove toolbar icon of the Design Tree.
Design Fragments in modular SPSs
When an SPS module is added to another SPS module, the Design Fragments in the added module are available for use within the referring SPS. For more information about using modular SPSs, see the section Modular SPSs.

Example file
For an example SPS, go to the (My) Documents folder C:\Documents and Settings \<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial \DesignFragments.

See also
- Inserting XML Content as Text
- Rest-of-Contents
- Main Template
- Global Templates
- Template Filters
- Modular SPSs
9.5 XSLT Templates

XSLT files can be imported into an SPS, and XSLT templates in them will be available to the stylesheet as global templates. If, during the processing of the XML document, one of the XML nodes matches a node in an imported XSLT template, then the imported XSLT template is applied to that node. If the imported XSLT file contains named templates, these are available for placement in the design.

Note:
- Imported XSLT templates cannot be modified in StyleVision.
- XSLT templates are not supported in Authentic View.

Importing the XSLT file
To import an XSLT File, do the following:

1. In the Design Overview sidebar (screenshot below), click the Add New XSLT File link.

![Design Overview Screenshot]

2. In the Open dialog that appears, browse for the required XSLT file, select it, and click Open. The XSLT file is imported. An xsl:import statement is added to the XSLT stylesheet, and, in the Design Tree sidebar (screenshot below), the XSLT Templates contained in the imported XSLT file are displayed under the XSLT Templates heading.

![Design Tree Screenshot]
There are two types of imported XSLT templates: (i) match templates (indicated by *Match*), and (ii) named templates (indicated by *Name*). In the Design Tree, these two types are listed with (i) the value of the *select* attribute of match templates, and (ii) by the value of the *name* attribute of named templates, respectively.

---

**Match templates**

Match templates will be used when a template, in the course of processing, applies templates to a node in the XML document instance, and the match template is selected to be applied. This will happen when the qualified name of the XML node matches the qualified name of the imported match template. If a global template has been created in the SPS that has the same qualified name, then it has precedence over an imported template and will be used. If there are several imported XSLT files, the file imported first (and listed first in the XSLT code) has the lowest precedence, followed by the second lowest precedence for the file imported second, and so on.

---

**Named templates**

A named template can be dragged from the Design Tree to any location in the design. At this location, it will be created as an *xsl:call-template* element (*screenshot below*) that calls the named template.

```xml
<xsl:call-template name="BoldItalic" />
```

The effect of this in the output is to implement the named template at that location in the design. This can be useful for inserting content that is independent of both the XML instance document as well as of the XSLT stylesheet.
9.6 Multiple Document Output

You can design an SPS to produce multiple output-documents: a main output-document and one or more additional documents. This is particularly useful if you wish to modularize the output. Output-documents are created in the design by inserting a New Document template (see screenshot below). Content for each output-document is placed within its New Document template.

New Document templates can be created anywhere in the document design, thus allowing the output to be modularized at any level. So, for example, a report about the various branch offices of a global organization can have separate output-documents at each of the following levels: (i) world, (ii) continent, (iii) country, (iv) state, and/or (v) branch office. Each branch office, for example, can be presented in a separate output-document or all the branch offices in a country can appear together in a single country report. In the design, a New Document template would have to be created at each of the hierarchical levels for which separate output-documents are required. How to set up the correct document structure is described in the section, New Document Templates and Design Structure.

The multiple output-document feature is available in all formats: HTML, RTF, PDF, and Word 2007. In Authentic View, which is intended as an editing view, content is displayed in a single document.

This description of multiple output-documents is organized into the following sub-sections:

- Inserting a New Document Template
- New Document Templates and Design Structure
- URLs of New Document Templates
- Preview and Output Document Files
- Document Properties and Styles

See also
• HTML fragments in output
Inserting a New Document Template

A New Document template can be placed in an SPS design in one of two ways:

- A new output-document template can be inserted at any location in the design. In this case the content of the New Document is added to the template after inserting the template. To insert a New Document template, place the cursor at the desired location in the design and select the command Insert | Insert New Document or right-click the location and, from the context menu that pops up, select Insert New Document.
- A new output-document can be placed in the design by enclosing content with a New Document template. The New Document template will, in this case, contain the enclosed content when it is created. You can add to or modify this content in the design. To place a New Document template so that it encloses content, highlight the content to be enclosed and then select the command Enclose With | New Document. Alternatively, you can select the content to be enclosed, then right-click it, and, from the context menu that pops up, select the command Enclose With | New Document.

A New Document template with content is shown in the screenshot below.

Notice the following from the screenshot above:

1. The New Document template tags contain the URL (path and name) of the output-document it will generate. The filename suffix will be generated automatically according to the file type of the output format. For example, for the HTML output format, the filename suffix .html will be appended to the filename in the URL. Issues relevant to the assigning of URLs are discussed in the section, URLs of New Document Templates.
2. The New Document Template contains one Initial Document Section. Additional document sections can be added to the initial document section as described in the section, Document Sections.

See also

- Multiple Document Output
- New Document Templates and Design Structure
- URLs of New Document Templates
- Preview and Output Document Files
- Document Properties and Styles
New Document Templates and Design Structure

When creating multiple output-documents, you must create the different New Document templates on the appropriate nodes of the source document. Therefore, you must consider both the output structure as well as the input (source XML document) structure when designing multiple output-documents.

Main output document and additional output documents (output structure)

When the first New Document template is added to the design, all design content outside this New Document template is automatically assigned to a separate document. This separate document is considered to be the main output document, and, in the output previews of StyleVision, it is referred to as Main Output Document.

In the generated output-documents (created using the command File | Save Generated Files), the name of the main output document will be the name you assign it when generating the output-document files using the Save Generated Files command. The names of the additional output-document files will be the names assigned in the URLs of the respective New Document templates.

New Document templates and source document structure

When a New Document template is created, the hierarchical location where it is created is significant. Two possibilities exist:

1. The node within which the New Document template is created is processed only once. In this case the New Document template is also processed only once. The filename in the URL property of the New Document template can therefore be a static name.

2. The node within which the New Document template is created is processed multiple times. As a result, the New Document template will be processed as many times as the node is processed. An example of such a situation would be the following. An Office element has multiple Department element children (for its various departments). If a New Document template is created within the Department node in the design, then, since the Department node will be processed multiple times (for all the different Department elements in that Office element), the New Document template also will be processed multiple times, once for each Department element in the source XML document. The filename in the URL property of the New Document template must therefore be a dynamic name. Otherwise, the output-documents created for the Department elements will each have the same filename.

See also
- Multiple Document Output
- Inserting a New Document Template
- URLs of New Document Templates
- Preview and Output Document Files
- Document Properties and Styles
URLs of New Document Templates

In this section we describe how the URLs of New Document templates relate to design structure, how URLs are edited, and how multiple output-documents can be linked among each other.

URLs of New Document templates

If the New Document template is processed only once (see preceding section), then the template's URL property can be a static URL. In the screenshot below, since the New Document template is immediately within the document element ($XML), it will be processed only once. The URL has been given a static value of TableOfContents. This value will therefore be the filename of the output-document. Since no path has been prefixed to the filename, the file will be generated in the same directory as the Main Document File (see Multiple Document Outputs and Previews for details). Alternatively, if the URL contained a path, the output-document will be saved to the location specified in the path.

If, on the other hand, a New Document template will be processed multiple times to generate multiple output-documents (see preceding section), then the template's URL property must be a dynamic URL that is selected with an XPath expression. In the screenshot below, the URL of the New Document template is the XPath expression: body/header/para. The New Document template is within the topic element, so it will be processed each time the topic element is processed. On every iteration through the topic element, the content of that topic element's body/header/para element will be assigned as the URL of the New Document template. This creates a new document for every topic element. Each of these documents has a different name, that of its body/header/para element (which is the text of the topic's header).
Editing the URL

When a New Document template is added to the design, it is created with a default URL. This is a static text string: \texttt{DocumentX} (where \texttt{X} is an integer). If you wish to edit the URL, right-click the New Document template and select the command \textit{Edit URL}. This pops up the Properties dialog (screenshot below), in which you can edit the Value field of the \texttt{URL without file ext} property.

If you wish to enter a static URL, edit the Value field to contain the required URL text. If you wish to enter a dynamic URL, click in the Value field, click the XPath button in the toolbar of the Properties dialog, and enter the XPath expression you want. Note the following: (i) The context node for the XPath expression is the node within which the New Document template has been inserted. (ii) To prefix a path to the XPath location expression, use the \texttt{concat()} function of XPath. For example: \texttt{concat(‘C:\MyOutput\’, body\header\para)}. This example expression will generate the URL string: \texttt{C:\MyOutput\filename}. The appropriate file extension will be generated automatically according to the output format.
Linking the documents

Multiple output-documents can be linked to one another using bookmarks and hyperlinks. A bookmark can be placed at the head of a New Document Template or anywhere within the New Document Template. Hyperlinks can then be created in other documents to link back to the bookmark. If bookmarks are required on a node that is processed multiple times, then make sure that the name of the bookmark is generated dynamically. Otherwise (if a static bookmark name is given) multiple nodes in the output will have the same bookmark name.

A Table of Contents (TOC) can also be used to link documents. The TOC could be in a separate document (for example, the main document) and link to the various output-documents, while the output-documents could link back to the TOC.

Note: Links to external documents are supported in the FO spec but might not be supported by the FO processor you are using. You should check support for this feature in your FO processor if you are planning to use links to external documents in your PDF output documents.

See also

- Multiple Document Output
- Inserting a New Document Template
- New Document Templates and Design Structure
- Preview and Output Document Files
- Document Properties and Styles
Preview Files and Output Document Files

The output previews of a design document show each of the multiple output-documents that have been specified in the design as separate documents (see screenshot below).

The screenshot above shows the HTML Preview of an SPS document that has been designed to generate multiple output-documents. Each output document can be called up in the view window by either: (i) navigating through the available documents using the arrow buttons at top left, or (ii) selecting the required document from the dropdown list of the combo box (see screenshot above). Notice that the items of the dropdown list show the entire URL (path plus filename).

Location of preview files

The preview files are created by default in the directory in which the SPS file is created. This default setting can be changed in the Paths tab of the SPS file’s Properties dialog (screenshot below), which is accessed with the File | Properties command. In this tab you can specify the directory of the Working XML File as an alternative location. If the URL of a New Document template contains a path, the location specified in this path will be used as the location of the respective preview files. If the location cannot be found, an error is returned. You should be aware of where the output-documents will be saved if you are setting up output-documents to link to each other.
In the Paths tab of the Properties dialog (see screenshot above), you can also specify where temporary additional files for previews, such as output-document files, and image and chart-image files, will be saved. Note that, if the URL of a New Document template contains a path, then the location specified in this path will be used.

---

**Generating output (paths etc)**

To generate the output-document files, do the following:

1. Mouse over the menu command **File | Save Generated Files** and click the required output format.
2. In the Save Generated File dialog that pops up browse for the folder in which you wish to save the generated file.
3. Enter the name of the Main Document File and click **Save**.

The location of all output-document files as well as other additionally generated files, such as image files and chart-image files, will be displayed in a pop-up window for your information.

The Main Document File will be saved to the folder location you selected in the Save Generated File dialog. All the multiple output-documents that were created with New Document templates and whose URLs have no path information will be saved to the same folder as the Main Document.
File. If a path was prefixed to the filename in a New Document template's URL, the output document will be saved to the location specified in the URL. If that folder location does not exist an error will be generated.

See also
- Multiple Document Output
- Inserting a New Document Template
- New Document Templates and Design Structure
- URLs of New Document Templates
- Document Properties and Styles
Document Properties and Styles

In an SPS design, you can split the output into multiple documents. Each of these documents can be assigned separate document properties and document styles. These are specified in the Document Properties and Document Styles tabs, respectively (see screenshot below), of the Properties dialog of the document’s Initial Document Section. To access the Properties dialog, click the Edit Properties link in the title bar of the Initial Document Section of the document for which you wish to set these properties. Document properties and document styles apply to the entire output document.

In the Document Properties tab, the Document Properties group of properties enable meta-information to be entered for the document. This meta-information will be saved to the respective output document and to the respective properties according to output format. For example, in the HTML output format, the properties are stored in the respective META tags of the HEAD element.

Document styles are described in the section Setting CSS Property Values.

See also
- Multiple Document Output
- Inserting a New Document Template
- New Document Templates and Design Structure
- URLs of New Document Templates
- Preview and Output Document Files
Chapter 10

SPS File: Advanced Features
10 **SPS File: Advanced Features**

How to create the basic content and structure of the SPS design is described in the sections, *SPS File Content* and *SPS File Structure*. Very often, however, you will also need to modify or manipulate the content and/or structure of source data in particular ways. For example, you might wish to sort a group of nodes, say nodes containing personnel information, on a particular criterion, say the alphabetical order of employee last names. Or you might wish to group all customers in a database by city. Or add up a product's sales turnover in a particular city. Such functionality is provided in StyleVision's advanced features, and these are described in this section.

Given below is a list of StyleVision’s SPS file advanced features:

- **Auto-Calculations**. Auto-Calculations are a powerful XPath-based mechanism to manipulate data and (i) present the manipulated data in the output as well as (ii) update nodes in the XML document with the result of the Auto-Calculation.
- **Conditions**. Processing of templates and the content of templates can be conditional upon data structures or values in the XML, or upon the result of an XPath expression.
- **Grouping**. Processing can be defined for a group of elements that are selected with an XPath expression.
- **Sorting**. A set of XML elements can be sorted on multiple sort-keys.
- **Parameters and Variables**. Parameters are declared at the global SPS level with a default value. These values can then be overridden at runtime by values passed to the stylesheet from the command line. Variables can be defined in the SPS and these variables can be referenced for use in the SPS.
- **Table of Contents (TOC) and Referencing**. Tables of Contents (TOCs) can be constructed at various locations in the document output, for all output formats. The TOC mechanism works by first selecting the items to be referenced in the TOC and then referencing these marked items in the TOC. Other features which use referencing are: (i) **Auto-Numbering** (repeating nodes in the document can be numbered automatically and the numbers formatted); (ii) **Text References** (text in the document can be marked for referencing and then referenced from elsewhere in the document); and (iii) **Bookmarks and Hyperlinks** (bookmarks mark key points in the output document, which can then be targeted by hyperlinks. Hyperlinks can also link to external resources using a variety of methods to determine the target URI (static, dynamic, a combination of both, and unparsed entity URIs).) All these referencing mechanisms are described in this section.

### See also

- *SPS File Content*
- *SPS File Structure*
- *SPS File Additional Functionality*
10.1 **Auto-Calculations**

The **Auto-Calculation** feature (i) displays the result of an XPath evaluation at any desired location in the output document, and (ii) optionally updates a node in the main XML document (the XML document being edited in Authentic View) with the result of the XPath evaluation.

The Auto-Calculation feature is a useful mechanism for:

- Inserting calculations involving operations on dynamic data values. For example, you can count the number of Employee elements in an Office element (with `count(Employee)`), or sum the values of all Price elements in each Invoice element (with `sum(Price)`), or join the FirstName and LastName elements of a Person element (with `concat(FirstName, ' ', LastName)`). In this way you can generate new data from dynamically changing data in the XML document, and send the generated data to the output.

- Displaying information derived from the structure of the document. For example, you can use the `position()` function of XPath to dynamically insert row numbers in a dynamic table, or to dynamically number the sections of a document. This has the advantage of automatically generating information based on dynamically changing document structures.

- Inserting data from external XML documents. The `doc()` function of XPath 2.0 provides access to the document root of external XML documents, and thus enables node content from the external XML document to be inserted in the output.

- Updating the value of nodes in the main XML document. For example, the node `Addressee` could be updated with an XPath expression like `concat(Title, ' ', FirstName, ' ', LastName)`.

- Presenting the contents of a node at any location in the design.

---

**See also**

- Insert | Auto-Calculation
- XPath Dialog
- XSLT Engine Information
Editing and Moving Auto-Calculations

Creating Auto-Calculations
To create an Auto-Calculation, do the following:

1. Place the cursor as an **insertion point** at the location where the Auto-Calculation result is to be displayed and click **Insert | Auto-Calculation**. In the submenu that appears, select Value if the result is to appear as plain text, select Input Field if it is to appear within an input field (i.e. a text box), or select Multiline Input Field if it is to appear in a multiline text box. (Note that the output of the Auto-Calculation is displayed as a value, or in an Input Field. It is an output in Authentic View, and cannot be edited there.) The Edit XPath Expression dialog pops up (*screenshot below*).

![Edit XPath Expression dialog](image)

2. In the **Expression** pane, enter the XPath expression for the Auto-Calculation via the keyboard. Alternatively, enter the expression by double-clicking nodes, operators, and/or functions in the respective panes of the dialog. It is important to be aware of the context node at the insertion point; the context node is highlighted in the schema source tree (in the screenshot above the context node, `newsitems`, is highlighted). If you have selected XSLT 1.0 as the version of the XSLT language for your SPS, then you must use XPath 1.0 expressions; if you have selected XSLT 2.0 or XSLT 3.0, then you must use, respectively, XPath 2.0 or XPath 3.0 expressions. For a detailed description of the Edit XPath Expression dialog, see the section *Edit XPath Expression*.

3. Optionally, if you wish to copy the value of the Auto-Calculation to a node in the XML document, you can select that node via an XPath expression. How to update nodes with the result of the Auto-Calculation is described in the section *Updating Nodes with Auto-Calculations*.

Click the **OK** button finish. In the Design tab, the Auto-Calculation symbol is displayed. To see the result of the Auto-Calculation, change to Authentic View or an Output View.
Java and .NET functions

Java and .NET functions can be used in the XPath expressions of Auto-Calculations. Here are examples of XPath expressions containing Java and .NET functions:

- **Java:** `java:java.util.Date.toString(java:java.util.Date.new())`
- **.NET:** `clitype:System.Math.PI()`

For more information about extensions, see Appendices | XSLT Engine Information | Extensions.

**Note:** Java and .NET functions in Auto-Calculations are supported only in the Enterprise editions of Authentic Desktop and Authentic Browser.

---

**Editing Auto-Calculations**

To edit the XPath expression of the Auto-Calculation, select the Auto-Calculation and, in the Properties sidebar, click the Edit button of the XPath property in the AutoCalc group of properties (screenshot below). This pops up the Edit XPath Expression dialog (screenshot above), in which you can edit the XPath expression.

---

**Formatting Auto-Calculations**

You can apply predefined formats and CSS styles to Auto-Calculations just as you would to normal text: select the Auto-Calculation and apply the formatting. Additionally, input formatting of an Auto-Calculation that is a numeric or date datatype can be specified via the Input Formatting property in the AutoCalc group of properties in the Properties window.

Note also that you can include carriage returns and/or linefeeds (CR/LFs) in the XPath expression. If the Auto-Calculation is enclosed in the **pre** special paragraph type, the output of a CR/LF will produce a new line in the output (except in the RTF output). An example of such an XPath expression is:

```
translate('a;b;c', ';', codepoints-to-string(13))
```
Moving Auto-Calculations
You can move an Auto-Calculation to another location by clicking the Auto-Calculation (to select it) and dragging it to the new location. You can also use cut/copy-and-paste to move/copy an Auto-Calculation. Note, however, that the XPath expression will need to be changed if the context node in the new location is not the same as that in the previous location.

Summary of important points
Note the following points:

- An Auto-Calculation can be inserted anywhere in the Design Document.
- The point at which you insert the Auto-Calculation determines the context node for the XPath evaluation.
- In Authentic View, an Auto-Calculation is re-evaluated each time any value relevant to the calculation (that is, any node included in the XPath expression) changes.
- An Auto-Calculation result is non-editable in Authentic View or any other output view.
- Any node in the XML document can be updated with the result of the Auto-Calculation.

See also
- Insert | Auto-Calculation
- XPath Dialog
- XSLT Engine Information
- Example: An Invoice
- Updating Nodes with Auto-Calculations
Updating Nodes with Auto-Calculations

You can copy the value (or result) of an Auto-Calculation to a node in the main XML document (the document being edited in Authentic View). You do this as follows:

1. Create the Auto-Calculation as described in Editing and Moving Auto-Calculations.
2. Select the Auto-Calculation and, in the Properties sidebar, click the Edit button of the Update XML node in Authentic property in the Authentic group of properties (screenshot below).

3. In the XPath Selector dialog that pops up (screenshot below), select the node to update and click OK to finish. If you wish the XPath expression to be absolute (that is, starting at the document root), check the Absolute XPath check box.
Alternatively, if you select the User-Defined XPath radio button, you can enter any XPath expression you like to select the node to be updated.

The XPath expression must select a single node. If the XPath expression selects multiple nodes, no node will be updated.

---

**IMPORTANT!**

Only nodes in the XML file of the main schema source will be updated. Nodes in the XML files of additional schemas will not be updated. For a node in the main XML document to be updated two conditions must be fulfilled:

1. The XPath expression for the Auto-Calculation must include at least one value that is related to an XML node, i.e. a dynamic value. For example, Price*1.2. This expression involves the element Price, and is therefore dynamic. If the XPath expression contains a static value (for example, string("Nanonull, Inc.")), then the Update XML Node feature will not work.
2. Any one of the nodes used in the XPath expression must be modified in Authentic View.
So, if the XPath expression is \( \text{Price} \times 1.2 \), and the Auto-Calculation is set to update the \( \text{VATPrice} \) node, then the \( \text{Price} \) element must be modified in Authentic View in order for the \( \text{VATPrice} \) node to be updated.

---

**Changing the node to update, cancelling the update**

To change the node to update, click the **Edit** button of the **Update XML Node** property in the **Authentic** group of properties of the Auto-Calculation (in the Properties window) and then select the required node from the Schema Selector dialog box that pops up. To delete the **Update XML Node** property, click the **Remove** button in the toolbar of the Properties window.

---

**Should you use the Auto-Calculation or the updated node contents for display?**

If there are no conditional templates involved, you can use either the Auto-Calculation or the contents of the updated node for display. It is immaterial which one you choose because the node update happens immediately after the Auto-Calculation is evaluated and there is no factor to complicate the update. You should, however, be aware that there is a different source for the content displayed in each of the two cases.

---

**Hiding the Auto-Calculation**

You may find yourself in the situation where you wish to use an Auto-Calculation to make a calculation in order to update a node with the value of the Auto-Calculation. In this case, one of the following scenarios arises:

- You wish to display the result just once. You cannot hide the Auto-Calculation since it would then not be evaluated. If there is no conditional template involved, it is best to display the Auto-Calculation and not display the contents of the updated node.
- You wish not to display the result, merely to update the node. The best way to handle this scenario is to apply text formatting to the Auto-Calculation, so that it is invisible on the output medium (for example, by applying a white color to an Auto-Calculation on a white background).

---

**See also**

- Editing and Moving Auto-Calculations
- Auto-Calculations Based on Updated Nodes
- Example: An Invoice
- XPath Dialog
Auto-Calculation Based on Updated Nodes

If you wish to create an Auto-Calculation (second Auto-Calculation) that uses a node updated by another Auto-Calculation (first Auto-Calculation), there are two possible situations:

- The two Auto-Calculations are in the same template.
- The two Auto-Calculations are in different templates.

Auto-Calculations in the same template

When two Auto-Calculations are in the same template, the SPS applies the following procedure:

1. A node used in the XPath expression of the first Auto-Calculation is modified.
2. All node values in the XML document are read and all Auto-Calculations are executed.
3. Assuming that the first Auto-Calculation is executed correctly, it updates the specified XML node (call it Node-A). The second Auto-Calculation, which is based on Node-A, will be executed but will use the value of Node-A before Node-A was updated. This is because the value of Node-A was read before it was updated, and has not been read since then.
4. If the document is now edited in any way or if document views are changed (from and to Authentic View), then the values of nodes are read afresh and Auto-Calculations are executed.
5. The second Auto-Calculation is now carried out. (If this Auto-Calculation is intended to update a node, then, as is usual for node updates, a node used in the XPath expression will have to be changed before the update takes place.)

The time lag between the updating of Node-A and the evaluation of the second Auto-Calculation with the updated value of Node-A could be confusing for the Authentic View user. To ensure that this situation does not occur, it is best that the XPath expression of the second Auto-Calculation contain the XPath expression of the first Auto-Calculation—not the updated node itself. As a result, the second Auto-Calculation will execute with the input to the first Auto-Calculation and perform that Auto-calculation as part of its own Auto-Calculation. This enables it to be evaluated independently of the contents of Node-A.

Example

The first Auto-Calculation calculates the VAT amount of a product using the nodes for (i) the net price, and (ii) the VAT rate; it updates the VAT-amount node. The second Auto-Calculation calculates the gross price, which is the sum of net price and VAT amount; it updates the gross price node.

- The Auto-Calculation to calculate the VAT amount is: NetPrice * VATRate div 100. When the VAT rate of the product is entered, the Auto-Calculation is executed and updates the VATAmount node.
- If the Auto-Calculation to calculate the gross price is: NetPrice + VATAmount, then the Auto-Calculation will execute with the value of VATAmount that was read in before VATAmount was updated.
- If, however, the Auto-Calculation to calculate the gross price is: NetPrice + (NetPrice * VATRate div 100), then the Auto-Calculation will execute with the value of VATRate and will update the GrossPrice node. The updated VATAmount node has been left out of...
the second Auto-Calculation.

For a detailed example, see Example: An Invoice.

---

**Auto-Calculations in different templates**

When two Auto-Calculations are in different templates, a node updated by the first Auto-Calculation can be used by the second Auto-Calculation. This is because Auto-Calculations are calculated and nodes updated for each template separately. For an example of how this would work, see Example: An Invoice.

- **See also**
  - Editing and Moving Auto-Calculations
  - Updating Nodes with Auto-Calculations
  - Example: An Invoice
  - XPath Dialog
Example: An Invoice

The Invoice.sps example in the (My) Documents folder, C:\Documents and Settings \<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial \Auto-Calculations\, demonstrates how Auto-Calculations can be used for the following purposes:

- Counting nodes
- Selecting a node based on input from the Authentic View user
- Updating the content of a node with the result of an Auto-Calculation
- Using the result of one Auto-Calculation in another Auto-Calculation

In the example file, the Auto-Calculations have been highlighted with a yellow background color (see screenshot below).

Counting nodes

In the Invoice example, each product in the list is numbered according to its position in the list of products that a customer has ordered (Product 1, Product 2, etc). This numbering is achieved with an Auto-Calculation (screenshot below).

In this particular case, the XPath expression position() would suffice to obtain the correct numbering. Another useful way to obtain the position of a node is to count the number of preceding siblings and add one. The XPath expression would be: count(preceding-sibling::Product)+1. The latter approach could prove useful in contexts where using the position() function is difficult to use or cannot be used. You can test this Auto-Calculation in the example file by deleting products, and/or adding and deleting new products.

Selecting a node based on user input

In the Invoice example, the user selects the category of product (Book, CD, DVD, or Electronics) via a combo box. This selection is entered in the //Product/Category node in the XML.
document. An Auto-Calculation then uses this value to reference a "lookup table" in the XML document and identify the node holding the VAT percentage for this product category. The XPath expression of this Auto-Calculation is:

for $i in Category return /Invoice/Categories/Category[. = $i]/@rate.

The VAT percentage is displayed at the Auto-Calculation location in the output. In the Invoices example, the lookup table is stored in the same XML document as the invoice data. However, such a table can also be stored in a separate document, in which case it would be accessed using the doc() function of XPath 2.0. To test this Auto-Calculation, change the product type selection in any product's Category combo box; the VAT value will change accordingly (Book=10%; CD=15%; DVD=15%; Electronics=20%).

### Updating content of a node with the result of an Auto-Calculation

The VAT percentage obtained by the Auto-Calculation from the lookup is dynamic and stored temporarily in memory (for use in Authentic View). The result of the Auto-Calculation can, however, be stored in the VAT node in the XML document. This has two advantages: (i) the content of the VAT node does not have to be entered by the user; it is entered automatically by the Auto-Calculation; (ii) each time the lookup table is modified, the changes will be reflected in the VAT node when Invoice.xml is opened in Authentic. To cause an Auto-Calculation to update a node, select the Auto-Calculation in Design View, and in the Properties window (screenshot below), click the Authentic | Update XML Node button. In the dialog that pops up, select the VAT node, and then click OK.

In Authentic View, when the user selects a different product category in the combo box, the Auto-Calculation obtains the VAT percentage by referencing the lookup table, displays the VAT percentage, and updates the VAT node.

### Using an Auto-Calculation-updated node in another Auto-Calculation

The VAT percentage, obtained by the Auto-Calculation described above, is required to calculate the gross price (net price + VAT amount) of each product. The formula to use would be derived as follows:
Gross Price = Net Price + VAT-amount

Since \[ \text{VAT-amount} = \text{Net Price} \times \text{VAT-percentage} \div 100 \]
Gross Price = Net Price + (Net Price \times \text{VAT-percentage} \div 100)

The net price of a product is obtained from the PriceNet node. The VAT percentage is calculated by an Auto-Calculation as described above, and this Auto-Calculation updates the VAT node. The content of the VAT node can now be used in an Auto-Calculation to generate the gross price. The XPath expression to do this would be:

\[ \text{PriceNet} + (\text{PriceNet} \times (\text{VAT} \div 100)) \]

The XPath expression can be viewed and edited in the Properties window. You can test the Auto-Calculation for the gross price by changing either the price or product category of any product. Notice that the gross price (price including VAT) of the product also changes.

<table>
<thead>
<tr>
<th>Product 6:</th>
<th>A Short History of the American Century</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net price:</td>
<td>€ 20.00</td>
</tr>
<tr>
<td>Category:</td>
<td>DVD</td>
</tr>
<tr>
<td>VAT:</td>
<td>15%</td>
</tr>
<tr>
<td>Price including VAT:</td>
<td>€ 23</td>
</tr>
</tbody>
</table>

**Price Total:** € 358.9

In the Invoice SPS, the gross price Auto-Calculation updates the PriceGross node in the XML document.

The updated PriceGross nodes can now be used in an Auto-Calculation that sums up the prices of all purchased products. The XPath expression would be: `sum(Order/Product/PriceGross)`. In the Invoice SPS, this Auto-Calculation updates the PriceTotal node. You can test this Auto-Calculation by modifying the prices of individual products and seeing the effect on the price total.

---

**An Auto-Calculation Exercise**

Now add two Auto-Calculation components to the SPS yourself.

1. Create an Auto-Calculation that calculates a volume discount for the entire invoice. If the order amount (price total) exceeds Euro 100, Euro 300, or Euro 600, discounts of 5%, 10%, and 12% apply, respectively. Display the discount amount (see screenshot below) and update the DiscountAmount node in the XML document.

2. Create an Auto-Calculation that calculates the discounted bill amount. This amount would be the price total less the discount amount (as calculated in the previous Auto-Calculation). Display the bill amount (see screenshot below) and update the BillAmount node in the XML document.
Set up these components so that the Authentic View output is as shown in the screenshot below.

| Price Total: | € 358.90 |
| Less Volume Discount: | € 35.09 |
| Bill Amount: | € 323.01 |

You can see these two additional Auto-Calculations in the file `InvoiceWithDiscounts.sps`, which is in the (My) Documents folder, C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial\Auto-Calculations folder.

- See also
  - Editing and Moving Auto-Calculations
  - Updating Nodes with Auto-Calculations
  - Auto-Calculations Based on Updated Nodes
  - XPath Dialog
10.2 Conditions

You can insert conditions anywhere in the design, in both the main template and global templates. A condition is an SPS component that is made up of one or more branches, with each branch being defined by an XPath expression. For example, consider a condition composed of two branches. The XPath expression of the first branch tests whether the value of the Location attribute of the context node is "US". The XPath expression of the second branch tests whether the value of the Location attribute is "EU". Each branch contains a template—a condition template. When a node is processed with a condition, the first branch with a test that evaluates to true is executed, that is, its condition template is processed, and the condition is exited; no further branches of that condition are evaluated. In this way, you can use different templates depending on the value of a node. In the example just cited, different templates could be used for US and EU locations.

This section consists of the following topics:

- **Setting Up the Conditions**, which describes how to create a condition and its branches.
- **Editing Conditions**, about how to edit the XPath expressions of condition branches after they have been created.
- **Conditions for Specific Outputs**, which shows how conditions are used to produce different output for different output formats.
- **Conditions and Auto-Calculations**, explains usage issues when conditions and Auto-Calculations are used in combination.

---

**See also**

- **Quick Start Tutorial: Using Conditions**
Setting Up the Conditions

Setting up the condition consists of the following steps:

1. Create the condition with its first branch.
2. Create additional branches for alternative processing.
3. Create and edit the templates within the various branches of the condition.

Creating the condition with its first branch

Set up a condition as follows:

1. Place the cursor anywhere in the design or select a component and then select the menu command Insert | Condition. The Edit XPath Expression dialog pops up (screenshot below).

![Edit XPath Expression](image)

2. In the Expression pane, enter the XPath expression for the condition branch via the keyboard. Alternatively, enter the expression by double-clicking nodes, operators, and/or functions in the panes of the dialog. It is important to be aware of the context node at the insertion point; the context node is highlighted in the schema sources tree when the dialog pops up.

3. Click OK to finish. The condition is created with its first branch; the XPath expression you entered is the XPath expression of the first branch. If the condition was inserted at a text insertion point, the first branch is empty (there is no template within it; see screenshot below). If the condition was inserted with a component selected, the condition is created around the component, and that component becomes the template of the first branch.

To select the entire condition, click the cell with the question mark. To select the first branch, click the cell with the number one.
After creating a condition with one branch (which may or may not have a template within it), you can create as many additional branches as required.

### Creating additional branches
Additional branches are created one at a time. An additional branch is created via the context menu (screenshot below) and can be created in two ways: (i) without any template within it (**Add New Branch**); and (ii) with a copy of an existing template within the new branch (**Copy Branch**).

![Context Menu](image)

To create a new branch, right-click any branch of the condition and select **Add New Branch** from the context menu. The **Edit XPath Expression dialog** will pop up. After entering an XPath expression and clicking **OK**, a new empty branch is added to the condition. This is indicated in the design by a new cell being added to the condition; the new cell has a number incremented by one over the last branch prior to the addition.

To create a copy of an existing branch, right-click the branch of the condition you wish to copy and select **Copy Branch**. The **Edit XPath Expression dialog** will pop up, containing the XPath expression of the branch being copied. After modifying the XPath expression and clicking **OK**, a new branch is added to the condition. The new branch contains a copy of the template of the branch that was copied. The new branch is indicated in the design by a new cell with a number incremented by one over the last branch prior to the addition.

### The Otherwise branch
The **Otherwise** branch is an alternative catch-all to specify a certain type of processing (template) in the event that none of the defined branches evaluate to **true**. Without the **Otherwise** branch, you would either have to create branches for all possible eventualities or be prepared for the possibility that the conditional template is exited without any branch being executed.

To insert an **Otherwise** branch, use either the **Add New Branch** or **Copy Branch** commands as described above, and in the **Edit XPath Expression dialog** click the **Otherwise** check box (see screenshot below).
Moving branches up and down
The order of the branches in the condition is important, because the first branch to evaluate to true is executed and the condition is then exited. To move branches up and down relative to each other, select the branch to be moved, then right-click and select Move Branch Up or Move Branch Down.

Deleting a branch
To delete a branch, select the branch to be deleted, then right-click and select Delete Branch.

See also
- Quick Start Tutorial: Using Conditions
- Editing Conditions
**Editing Conditions**

To edit the XPath expression of a condition branch, do the following:

1. Select the condition branch (not the condition).
2. In the Properties sidebar, select condition branch in the Properties For column (screenshot below).

3. Click the **Edit** button of the XPath property in the When group of properties. This pops up the **Edit XPath Expression dialog**, in which you can edit the XPath expression for that branch of the condition.

---

**See also**

- [Quick Start Tutorial: Using Conditions](#)
- [Setting Up the Conditions](#)
Output-Based Conditions

Individual components in the document design can be processed differently for StyleVision's different output formats (Authentic View, RTF, PDF, Word 2007+ and HTML). For example, consider the case where you wish to create a link, which, in Authentic View should point to a file on a local system, but in the HTML output should point to a Web page. In this case, you can create one condition to process content for Authentic View output and a second condition to process content for HTML output. Or consider the case where you want some text to be included in the Authentic View output but not in the HTML output. A condition could be created with a branch for processing Authentic View output, and no branch for HTML output.

Note: Conditions for specific output can be placed around individual parts or components of the document, thus providing considerable flexibility in the way the different output documents are structured.

Creating conditions for specific output

To create conditions for specific output, do the following:

1. In Design View, select the component (or highlight the document part) which you wish to create differently for different output formats.
2. Right-click, and, from the context menu that pops up, select Enclose with | Output-Based Condition. This inserts the output condition with five branches, each having the same content (the selected component). Each branch represents a single output (Authentic View, RTF, PDF, Word 2007+ or HTML). To determine which branch represents which output, mouseover the branch tag or check the XPath expression of the selected branch (in the Properties sidebar, in the Condition Branch entry, click the Edit button).
3. Within each branch, define the required processing. If you wish not to have any processing for a particular output format, then delete the branch for that format (select the branch and press Delete, or select the branch and in the (right-click) context menu select Delete Branch).

Note: The output-based condition can also be created first and (static and/or dynamic) content for each branch inserted later. First insert the output-based condition at a cursor insertion point in the design. Then within the respective branches, insert the required static and/or dynamic content.

Editing the branches of an Output-Based Condition

The XPath expression of a branch of an output-based condition is $SV_OutputFormat = 'format', where format is one of the values: Authentic, RTF, PDF, Word 2007+ or HTML. You can edit the XPath expression of a condition branch (in the Properties sidebar, in the Condition Branch entry, click the Edit button). For example, you could combine the Authentic View and HTML output formats in one condition branch (using the XPath expression: $SV_OutputFormat = 'Authentic' or $SV_OutputFormat = 'HTML').
You can also (a) delete one or more branches; (b) create an otherwise branch in a condition; and (c) move the branches up or down relative to each other, thus changing the relative priority of the branches. For information on how to carry out these actions, see Setting Up the Conditions and Editing Conditions.

Using the \$SV_OutputFormat parameter

In the XSLT file generated for each output, \$SV_OutputFormat is created as a global parameter and assigned the value appropriate to that output format (that is, Authentic, RTF, PDF, Word 2007 + or HTML). This parameter can be overridden by passing another value for it to the processor at runtime. This could be useful, if, for example, you wish to create two alternative HTML output options, one of which will be selected at runtime. You could then create condition branches \$SV_OutputFormat = 'HTML-1' and \$SV_OutputFormat = 'HTML-2'. At runtime you could pass the required parameter value (HTML-1 or HTML-2) to the processor.

- See also
  - Quick Start Tutorial: Using Conditions
  - Editing Conditions
Conditions and Auto-Calculations

When using Conditions and Auto-Calculations together, there are a few issues to bear in mind. The two most fundamental points to bear in mind are:

- Only Auto-Calculations in visible conditions—that is the branch selected as true—are evaluated.
- Auto-Calculations are evaluated before Conditions.

Here are a few guidelines that summarize these issues.

1. If an Auto-Calculation updates a node, and if that node is involved in a Condition (either by being in the XPath expression of a branch or in the content of a conditional template), then keep the Auto-Calculation outside the condition if possible. This ensures that the Auto-Calculation is always visible—no matter what branch of the condition is visible—and that the node will always be updated when the Auto-Calculation is triggered. If the Auto-Calculation were inside a branch that is not visible, then it would not be triggered and the node not updated.

2. If an Auto-Calculation must be placed inside a condition, ensure (i) that it is placed in every branch of the condition, and (ii) that the various branches of the condition cover all possible conditions. There should be no eventuality that is not covered by a condition in the Conditional Template; otherwise there is a risk (if the Auto-Calculation is not in any visible template) that the Auto-Calculation might not be triggered.

3. If you require different Auto-Calculations for different conditions, ensure that all possible eventualities for every Auto-Calculation are covered.

4. Remember that the order in which conditions are defined in a conditional template is significant. The first condition to evaluate to true is executed. The otherwise condition is a convenient catch-all for non-specific eventualities.

See also

- Quick Start Tutorial: Using Conditions
- Editing Conditions
- Auto-Calculations
10.3 Conditional Presence

The Conditional Presence feature enables you to create certain design components if a specified condition is fulfilled. The design components that can be created conditionally are:

- Hyperlinks
- Bookmarks
- New Documents
- User-Defined Elements
- TOC Levels

Specifying conditional presence

To specify that one of these design components be created conditionally and to set the required condition, do the following:

1. Select the design component.
2. In the Common group of properties in the Properties sidebar of that design component (screenshot below), click the Edit XPath icon of the Conditional Presence property.
3. In the Edit XPath Expression dialog that pops up, enter the XPath expression that is required as the condition to be fulfilled in order for the design component to be implemented in the output.
4. Click OK to finish.

Note: When the condition is fulfilled, the design component is implemented. Otherwise it is not, but all content of the design component is output—without the presence of that design component. For example, in the screenshot above, a hyperlink is created to be conditionally present. The condition tests whether the current node is not empty. If the node is not empty, then the condition test evaluates to true and the hyperlink is created. The text of the hyperlink text is derived from the content of the Hyperlink design component. (The URL of the hyperlink is specified elsewhere, in the Hyperlink group of properties.)
properties.) If the condition test evaluates to false, then the Hyperlink text (derived from the Hyperlink design component) is output, but as plain text and not as a hyperlink.

In the same way, in the case of the other design components that can be conditionally created, it is the design component itself that is conditionally created or not. The content of the design component is created in either event (whether the condition test evaluates to true or false).

See also
- Quick Start Tutorial: Using Conditions
- Conditions
10.4 Grouping

The grouping functionality is available in XSLT 2.0 and 3.0 SPSs and for HTML, RTF, PDF, and Word 2007+ output. **Grouping is not supported for Authentic View output.**

Grouping enables items (typically nodes) to be processed in groups. For example, consider an inventory of cars, in which the details of each car is held under a `car` element. If, for example, the `car` element has a `brand` attribute, then cars can be grouped by brand. This can be useful for a variety of reasons. For example:

- All cars of a single brand can be presented together in the output, under the heading of its brand name.
- Operations can be carried out within a group and the results of that operation presented separately for each group. For example, the number of models available for each brand can be listed.

Additionally, a group can be further processed in sub-groups. For example, within each brand, cars can be grouped by model and then by year.

---

**Grouping criteria**

Items can be grouped using two general criteria: (i) a grouping key, which typically tests the value of a node, and (ii) the relative position of items. The following specific grouping criteria are available:

- **group-by**, which groups items on the basis of an XPath-defined key. For example, `car` elements can be grouped on the basis of their `brand` attributes. The grouping is set on the `car` element, and an XPath expression selects the `brand` attribute.
- **group-adjacent** uses a combination of grouping-key and position criteria. All adjacent items that have the same value for the grouping key are included in one group. If the grouping-key value of an item is different from that of the previous item, then this item starts a new group.
- **group-starting-with** starts a new group when a node matches a defined XPath pattern. If a node does not match the defined XPath pattern, then it is assigned to the current group.
- **group-ending-with** ends a group when a node matches a defined XPath pattern; the matching node is the last in that group. The next node starts a new group. If a node subsequent to that which starts a group does not match the defined XPath pattern it is assigned to the current group.

---

**Creating groups**

Groups can be created on either a node or a current-group template via the context menu. To create a group, right-click the node or current-group template, and in the context menu that appears, select the **Group by** command. This pops up the Define Output Grouping dialog (screenshot below).
In the dialog, check the Enable Grouping check box, then select the required Grouping Type and, in the Match text box, enter the XPath expression that defines the grouping key (for the group-by and group-adjacent options) or the desired match pattern (for the group-starting-with and group-ending-with options). When you click OK, a dialog pops up asking whether you wish to sort the group-set alphabetically (in ascending order). You can always sort group-sets subsequently or remove such sorting subsequently. The screenshot below shows nodes and current-group templates which have had grouping added to them.
In the screenshot above, the person node has been grouped and the resulting groups sorted. For example if the person elements have been grouped by department, then the various departments can be sorted in alphabetically ascending order. The groups thus created have been further grouped by creating grouping on the current-group() template. In this way person elements can be grouped, say, first by department, and then by employment grade.

---

**Sorting groups**

After confirming a grouping definition, a pop-up asks you to confirm whether the groups should be sorted in ascending order or not. You can set sorting subsequently at any time, or modify or delete, at any time, the sorting set at this stage.

To set, modify, or delete sorting subsequently, right-click the required grouping template and select **Sort by**. This pops up the Define Output Sort Order dialog. How to use this dialog is described in the section **Sorting**. The important point to note is that to sort groups on the basis of their grouping-key, you must select the XPath function `current-grouping-key()` as the sorting key. For examples, see the files described in the following sections.

---

**Viewing and editing grouping and sorting settings**

To view and edit the grouping and sorting settings on a template, right-click the template and select **Group by** or **Sort by**, respectively. This pops up the respective dialog, in which the settings can be viewed or modified.

---

**User-defined templates**

User-defined templates are templates that are applied to items selected by an XPath expression you specify. The nodes selected by the XPath expression of a user-defined template can also be grouped. In this case, the grouping is applied on the user-defined template.

---

**See also**

- [Output Structure](#)
- [Sorting](#)
- [User-defined Templates](#)
- Example: Group-By (Persons.sps)
- Example: Group-By (Scores.sps)
Example: Group-By (Persons.sps)

The Persons.sps example is based on the Persons.xsd schema and uses Persons.xml as its Working XML File. It is located in the (My) Documents folder, C:\Documents and Settings \<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial \Grouping\Persons\. The XML document structure is as follows: an employees document element can contain an unlimited number of person employees. Each person employee is structured according to this example:

```xml
<person first="Vernon" last="Callaby" department="Administration" grade="C"/>
```

In the design we group persons according to department. Each department is represented by a separate table and the departments are sorted in ascending alphabetical order. Within each department table, persons are grouped according to grade (sorted in ascending alphabetical order) and, within each grade, persons are listed on in ascending alphabetical order of their last names.

Strategy

The strategy for creating the groups is as follows. The grouping is created on the person element with the department attribute being the grouping-key. This causes the person elements to be ordered in groups based on the value of the department attribute. (If sorting is specified, then the department groups can be organized in alphabetical order, for example, Administration first, and so on.) Since the departments are to be created as separate tables, the current-grouping (which is based on the department grouping-key) is created as a table. Now, within this grouped order of Person elements, we specify that each group must be further ordered with the grade attribute as the grouping-key.

Creating the SPS

The design was created as follows:

1. Drag the person element from the schema tree and create it as contents.
2. Right-click the person element tag and, in the context menu, select Group by.
3. In the Define Output Grouping dialog, select group-by, set the XPath expression in the Match text box to @department, and click Yes.
4. A dialog pops up asking whether the groups should be sorted. Since we wish the groups to be sorted according to the default ascending alphabetical sorting, click OK. (Sorting can always be set, modified, or deleted subsequently.)
5. Since each group (which is a department) is to be created in a separate table, create the current group as a table. Do this by right-clicking the current-group() tag (screenshot below), and selecting Change to | Table, selecting the child attributes @last and @grade as the columns of the table.
6. Re-organize the contents of the columns and cells of the table so that the first column contains \texttt{@grade} and the second column contains the \texttt{@first} and \texttt{@last} nodes (see screenshot below).

7. Within the current group, which is grouped by department, we wish to group by grade. So on the \texttt{current-group()} template, create a grouping for the \texttt{grade} attribute. Confirm the default sorting. A new \texttt{current-group()} template is created (see screenshot below).

8. Sort this current group (which is the sub-group of persons and grouped by grade), on the \texttt{last} attribute.

9. Set formatting for the table.

10. Above the table provide a heading for the table. Since each table represents a department, the name of the department can be dynamically obtained from the current context by using an Auto-Calculation with an XPath expression that calls the \texttt{current-grouping-key()} function of XPath 2.0/3.0.

11. Repeat the entire process, to create similar output, but this this time grouping persons by grade and then by department.

To view or modify the grouping or sorting of a template, right-click that template and select \texttt{Group by} or \texttt{Sort by} from the context menu. This pops up the respective dialog, in which the settings can be viewed or modified.

- \texttt{See also}
  - Output Structure
  - Sorting
  - Example: Group-By (Scores.sps)
Example: Group-By (Scores.sps)

The Scores.sps example is based on the Scores.xsd schema and uses Scores.xml as its Working XML File. It is located in the (My) Documents folder, C:\Documents and Settings \<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial \Grouping\Scores. The XML document structure is as follows: a results document element contains one or more group elements and one or more match elements. A group element contains one or more team elements, and a match element is structured according to this example:

```xml
<match group="A" date="2007-10-12">
    <team name="Brazil" for="2" points="3"/>
    <team name="Germany" for="1" points="0"/>
</match>
```

The design consists of three parts (screenshot below): (i) the match results presented by day (grouped on //match/@date); (ii) the match results presented by group (grouped on //match/@group); and (iii) group tables providing an overview of the standings by group (a dynamic table of the group element, with Auto-Calculations to calculate the required data).
Match Results: Day-by-Day

2007-10-12
Brazil - Germany 2 - 1
Italy - Holland 2 - 2

2007-10-13
Argentina - France 2 - 0
England - Spain 0 - 0

Match Results: By Group

Group A
Brazil - Germany 2 - 1
Italy - Holland 2 - 2
Brazil - Italy 1 - 2
Germany - Holland 2 - 2
Brazil - Holland 1 - 0
Germany - Italy 1 - 1

Group Tables

<table>
<thead>
<tr>
<th>Team</th>
<th>P</th>
<th>W</th>
<th>D</th>
<th>L</th>
<th>F</th>
<th>A</th>
<th>Pts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brazil</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>14</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Italy</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Germany</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Holland</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

Strategy
For the two sections containing the match results, we group matches by date and tournament-group. For members of each group (date and tournament group), we create borderless tables (for
alignment purposes). So matches played on a single date will be in a separate table, and all the match results of a single tournament group will be in a separate table (for example, Group A matches). For the group-tables section, the group element is created as a dynamic table, with Auto-Calculations providing the value of the required data.

Creating the SPS
The design was created as follows:

1. Drag the /results/match element from the schema tree and create it as contents.
2. Right-click the match element tag and, in the context menu, select Group By.
3. In the Define Output Grouping dialog, select group-by, set the XPath expression in the Match text box to @date, and click OK.
4. A dialog pops up asking whether the groups should be sorted. Since we wish the groups to be sorted according to the default ascending alphabetical sorting, click Yes. (Sorting can always be set, modified, or deleted subsequently.)
5. Since each group (which is a date) is to be created in a separate table, create the current group as a table. Do this by right-clicking the current-group() tag, selecting Change to | Table, and then selecting the descendant nodes team/@name and team/@for as the columns of the table (see screenshot below).

6. Set a hyphen in each cell that will be output if the match is not the last in the current group. Do this by using a conditional template with a condition set to position() != last(). This provides output such as: Brazil - Germany or 2 - 1.
7. Put an Auto-Calculation in the header that outputs the current grouping key for the respective group (XPath expression: current-grouping-key()).
8. Format the table as required.
9. To group the matches by tournament group, repeat the entire process, but group matches this time on the group attribute of match.
10. For the group tables (in the third section of the design), which contain the standings of
each team in the group, create the /results/group element as a dynamic table. Add columns as required (using the Table | Append Column or Table | Insert Column commands). Set up Auto-Calculations in each column to calculate the required output (3 point for a win; 1 point for a draw; 0 points for a loss). And, finally, sort the table in descending order of total points obtained. To see the XPath expressions used to obtain these results, right-click the Auto-Calculation or sorted template, and select, respectively, the Edit XPath and Sort by commands.

See also
- Output Structure
- Sorting
- Example: Group-By (Persons.sps)
10.5 Sorting

The sorting functionality is available for HTML, RTF, PDF, and Word 2007+ output. Sorting is not supported for Authentic View output.

A set of sibling element nodes of the same qualified name can be sorted on one or more sort-keys you select. For example, all the Person elements (within, say, a Company element) can be sorted on the LastName child element of the Person element. The sort-key must be a node in the document, and is typically a descendant node (element or attribute) of the element node being sorted. In the example mentioned, LastName is the sort-key.

If there are two elements in the set submitted for sorting that have sort-key nodes with the same value, then an additional sort-key could provide further sorting. In the Person example just cited, in addition to a first sort-key of LastName, a second sort-key of FirstName could be specified. So, for Person elements with the same LastName value, an additional sort could be done on FirstName. In this way, in an SPS, multiple sort instructions (each using one sort-key) can be defined for a single sort action.

The template is applied to the sorted set and the results are sent to the output in the sorted order. Sorting is supported in the HTML, RTF, PDF, and Word 2007+ output.

User-defined templates
User-defined templates are templates that are applied to items selected by an XPath expression you specify. The nodes selected by the XPath expression of a user-defined template can also be sorted. In this case, the sorting is applied on the user-defined template.

In this section
- The sorting mechanism is described.
- An example demonstrates how sorting is used.

See also
- Creating Dynamic Tables
- XPath Dialog
- User-defined Templates
The Sorting Mechanism

Setting up a schema element node for sorting consists of two steps:

1. In Design View, select the schema element node that is to be sorted. Note that it is the instances of this element in the XML document that will be sorted. Often it might not immediately be apparent which element is to be sorted. For example, consider the structure shown in the screenshot below.

```
<newsitem>
  <dateline>
    <place>
      <city>...</city>
    </place>
  </dateline>
</newsitem>
```

Each `newsitem` has a `dateline` containing a `place` element with a `city` attribute. The `@city` nodes of all `newsitem` elements are to be output in alphabetical order. In the design, should the `@city` node be selected for sorting, or the `place`, `dateline`, or `newsitem` elements? With `@city` selected, there will be only the one `city` node that will be sorted. With `place` or `dateline` selected, again there will be just the one respective element to sort, since within their parents they occur singly. With `newsitem` selected, however, there will be multiple `newsitem` elements within the parent `newsitems` element. In this case, it is the `newsitem` element that should be sorted, using a sort-key of `dateline/place/@city`.

2. After selecting the element to sort, in the context menu (obtained by right-clicking the element selection), click the Sort Output command. This pops up the Define Output Sort Order dialog (screenshot below), in which you insert or append one or more sort instructions.

```
Define Output Sort Order

Sort nodes:  n1:Person

Match Datatype Order
LastName  Text  Ascending

OK  Cancel
```

Each sort instruction contains: (i) a sort-key (entered in the Match column); (ii) the datatype that the sort-key node should be considered to be (text or number); (iii) the order of the sorting (ascending or descending). The order in which the sort instructions are listed is significant. Sorting is carried out using each sort instruction in turn, starting with the first, and working down the list when multiple items have the same value. Any
number of sort instructions are allowed.

For an example of how sorting is used, see Example: Sorting on Multiple Sort-Keys.

User-defined templates

User-defined templates are templates that are applied to items selected by an XPath expression you specify. The nodes selected by the XPath expression of a user-defined template can also be sorted. In this case, the sorting is applied on the user-defined template.

A note about sort-keys

The XPath expression you enter for the sort-key must select a single node for each element instance—not a nodeset (XPath 1.0) or a sequence of items (XPath 2.0 and XPath 3.0); the key for each element should be resolvable to a string or number value.

In an XSLT 2.0 or 3.0 SPS, if the sort-key returns a sequence of nodes, an XSLT processing error will be returned. So, in the Person example cited above, with a context node of Person, an XPath expression such as: ../Person/LastName would return an error because this expression returns all the LastName elements contained in the parent of Person (assuming there is more than one Person element). The correct XPath expression, with Person as the context node, would be: LastName (since there is only one LastName node for each Person element).

In XSLT 1.0, the specification requires that when a nodeset is returned by the sort-key selector, the text value of the first node is used. StyleVision therefore returns no error if the XPath expression selects multiple nodes for the sort-key; the text of the first node is used and the other nodes are ignored. However, the first node selected might not be the desired sort-key. For example, the XPath expression ../Person/LastName of the example described above would not return an error. But neither would it sort, because it is the same value for each element in the entire sort loop (the text value of the first LastName node). An expression of the kind: location/@*, however, would sort, using the first attribute of the location child element as the sort-key. This kind of expression, however, is to be avoided, and a more precise selection of the sort-key (selecting a single node) is advised.

See also

- Example: Sorting on Multiple Sort-Keys
- User-defined Templates
- Creating Dynamic Tables
- XPath Dialog
Example: Sorting on Multiple Sort-Keys

In the simple example below (available in the (My) Documents folder, \Documents and Settings\<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial\Sorting\SortingOnTwoTextKeys.sps), team-members are listed in a table. Each member is listed with first name, last name, and email address in a row of the table. Let us say we wish to sort the list of members alphabetically, first on last name and then on first name. This is how one does it.

When the list is unsorted, the output order is the order in which the member elements are listed in the XML document (screenshot below, which is the HTML output).

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrew</td>
<td>Beninck</td>
<td><a href="mailto:a.bentinck@nanonull.com">a.bentinck@nanonull.com</a></td>
</tr>
<tr>
<td>Nadia</td>
<td>Edwards</td>
<td><a href="mailto:n.edwards@nanonull.com">n.edwards@nanonull.com</a></td>
</tr>
<tr>
<td>John</td>
<td>Edwards</td>
<td><a href="mailto:j.edwards@nanonull.com">j.edwards@nanonull.com</a></td>
</tr>
<tr>
<td>Janet</td>
<td>Ashc</td>
<td><a href="mailto:j.ashe@nanonull.com">j.ashe@nanonull.com</a></td>
</tr>
</tbody>
</table>

In Design View, right-click the member element (highlighted blue in screenshot below), and from the context menu that appears, select the Sort Output command.

This pops up the Define Output Sort Order dialog (screenshot below). Notice that the element selected for sorting, member, is named at the Sort Nodes entry. This node is also the context node for XPath expressions to select the sort-key. Click the Add Row button (at left of pane toolbar) to add the first sort instruction. In the row that is added, enter an XPath expression in the Match column to select the node last. Alternatively, click the Build button to build the XPath expression. The Datatype column enables you to select how the sort-key content is to be evaluated: as text or as a number. The Order column lists the order of the sort: ascending or descending. Select Text and Ascending. Click OK to finish.
In Design View, the member tag displays an icon indicating that it contains a sort filter. The HTML output of the team-member list, sorted on last name, is shown below. Notice that the two Edwards are not alphabetically sorted (Nadia is listed before John, which is the order in the XML document). A second sort-key is required to sort on first name.

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janet</td>
<td>Ashe</td>
<td><a href="mailto:j.ashe@nanonull.com">j.ashe@nanonull.com</a></td>
</tr>
<tr>
<td>Andrew</td>
<td>Bentyck</td>
<td><a href="mailto:a.bentinck@nanonull.com">a.bentinck@nanonull.com</a></td>
</tr>
<tr>
<td>Nadia</td>
<td>Edwards</td>
<td><a href="mailto:n.edwards@nanonull.com">n.edwards@nanonull.com</a></td>
</tr>
<tr>
<td>John</td>
<td>Edwards</td>
<td><a href="mailto:j.edwards@nanonull.com">j.edwards@nanonull.com</a></td>
</tr>
</tbody>
</table>

In Design View, right-click the member tag and select the Sort Output command from the context menu. The Define Output Sort Order dialog pops up with the last sort instruction listed. To add another sort instruction, append a new row and enter the first element as its sort-key (screenshot below). Click OK to finish.
In the HTML output, the list is now sorted alphabetically on last name and then first name.

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janet</td>
<td>Ashe</td>
<td><a href="mailto:j.ashe@nanonull.com">j.ashe@nanonull.com</a></td>
</tr>
<tr>
<td>Andrew</td>
<td>Bentinck</td>
<td><a href="mailto:a.bentinck@nanonull.com">a.bentinck@nanonull.com</a></td>
</tr>
<tr>
<td>John</td>
<td>Edwards</td>
<td><a href="mailto:j.edwards@nanonull.com">j.edwards@nanonull.com</a></td>
</tr>
<tr>
<td>Nadia</td>
<td>Edwards</td>
<td><a href="mailto:n.edwards@nanonull.com">n.edwards@nanonull.com</a></td>
</tr>
</tbody>
</table>

See also

- Creating Dynamic Tables
- XPath Dialog
- User-defined Templates
10.6 Parameters and Variables

Parameters and variables can be declared and referenced in the SPS. The difference between the two is that while a variable's value is defined when it is declared, a parameter can have a value passed to it (at run-time via the command line) that overrides the optional default value assigned when the parameter was declared.

In this section, we describe the functionality available for parameters and variables:

- **User-Declared Parameters** explains how user-defined parameters can be used in an SPS.
- **Parameters for Design Fragments** describes how parameters can be used with design fragments.
- **SPS Parameters for Sources** are a special type of parameter. They are automatically defined by StyleVision for schema sources (specifically, the Working XML Files of schemas). Since the name and value of such a parameter are known to the user, the parameter can be referenced within the SPS and a value passed to it at run-time from the command line.
- **Variables** enable you to: (i) declare a variable with a certain scope and define its value, and (ii) to reference the value of declared variables and create a template on a node or nodes selected by the variable.
- **Editable Variables in Authentic** allow the Authentic View user to edit variables and consequently to control the display of content in Authentic View.

---

See also

- SPS File Content
- SPS Parameters for Sources
- Templates and Design Fragments
User-Declared Parameters

In an SPS, user-declared parameters are declared globally with a name and a default string value. Once declared, they can be used in XPath expressions anywhere in the SPS. The default value of the parameter can be overridden for individual XSLT transformations by passing the XSLT stylesheet a new global value via StyleVision Server.

Use of parameters

User-declared parameters are useful in the following situations:

- If you wish to use one value in multiple locations or as an input for several calculations. In this case, you can save the required value as a parameter value and use the parameter in the required locations and calculations.
- If you wish to pass a value to the stylesheet at processing time. In the SPS (and stylesheet), you use a parameter with a default value. At processing time, you pass the desired value to the parameter via StyleVision Server.

Usage mechanism

Working with user-declared parameters in the SPS consists of two steps:

1. Declaring the required parameters.
2. Referencing the declared parameters.

Declaring parameters

All user-defined parameters are declared and edited in the Edit Parameters dialog (screenshot below). The Edit Parameters dialog is accessed via: the Edit | Stylesheet Parameters command and the Parameters button in the Edit Database Filters dialog (Edit | Edit DB Filter).
Declaring a parameter involves giving it a name and a string value—its default value. If no value is specified, the default value is an empty string.

To declare a parameter, do the following:

1. In the Edit Parameters dialog, append or insert a new parameter by clicking the Append or Insert buttons. A new line appears.
2. Enter the name of the parameter. Parameter names must begin with a letter, and can contain the characters A to Z, a to z, 0 to 9, and the underscore.
3. Enter a default value for that parameter. The value you enter is accepted as a text string.

You can insert any number of parameters and modify existing parameters at any time while editing either the SPS or Authentic View.

**Note:**
- The Edit Parameters dialog contains all the user-defined parameters in an SPS.
- Parameters can also be declared in the Design Overview sidebar.

---

**Referencing declared parameters**

Parameters can be referenced in XPath expressions by prefixing a $ character before the parameter name. For example, you could reference a parameter in the XPath expression of an Auto-Calculation (e.g. `concat('www.', $company, '.com')`). If your SPS is DB-based, then you can also use parameters as the values of DB Filter criteria. The DB parameters, however, are declared and edited in the Edit Parameters dialog.

**Note:** While it is an error to reference an undeclared parameter, it is not an error to declare a parameter and not reference it.
Parameters for Design Fragments

Parameters for Design Fragments enable you to define a parameter on a design fragment you have created and to give this parameter a default value. At each location where this design fragment is used in the design, you can enter a different parameter value, thus enabling you to modify the output of individual design fragments.

For example, a design fragment named `EMailAddresses` can be created with a parameter named `Domain` that has a default value of `altova.com`. Now, say this parameter is used in an Auto-Calculation in the design fragment to generate the email addresses of company employees. For the EU addresses, we could use the design fragment `EmailAddresses` and edit the value of the `Domain` parameter to be `altova.eu`. In the same way, in the template for Japanese employees, we could edit the value of the `Domain` parameter to be `altova.jp`. For the US employees of the company, we could leave the parameter value of `Domain` unchanged, thus generating the default value of `altova.com`.

Using parameters for design fragments consists of two parts:

1. **Defining the parameter** with a default value on the design fragment where it is created.
2. **Editing the parameter value** where the design fragment is used.

These parts are explained in detail below.

**Note:** Parameters for Design Fragments are supported in Authentic View only in the Enterprise Editions of Altova products.

---

**Defining the parameter**

Each design fragment can be assigned any number of parameters. To do this, click the Define Parameters link in the title bar of the design fragment (see screenshot below).

```xml
'EmailPerson'

Define Parameters...

On1:Person

On1:First (content) On1:First On1:Last (content) On1:Last: = (AutoCalc)

On1:Person
```

This pops up the Define Parameters for Design Fragments dialog (screenshot below). Click the **Append** or **Insert** icon at top left to add a parameter entry line. Enter or select the name, datatype, number of occurrences, and default value of the parameter. The **Occurrence** attribute of the parameter specifies the number of items returned by evaluating the XPath expression specified as the default value of the parameter. The **Occurrence** attribute is optional and is, by default, none or one. You can add as many parameters as you like.
There are two types of **Delete** icon. The Delete icon to the right of each parameter entry deletes the default value of that parameter. The **Delete** icon at the top right of the pane deletes the currently highlighted parameter.

**Note:** If the SPS uses XSLT 1.0, then the XPath expression you enter must return a node-set. Otherwise an error is reported.

---

**Using the parameter**

After a design fragment has been created, it can be inserted at multiple locations in the design (by dragging it from the Design Tree or Schema Tree). The screenshot below shows the design fragment `EmailPerson`, inserted after the `n1:Name` element.

If a parameter has been defined for this design fragment, then its value can be edited for this particular usage instance of the design fragment. Do this by right-clicking the design fragment and selecting the command **Edit Parameters**. This pops up the Edit Parameters for Design Fragments dialog (screenshot below).
You can edit the value of the parameter in this dialog. Click **OK** to finish. The new parameter value will be used in this usage instance of the design fragment. If the parameter value is not edited, the original (or default) parameter value will be used.

**Note:** If XSLT 1.0 is being used, then the XPath expression must return a node-set. Otherwise an error is reported.

**See also**
- [User-Declared Parameters](#), for a description of how to use stylesheet parameters that are valid for the entire document.
- [SPS Parameters for Sources](#), for a description of how parameters are declared for sources, and how these can be used.
SPS Parameters for Sources

An SPS can have multiple schema sources, where a schema could be a DTD or XML Schema on which an XML document is based, or an XML Schema that is generated from a DB and on which the DB is based.

In each SPS, there is one main schema, and, optionally, one or more additional schemas. When you add a new schema source, StyleVision automatically declares a parameter for that schema and assigns the parameter a value that is the URI of the Working XML File you assign to that schema. In the case of DBs, StyleVision generates a temporary XML file from the DB, and sets the parameter to target the document node of this temporary XML file.

Referencing parameters for sources

Each SPS parameter for a schema source addresses the document node of an XML file corresponding to that schema. In StyleVision, the XML file for each schema is the Working XML File or the XML file generated from a DB. SPS parameters for sources can therefore be used in two ways:

1. In XPath expressions within the SPS, to locate nodes in various documents. The parameter is used to identify the document, and subsequent locator steps in the XPath expression locate the required node within that document. For example, the expression:
   \[
   \text{count($XML2//Department/Employee)}
   \]
   returns the number of Employee elements in all Department elements in the XML document that is the Working XML File assigned to the schema source designated \$XML2.

2. On the command line, the URI of another XML file can be passed as the value of an SPS parameter for sources. Of course, the new XML file would have to be based on the schema represented by that parameter. For example, if FileA.xml and FileB.xml are both valid according to the same schema, and FileA.xml is the Working XML File assigned to a schema \$XML3 used in an SPS, then when an XSLT transformation for that SPS is invoked from the command line, FileB.xml can be substituted for FileA.xml by using the parameter \$XML3="FileB.xml". You should also note that, on the command line, values should be entered for all SPS parameters for sources except the parameter for the main schema. The XML file corresponding to the main schema will be the entry point for the XSLT stylesheet, and will therefore be the XML file on which the transformation is run.

See also

- User-Declared Parameters
Variables

Using variables consists of two parts: (i) **declaring the variable**, and (ii) **using the variable**.

**Note:** Variables are supported in Authentic View only in the Enterprise Editions of Altova products.

---

**Declaring a variable**

A variable can be declared on any template included in the design. It is given a name, a datatype, and a value. Additionally, you can specify whether it is to be editable in the Enterprise editions of Authentic View. The variable will then be in scope on this template and can be used within it. To declare a variable so that it is in scope for the entire document, declare the variable on the root template. A major advantage of declaring a variable only on the template where it is needed is that XPath expressions to locate a descendant node will be simpler.

Declare a variable as follows:

1. Right-click the node template on which the variable is to be created and select the command **Define Variables**.
2. In the Define Variables dialog that appears (screenshot below), click the **Append Variable** icon in the top left of the Variables pane, then enter a variable name. The value of the variable is given via an XPath expression. If you wish to enter a string as the value of the variable (as in the first variable in the screenshot below), then enclose the string in quotation marks. In the screenshot below, the value of the `SelectGroup` variable is the empty string. Otherwise, the text will be read as a node name or a function-call.

3. Setting a variable to Editable (by checking the **Editable** check box) enables the variable to **be edited in Authentic View**. In this case, you must also set the datatype value to the correct type, such as `xs:string`. When a variable is editable, the original value set by the SPS designer can be edited when the Authentic View user makes changes to the document in Authentic View. Such changes can be the explicit editing of the variable (such as when the variable value is created as editable (contents) or an editable text.
box and this is edited by the Authentic View user), or when a node or value used in the variable's XPath expression is modified by the Authentic View user.

4. If the variable is set to Editable, then two more options relevant to Authentic View are enabled: **Undoable** and **Calc**. Checking the **Undoable** option generates an Undo step for every change made to the variable. The Authentic View user can therefore click through the Undo cycle to retrieve an earlier value of the variable. The **Calc** value can be either **Once** or **Auto**. If this option is set to **Once**, the variable value is calculated once, when the template containing the variable is evaluated. The value can only be changed when the user explicitly edits the variable (for example, if the variable is created as editable (contents) or an editable text box). On the other hand, if this option is set to **Auto**, the variable will be re-calculated also each time a node or value used in the variable's XPath expression is modified.

5. You can add as many variables as you like, but the name of a variable must not be the name of an already declared in-scope variable. To delete a variable click the **Delete** icon in the top right of the pane.

6. Click **OK** when done. The template tag will now have a $ icon to indicate that one or more variables have been declared on it.

In this way, variables can be created for each node template that is present in the design. Each of these variables will have a name and a value, and will be in scope within the template on which it was declared. To edit a variable subsequently, right-click the node template on which the variable was created and select the command **Define Variables** to access the Define Variables dialog.

---

**Using a variable**

For a variable to be used at any location, it must be in scope at that location. This means that a variable can only be used within the template on which it was defined. Variables can also be edited in Authentic View so that users can control the display. The edited value is discarded when the SPS is closed.

A variable can be used in any XPath expression, and is referenced in the XPath expression by prefixing its name with a $ symbol. For example, the XPath expression $VarName/Name selects the Name child element of the node selected by the variable named VarName.

When you enter an XPath expression in the **Edit XPath Expression** dialog, in-scope variables appear in a pop-up (see screenshot above). Selecting a variable in the pop-up and pressing **Enter** inserts the variable reference in the expression.
See also

- SPS File Content
- Templates and Design Fragments
- Node-Template Operations
Editable Variables in Authentic

Variables that are in scope can be edited in Authentic View by the Authentic View user to control the display in Authentic View. For example, if a very large XML document containing, say the personnel data of several branches of a company, is being used, the Authentic View user can be given the option of selecting one particular company branch. The Authentic View display of the XML document could in this way be restricted to the branch selected by the Authentic View user.

How it works

Three steps are involved in setting up editable variables in Authentic View (also see screenshot below):

1. The required variable is defined on the template within which it will be used. This template delimits the scope of the variable. The variable can only be used within the template on which it is in scope.

2. A User-Defined Template is created with the name of the variable. The dynamic content of this User-Defined Template will contain the value of the variable. If the (contents) placeholder or an input field is inserted in the design as the content of the User-Defined Template, then the Authentic View user can enter any content as the value of the variable. The options available to the Authentic View user can however be restricted by inserting a form control, such as a combo box, as the content of the User-Defined Template.

3. The variable can be used in an XPath expression to control the Authentic View display. For example, the variable can be used in a condition. Depending on the value of each branch of the condition, a different display can be specified. Another mechanism where a variable can be well used is in a template match expression or template filter.

Note: Note the following points:

- Since grouping and sorting are not supported in Authentic View, editable variables in Authentic View cannot be used in an SPS containing any of these features.
- Since the editable variables feature applies only to Authentic View, the design for Authentic View will need to be different than that for the other outputs. This can be designed easily using the Output-Based Conditions feature.

Example file

The file AuthenticVariables.sps in the (My) Documents folder, C:\Documents and Settings \<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial \Grouping\Scores, shows how editable variables in Authentic View can be used. The XML file contains the results of matches in a tournament. The teams participating in the tournament are divided into two groups. The editable variable enables the Authentic View user to select the group to be viewed and restrict the display to that group.

The screenshot below displays the entire SPS design. One editable variable is created and the different steps required to set it up are labeled in the screenshot below. A description of the actual steps is given below the screenshot.
The key steps in setting up the editable variable were as follows:

- On the $XML template (the root template), an editable variable called $SelectGroup is defined with a type of xs:string. This variable will be in scope for the entire template.
- On the $XML template, a non-editable variable called Groups is defined with a type of <auto>. Its purpose is to dynamically collect the distinct values of all the results/group/@id attributes. These distinct values are planned to be displayed in the dropdown list of the combo box from which the Authentic View user will select the group he or she wishes to have displayed.
- A User-Defined Template is created and given the name $SelectGroup (the name of the editable variable). The location of the User-Defined template does not matter as long as it is within the scope of the editable variable.
- Within this User-Defined Template, a combo box is inserted. The combo box uses the XPath expression $Groups, 'All' to select the entry values of the dropdown list. This XPath expression returns the sequence of items contained in the variable $Groups (which dynamically collects all the available groups), and adds an item All to the sequence returned by $Groups. The All entry item of the combo box will be used to display all groups.
- The User-Defined Template is enclosed in an output-based conditional template with the output set for Authentic View. This is because the editable variable can only be used in Authentic View.
- The next step is to use the value of the editable variable that the Authentic View user selects. This will be used to filter the display down to the group that the Authentic View user selects. It is done by specifying a filter on the results/group template. The XPath expression of this filter is:

\[
\text{if ($SelectGroup != 'All') then @id=$SelectGroup else @id}
\]

This filter expression sets up a predicate step on the group element. If the $SelectGroup variable has a value not equal to All, then the predicate step will be [@id=X], where X is the value of the $SelectGroup variable (that the Authentic View user selected in the combo box). This filter has the effect of selecting the group that has an id attribute with the value the Authentic View user selected. If the $SelectGroup variable has a value of All, then the predicate expression will select groups with any id attribute value, that is,
all groups.
- The group template is enclosed within an output-based condition, each branch of which selects a different output. Only in the Authentic View branch does the group template have the filter applied.

The Authentic View output will look like this:

![Group Tables](image)

Notice the entries in the combo box, the combo box selection of Group B, and the display limited to Group B.

- See also
  - Variables
  - User-Defined Templates
  - Node-Template Operations
  - Conditions
10.7 Table of Contents, Referencing, Bookmarks

The Table of Contents (TOC) and other referencing mechanisms work by creating anchors at the required points in the design document and then referring back to these references from TOCs, text references, auto-numbering sequences, and hyperlinks.

We will look briefly at the anchoring (or bookmarking) mechanism first and then look at the overall TOC mechanism. We do this because understanding the bookmarking mechanism first will provide a better understanding of the overall TOC mechanism.

The bookmarking mechanism

Two types of bookmarking mechanism are used: simple and complex. The complex bookmarking mechanism is the one used for creating TOCs.

- A simple bookmark is created at a point in the design document. The bookmark is given a unique name which is used as the target of links that point to it. This simple bookmarking mechanism is the mechanism used for the Bookmarks and Hyperlinks feature. (Note that hyperlinks can additionally point to URLs outside the document.)
- For more complex referencing, such as for TOCs and for the auto-numbering of document sections, building the bookmark involves two parts.

1. The design document is structured into a hierarchy of levels required for the TOC. These levels are known as TOC levels. The structuring is achieved by assigning TOC levels to different points in the document structure. TOC levels can be nested within other TOC levels so as to give the document a hierarchical TOC structure. (For example, a TOC level can be assigned to a book chapter, and another TOC level can be assigned within that level to the sections of the chapter.)
2. TOC bookmarks are created within the various TOC levels. These TOC bookmarks identify the document sections at various levels that are to go into the TOC. Additionally, each TOC bookmark must be defined to provide the text that will appear in the referencing component.

After the TOC levels and the TOC bookmarks’ reference texts have been defined, the TOC template containing the referencing components can be designed.

The overall TOC mechanism is broadly described below, under The TOC mechanism. The various referencing features are explained in detail in the rest of this section.

The TOC mechanism

If you have selected XSLT 2.0 or XSLT 3.0 (not XSLT 1.0) as the XSLT version of your SPS, you can create a table of contents (TOC)—essentially a template for the TOC—at any location in the design.

- It is recommended that the items from the design that are to be included in and linked to from the TOC are bookmarked in the design first. These items can be static content or dynamic content. In the bottom half of the screenshot below, yellow TOC bookmark tags within the header tag indicates that the header item has
been bookmarked (for inclusion in the TOC template).

- A template is created for the TOC (highlighted in screenshot below). The TOC template contains the design of the TOC; it can be located anywhere in the design. In the example shown in the screenshot below, the TOC template is located near the top of the document.

```
Initial Document Section Edit Properties Add Header/Footer
```

**Note:** Either of these two parts can be created first, or both parts can be created concomitantly. We recommend, however, that the TOC bookmarks are created before the TOC template.

The TOC is displayed in Authentic View and in the HTML, RTF, PDF, and Word 2007+ output. Also note that: (i) TOCs can be created with a flat or a hierarchical structure (with corresponding numbering), and (ii) multiple TOCs can be created within a design. As a result, a stylesheet designer can create a document with, say, one (hierarchical) TOC at the book level and others (also hierarchical) at the chapter level, plus (flat) lists of figures and tables.
Procedure for creating TOCs

Given below is one step-by-step way of creating a TOC. Items are first bookmarked for inclusion. The TOC template is constructed after that. (Alternatively, you can create the TOC template first, and then bookmark items for inclusion. Or you can create the TOC template and select items for inclusion in parallel.)

1. Make sure that XSLT 2.0 is the selected XSLT version.
2. **Structure the document in TOC levels.** If the TOC is to have multiple levels, structure the document design in a hierarchy of nested TOC levels. If the TOC is to have a flat structure (that is, one level only), then create at least one TOC level (in the document design) that will enclose the TOC bookmarks.
3. **Create one or more TOC bookmarks within each level in the document design.** The TOC bookmarks identify the components within each TOC level that are to appear in the TOC.
4. **Create a TOC template containing TOC level references (levelrefs).** The TOC template should have the required number of TOC level references (levelrefs). In the case of a multi-level TOC, the levelrefs in the TOC template should be nested (see screenshot above).
5. **Create TOC references (TOCrefs) in the TOC template.** In the TOC template, set up a TOCref for each levelref. Each TOCref will reference, by name, the TOC bookmarks within the corresponding TOC level in the document. Alternatively, the TOCref can reference TOC bookmarks in other levels.
6. **Format the TOC items.** Each text item in the TOC output is generated by a TOCref in the TOC template. TOCref definitions can specify item numbering (including hierarchical), the TOC item text, a leader, and, for paged media, a page number. Each TOCref and its individual parts can be formatted separately as required. (Note that automatic numbering can also be defined within a TOC bookmark in the main body of the document. See the section, **Auto-Numbering**, for details.)

Terminology

The names of the main TOC-related components used in the interface are given in the table below. Components have been put in two different columns according to where they occur: in the **document body**, or in the **TOC template** (which is the template that specifies the design of the actual Table of Contents and typically occurs at the beginning of the document).

- The **TOC components in the document body** mark out items that will be used in the TOC template.
- The **TOC components in the TOC template** reference the marked items in the document body. Components in the TOC template have the word ‘reference’ in their names.

<table>
<thead>
<tr>
<th>Document body</th>
<th>TOC template</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TOC level</strong>: The TOC levels structure the document in a nested hierarchy.</td>
<td><strong>Level references (levelrefs)</strong>: Correspond to the TOC-level structure defined in the document body. Enables TOCrefs in a given level to target TOC bookmarks at the corresponding level.</td>
</tr>
<tr>
<td><strong>TOC bookmark</strong>: Has a name, with which it identifies a node in the document as a TOC item.</td>
<td><strong>TOC references (TOCrefs)</strong>: References a TOC bookmark by its name.</td>
</tr>
</tbody>
</table>
See also

- Bookmarking Items for TOC Inclusion
- Creating the TOC Template
- Bookmarks and Hyperlinks
Bookmarking Items for TOC Inclusion

Bookmarking an item in the design for inclusion in a TOC consists of two steps, which can be done in any order:

1. **Structuring the design document in a hierarchy of nested TOC levels.** A TOC level can be created in the design either on a template or around a design component. In the screenshot below, a TOC level has been created on the topic template.

   ![Diagram of TOC level on topic template]

   When a level is created on a template, this is indicated by the level icon inside the start tag of the template, for example, `\(\text{\text{topic}}\)`. When a level is created around a component it is indicated by TOC level tags. In the screenshot above, the topics template component is enclosed by a level. The difference between the two ways of marking levels is explained in the section *Structuring the Design in Levels*. When the TOC template is created, it must be structured in a hierarchy of levels, with the levels in the TOC template corresponding to the levels you have created in the design. Even for TOCs with a flat structure (one level), the design must have a corresponding level.

2. **Creating a TOC bookmark** in the design with a name and TOC-item text. The TOC bookmark can either enclose or not enclose a design component; in the latter case it is empty. In the screenshot below, the TOC bookmark does not enclose a design component.

   ![Diagram of TOC bookmark in design]

   The TOC bookmark serves as an anchor in the document. In the screenshot above, the TOC bookmark (and anchor) is located at the start of para element instances. The TOC bookmark has two attributes: (i) a name that will be used to reference the TOC bookmark when creating the TOC item in the TOC template, and (ii) a text string that will be used as the text of the corresponding TOC item. How these two attributes are assigned is described in the section, *Creating TOC Bookmarks*.

---

**How bookmarked items are referenced in the TOC template**

The TOC template is structured in nested levels (called level references (levelrefs) to differentiate them from the levels created in the main body of the design template). Within each levelref,
a TOC reference (TOCref) is inserted (see screenshot below). The TOCref within a levelref references TOC bookmarks using the TOC bookmark's name. Each TOC bookmark with that name and in the corresponding level in the XML document will be created as a TOC item at this level in the TOC. For example, the TOCref indicated with the tag references all TOC bookmarks named chapters in the corresponding level in the XML document (when the scope of the TOCref has been set to current). The text attribute of the respective instantiated TOC bookmarks will be output as the text of the TOC item.

In the screenshot above of a TOC template, there are three nested levelrefs, within each of which is a TOCref that contains the template for the TOC item of that level. For example, in the first levelref, there is a TOCref that references TOC bookmarks that have a name of MyTOC. As a result, all TOC bookmarks in the first level (as structured in the design) and named MyTOC will be accessed for output at this level in the TOC. The TOCref within the second levelref also references TOC bookmarks having a name of MyTOC. As a result, all TOC bookmarks in the second level of the document and that are named MyTOC will be used for second-level items in the TOC. The third levelref works in the same way: TOC bookmarks named MyTOC that occur within the document's third level are referenced for third-level items in the TOC.

In the sub-sections of this section, we describe: (i) how the design is structured into levels, and (ii) how bookmarks are created. How the TOC template is created is described in the section, Creating the TOC Template.

See also
- Table of Contents (TOC)
- Structuring the Design in Levels
- Creating TOC Bookmarks
- Creating the TOC Template
Structuring the Design in TOC Levels

The hierarchical structure you wish to design for the TOC is specified as a set of nested levels. As such it is a hierarchical structure which, although related to the XML document structure, is separate from it. This structure is specified in the SPS document design. The TOC template that you construct will use a structure corresponding to this hierarchical structure. In the case of a TOC with a flat structure (one level only), the design document must have at least one level. If more than one level exists in the document, a flat TOC can then be created for any of these levels or for multiple levels (aggregated together as one level).

In the design, levels can be created in the main template, in global templates, or in a combination of main template and global templates. The important thing to note is that, wherever created, these levels must together, in combination, define a clear hierarchical structure.

Creating levels

Each level in the design is created separately. A level can be created on a template or around a component. In the screenshot below, one level has been created on the topic template (indicated by \[\text{\texttt{template}}\]) and another around the topics element (indicated by \[\text{\texttt{topics}}\]). The essential difference between these two ways of creating levels is that the enclose-within-a-level option \[\text{\texttt{TOC level}}\] enables levels to be created around components other than templates.

To create a level, do the following:

1. Select the component (template or other).
2. Right-click, and from the context menu select Template Serves As Level (enabled when a template is selected) or Enclose With | TOC Level. Both these options are also available in the Insert | Insert Table of Contents menu: TOC Level or Template Serves as Level.

Levels in global templates

Levels can also be set in global templates. In these cases, care must be taken to ensure that the levels created in various global templates, as well as those in the main template, together define a hierarchical structure when the SPS is executed. The screenshot below shows two levels, one in the main template (on the topic template) and one in the global template for topic (on the topic template).
In the content model represented by the screenshot above, topic is a recursive element, that is, a topic element can itself contain a descendant topic element. In the main template (the end of which is indicated by the $div$ tag), a level has been set on the first level of topic. The rest-of-contents instruction in the main template specifies that templates will be applied for all child elements of topic/body except header. This means that the global template for topic children of topic/body will be processed.

In the global template for topic, a level has been set on the topic template (indicated by $div$). This second level of the TOC hierarchy, which occurs on the second level of topic elements, is nested within the first level of the TOC hierarchy. Since this global template also has a rest-of-contents instruction, the global template for topic will be applied to all recursive topic elements, thus creating additional nested levels in the TOC hierarchy: third level, fourth level, and so on.

As a designer, you should be aware of the number of levels created in the design, because when the TOC template is constructed, you will need to explicitly specify how TOC items for each level will be selected and formatted.

**Levels in flat TOCs**

In a flat TOC hierarchy, TOC items will be output at a single level: for example, a simple list of the images in a document.

A flat hierarchy can be obtained in a number of ways.
The design document can be structured with just a single TOC level. The TOC template will then have a single levelref with a single TOC reference (TOCref) within it.

If the design document has more than one TOC level, then the TOC template could have a number of levelrefs equal to the sequential position of the TOC level being referenced. The levelref corresponding to the targeted TOC level will contain the single TOCref in the TOC template.

If the design document has more than one TOC level, then the single TOCref in the TOC template must have a scope that covers all the targeted document levels, which, in effect, will be flattened into a single level.

Let us say that we wish to gather all the images in a document in a single flat-hierarchy TOC. The document design must therefore contain at least one level, and this level must contain all the required TOC bookmarks. In the TOC template, the images to be listed are referenced in the usual way: (i) by creating a corresponding number of levelrefs; and (ii) creating a TOCref within the levelref corresponding to the targeted TOC level. The TOCref will have the name of TOC bookmarks in the targeted TOC level.

In the TOC template shown below, there is one levelref containing a TOCref that references TOC bookmarks named `images`. The scope of the TOCref has been set to `Current level and below`. As a result, all TOC bookmarks named `images` in the first level and below (that is, in the whole document) will be referenced.

```
$div
List of Images $div
$div
<images>
$div
(num-seq): (text ref)(....)(page ref)$div
<images>
```

If the design contains more than one level, and a flat TOC is required, say, for items in the second level, then the TOC template could have two levelrefs with a TOCref only within the second level (no TOCref within the first level). Alternatively, the scope property of TOCrefs can be used to specify what level/s in the design document should be looked up for bookmarks of a given name.

- See also
  - Table of Contents (TOC)
  - Bookmarking Items for TOC Inclusion
  - Creating TOC Bookmarks
  - Creating the TOC Template
  - Scope property of TOCrefs
Creating TOC Bookmarks

TOC bookmarks are created within a TOC level in the document design. They can be created in the main template and/or in global templates. A TOC bookmark serves two purposes:

- It marks a (static or dynamic) component in the design with a static name you assign. It can either enclose or not enclose a design component; in the latter case it is empty. In the output, the TOC bookmark is instantiated as an anchor identified by a name. This named anchor can be referenced by items in the TOC (template).
- A TOC bookmark also defines the text string that will be used as the text of a TOC item. This text string can be the content of child elements of the node where the marker is located, or it can be the output of an XPath expression.

You can create a TOC bookmark in two ways:

- By using the Create TOC Bookmark Wizard, which enables you to specify the TOC bookmark’s name, its text entry, whether auto-numbering should be used, and the level within which it appears.
- By inserting an empty TOC bookmark, the properties of which will be defined subsequently.

Creating the TOC bookmark with the Create TOC Bookmark Wizard

To create a TOC bookmark using the TOC Bookmark Wizard, do the following:

1. Place the cursor at the point in the design where you wish to insert the TOC bookmark. Alternatively, select the design component around which you wish to insert the TOC bookmark.
2. From the context menu (obtained by right-clicking) or from the Insert menu, select Insert Table of Contents | TOC Bookmark (Wizard). If you are enclosing an a node with a TOC Bookmark, use the command Enclose with | TOC Bookmark (Wizard). This pops up the Create Marker Wizard (screenshot below).
3. In the wizard’s first screen (screenshot above) you: (i) define the text for the TOC item; (ii) set the TOC bookmark name; and (iii) specify whether this TOC bookmark should be numbered in the output. For the text entry you can select whether the text of child elements should be used, or an XPath expression. For the name of the TOC bookmark, you can enter text directly or select from a dropdown list containing the names of already specified TOC bookmark names. When you are done, click Next.

4. In the wizard’s second screen (screenshot below), you can create a TOC level on a template if you wish to do so. Ancestor templates of the insertion point location are shown in a tree. If a template has already been created as a TOC level, this is indicated with a symbol. In the screenshot below, the symbol next to the topic template indicates that it has already been created as a level. If you wish to create an additional level on any of the ancestor templates, select that template. Alternatively, you can choose to define the level later by checking the Define Level Later check box. When you have completed making your selection, click Finish. (Note that, if a TOC level already exists on a template, selecting such a template and clicking Finish will not create a new TOC level on that template.)

On clicking Finish, a TOC bookmark will be created at the insertion point and, if it was specified in the second screen of the wizard, a TOC level will be created on one template. The TOC bookmark that has been will be in the TOC level that immediately contains it. For example, if that TOC level is the third TOC level in the TOC level hierarchy, then the inserted TOC bookmark will be in the third TOC level.

Creating a TOC bookmark
To create a TOC bookmark without attributes (TOC bookmark name, TOC item text, etc), do the following:

1. Place the cursor at the point in the design document where you wish to insert the TOC bookmark, or select the design component around which you wish to insert the TOC bookmark.
2. From the context menu (obtained by right-clicking) or from the Insert menu, select Insert.
**Table of Contents | TOC Bookmark.** A TOC bookmark is inserted. This TOC bookmark has neither a name nor a text entry. These can be defined subsequently using the Edit commands (see below).

---

**Inserting hierarchical or sequential numbering for a component**
Hierarchical or sequential numbering within the main body of the output document (not within the TOC) can be inserted within (but also outside) a TOC bookmark's tags. Right-click at the location where you wish to insert the numbering, then select Insert Table Of Contents | Hierarchical Numbering / Sequential Numbering. For example, an auto-numbering TOC bookmark that is placed around the chapter heading template will generate numbering for all the chapter headings generated by the chapter heading template.

Note that numbering is based on the structure of TOC levels. So, for example, if a chapter heading element is in the first TOC level, then the fourth chapter heading will be numbered 4 because it is the fourth instance of a chapter heading within the first TOC level. If the sections of a chapter occur within the second TOC level, then the third section of the fourth chapter will be numbered 4.3. This is because, within the first (chapter) TOC level, it is the fourth instance of a chapter, and within the second (section) TOC level (of the fourth chapter), it is the third instance of a section.

---

**Editing the name and text entry of a TOC bookmark**
The name and text entry of the TOC bookmark can be edited in the Properties window (screenshot below). To edit these properties, select the TOC bookmark, and either directly edit the property in the Property window or right-click the TOC bookmark and select the property you wish to edit.

![Properties Window](image)

The TOC bookmark has the following properties: (i) the name of the TOC bookmark group (Group); (ii) a unique ID; (iii) an option to remove the bookmark if it is not referenced; and (iv) an option (Text From) to specify the text entry, which could come from the bookmark's content or from an XPath expression.
- See also
  - Table of Contents (TOC)
  - Bookmarking Items for TOC Inclusion
  - Structuring the Design in Levels
  - Creating the TOC Template
Creating the TOC Template

The TOC template is the template that produces the table of contents in the output. It can be created anywhere within the SPS design, and multiple TOC templates can be created in a single SPS design.

The steps to create a TOC template are as follows:

1. Place the cursor at the location where the TOC template is to be inserted.
2. Click the menu command Insert | Insert Table of Contents | Table of Contents. This pops up the Create TOC Page dialog (screenshot below). (Alternatively, this command can be accessed via the context menu, which appears when you right-click.)
   
   ![Create TOC Page dialog](screenshot)

3. Enter the information requested in the dialog: (i) The name of the generated TOC page is the (TOCref) name that will be used to reference the TOC bookmarks in the design document. If you select multiple levels for the TOC (level references, to be more accurate; next option), the same TOCref name will be used in all level references (though individual TOCref names can be edited subsequently). (ii) The number of TOC level references (levelrefs) specifies how many level references the TOC is to have. (iii) For printed media, the option to output page references (i.e. page numbers) is available. (iv) The text entries in the TOC can be used as links to the TOC bookmarks.
4. Click OK to finish. The TOC template is created with the specified number of levelrefs (screenshot below; the formatting of the TOC template has been modified from that which is created initially).

   ![Created TOC template](screenshot)

Within each levelref is a TOCref having a name that identifies TOC bookmarks that are to be the TOC items for that levelref. Within each TOCref is a default template for the TOC item, which you can edit at any time.
Editing the TOC template
The following editing options are available:

- The TOC template can be dragged to another location in the SPS. Note, however, that a change of context node could affect XPath expressions within the TOC template.
- Levelrefs can be added to or deleted from the structure of the TOC template.
- The properties of individual TOC references (TOCrefs) can be edited. The name and scope of a TOCref can be changed, and you can choose whether the TOC item corresponding to the TOCref is created as a hyperlink or not.
- TOCrefs can be added to or deleted from any levelref in the TOC template.
- The TOC item within a TOCref can be formatted with CSS properties using the standard StyleVision mechanisms.
- Standard SPS features (such as images, Auto-Calculations, and block-formatting components) can be inserted anywhere in the TOC template.

See also
- Bookmarking Items for TOC Inclusion
- Level references (Levelrefs) in the TOC Template
- TOC References: Name, Scope, Hyperlink
- Formatting TOC Items
Levelrefs in the TOC Template

The TOC template is structured in level references (or levelrefs); see screenshot below. These levels are initially created when the TOC template is created, and the number of levelrefs are the number you specify in the Create TOC Page dialog.

Notice that the levelrefs are nested. For the purposes of the TOC design there is a one-to-one correspondence between the levelrefs in the TOC template and the levels in the SPS design. Thus, the first levelref of the TOC template corresponds to the first level in the SPS design, the second levelref in the TOC template to the second level in the SPS design, and so on. The TOCrefs within a given levelref of the TOC template identify TOC bookmarks within a specified scope in the SPS design. For example, a TOCref can specify that the TOCref target TOC bookmarks in the corresponding document level, or target TOC bookmarks in all document levels, or those in the current document level and lower document levels.

Inserting and removing levelrefs

Levelrefs can be inserted in or deleted from the TOC template after the TOC template has been created.

To insert a levelref around content, select the content in the TOC template around which the levelref is to be created, then, from the context menu or via the menu bar, select the command Enclose With | TOC Level Reference. You can also insert an empty levelref at the cursor insertion point with the menu command Insert | Insert Table of Contents | TOC Level Reference (also available in the context menu).

To remove a levelref from the TOC template, select the levelref to be removed and either press the Delete key or select Remove from the context menu. Note that only the levelref will be removed—not its contents.

- See also
  - Creating the TOC Template
  - Bookmarking Items for TOC Inclusion
  - TOC References: Name, Scope, Hyperlink
TOC References: Name, Scope, Hyperlink

TOC references (TOCrefs) occur within level references (levelrefs) and have four properties (see screenshot below):

- A hyperlink property which can be toggled between yes and no to specify whether the corresponding TOC items are created as hyperlinks or not.
- A group property, which is the name of the TOCref and identifies TOC bookmarks of the same name that occur within the specified scope (see below). The TOC bookmarks so identified provide the items to be included at that levelref of the TOC.
- An id to uniquely identify the TOCref.
- A scope, which specifies to which corresponding levels in the SPS design the TOCref applies. Three options are available: (i) global, (ii) current level, (iii) current level and descendant levels (see screenshot below).

To insert a TOCref, place the cursor within a levelref and, from the Insert menu or context menu, select Insert Table of Contents | TOC Reference.

To edit a TOCref property, right-click the TOCref tag in the TOC template and select the property you wish to edit (Create Hyperlink, Edit ID, Edit Group, or Edit Scope). This pops up the Properties window with the specified property selected for editing (screenshot below).

Alternatively, with the TOCref tag selected, go directly to the required property in the Properties window (TOC reference group of properties).

See also
- Bookmarking Items for TOC Inclusion
- Creating the TOC Template
- Level references (Levelrefs) in the TOC Template
Formatting TOC Items

The TOC item can contain up to four standard components, plus optional user-specified content. The four standard components are (see also screenshot below):

- the text entry of the TOC item, indicated in the TOC template by (text_ref)
- the leader between the text entry and the page number (for paged media output), indicated by (.....)
- the page reference of the TOC item (for paged media output), indicated by (page_ref)
- hierarchical or sequential numbering, indicated by (num-lvl) and (num-seq), respectively

When the TOC template is initially created, the text entry is automatically inserted within TOCrefs. If the Include Page Reference option was selected, then the leader and page reference components are also included. Subsequently, components can be inserted and deleted from the TOC item. To insert a component, place the cursor at the desired insertion point within the TOC item, and in the context menu, select Insert Table Of Contents | TOC Reference | Entry Text / Leader / Page Reference or Insert Table Of Contents | Hierarchical Numbering / Sequential Numbering as required. (Hierarchical numbering should be inserted when the design is structured into nested levels, sequential numbering when there is no hierarchy, that is, just one flat TOC level. See the note below on flat TOCs) To delete a component, select it and press the Delete key.

Additionally, you can insert static content (e.g. text) and dynamic content (e.g. Auto-Calculations) within the TOC item.

Levels in flat TOCs

In a flat TOC hierarchy, TOC items will be output at a single level: for example, a simple list of the images in a document.

A flat hierarchy can be obtained in a number of ways.

- The design document can be structured with just a single TOC level. The TOC template will then have a single levelref with a single TOC reference (TOCref) within it.
- If the design document has more than one TOC level, then the TOC template could have a number of levelrefs equal to the sequential position of the TOC level being referenced. The levelref corresponding to the targeted TOC level will contain the single TOCref in the TOC template.
- If the design document has more than one TOC level, then the single TOCref in the TOC template must have a scope that covers all the targeted document levels, which, in effect, will be flattened into a single level.

Let us say that we wish to gather all the images in a document in a single flat-hierarchy TOC. The document design must therefore contain at least one level, and this level must contain all the required TOC bookmarks. In the TOC template, the images to be listed are referenced in the usual way: (i) by creating a corresponding number of levelrefs; and (ii) creating a TOCref within the levelref corresponding to the targeted TOC level. The TOCref will have the name of TOC.
bookmarks in the targeted TOC level.

In the TOC template shown below, there is one levelref containing a TOCref that references TOC bookmarks named images. The scope of the TOCref has been set to Current level and below. As a result, all TOC bookmarks named images in the first level and below (that is, in the whole document) will be referenced.

If the design contains more than one level, and a flat TOC is required, say, for items in the second level, then the TOC template could have two levelrefs with a TOCref only within the second level (no TOCref within the first level). Alternatively, the scope property of TOCrefs can be used to specify what level/s in the design document should be looked up for bookmarks of a given name.

**Formatting the TOC item**

The TOC item can be formatted with CSS styles via the Styles sidebar. Individual TOC item components can be separately formatted by selecting the component and assigning it style properties in the Styles sidebar.

- See also
  - Working with CSS Styles
  - Creating the TOC Template
Example: Simple TOC

An example SPS file to demonstrate the basic use of TOCs, called ChaptersSimple.sps, is in the (My) Documents folder, C:\Documents and Settings\<username>\My Documents\Altova \StyleVision2015\StyleVisionExamples\Tutorial\TOC. This SPS is based on a schema that defines the content model of a large chapter-based document. The schema structure is shown in the screenshot below and can be viewed in the Schema Tree window of StyleVision when you open ChaptersSimple.sps. (A more complex TOC example based on the same schema is described in the next section of this documentation, Example: Hierarchical and Sequential TOCs.)

![Schema Tree](image)

The document element is `helpproject`, which contains a child `topics` element. The `topics` element can contain an unlimited number of `topic` elements, each of which can in turn contain descendant `topic` elements. The first level of `topic` elements can be considered to be the chapters of the document, while descendant `topic` elements are sections, sub-sections, and so on.

This SPS creates a TOC, located at the top of the document, which lists the names of each chapter (the first-level topics). Creating the TOC involves three steps:

1. **Structuring the design in TOC levels**: One or more levels are inserted in the design document to structure the (output) document hierarchically. This hierarchical structure will be the one that the TOC reflects. In our current example, to keep things simple, only one TOC level has been created—on the Topic template. Because there is only one level in
the design, the TOC template—when it is created subsequently—can have only one level in its structure (i.e. one level reference).

2. Creating TOC bookmarks: A TOC bookmark is created within the TOC level that was created in Step 1 (in the design document). This enables TOC references in the TOC template (which will be created in the next step) to point back to this TOC bookmark. The TOC bookmark also specifies the text that will appear in the TOC item that points to it.

3. Creating the TOC template: This is the template that creates the TOC in the document. It is structured into level references (levelrefs), which must correspond to the structure of TOC levels in the design document. For example if there are three nested levelrefs in the TOC template, then the design document must have at least three nested levels. In this example we have a single levelref to correspond to the single TOC level in the design document. It is within the levelref that the TOC reference (TOCref) is placed. It is this TOCref that generates the TOC items for this level in the TOC.

---

**SPS structure and levels**

Look at the structure of the design in the SPS. Notice that the main template (with the green $XML tags) contains the TOC. The rest of the main template specifies, through the rest-of-contents instruction, that global and default templates are to be applied. The rest of the SPS design—outside the main template and after it—consists of global templates.

The TOC definitions (TOC levels and TOC bookmarks in the design) are in the global template for topic (screenshot below). In this global template a condition has been inserted to separate topic elements according to how many ancestor topic elements each has, thus providing separate processing (within different conditional branches) for chapters, sections, and sub-sections.

```
<topic (complex)
  $div
  (num-lvl): (content) $div
  $para
  (header)
  (rest-of-contents)
  $body

The screenshot above shows the contents of the first conditional branch, for first-level, chapter-type topic elements. Note that a TOC level has been created on the template start-tag of this topic element. In the other two conditional branches no TOC level has been created on the topic template. As a result, the document has been assigned only one TOC level, and this is at the level of the first-level (chapter-type) topic element.

---

**Creating the TOC bookmark**

A TOC bookmark (yellow tags in screenshot below) has been created within the header descendant element of topic (but outside the para element). This TOC bookmark serves as an
anchor for every top-level, chapter-type topic element.

The properties of the TOC bookmark can be edited in the Properties sidebar (screenshot below).

The Group property sets the TOC bookmark group (and is the name of the TOC bookmark). In our example, we have specified the value MyTOC for this property. The bookmark group will be referenced in the TOC when it is created, and it enables different TOC groups to be specified within the same level. The ID property enables unique IDs to be specified for the bookmark instances created. The Remove if not referenced property is an option to remove the bookmark if it is not referenced. The Text From property specifies the text entry that will be used as the text of the TOC item in the TOC. The text could come from the bookmark's content (the content between the start and end tags of the bookmark in the design) or from an XPath expression. In our example, an XPath expression is used which returns the header text, respectively, of each first-level topic element.

TOC template

Inside the TOC template (screenshot below), a single Level Reference (levelref) has been inserted. This levelref corresponds to the TOC Level assigned on the first-level, chapter-type topic element in the design (see 'SPS Structure and Levels' above).

Within this levelref, a TOC reference (TOCref) has been inserted. This TOCref has been set to select bookmarks (i) that are in the bookmark group named MyTOC (see 'Creating the TOC bookmark' above), and (ii) that are within the scope of the current level only. These settings can be made in the Properties sidebar when the TOCref is selected, or by right-clicking the
TOCref in the design and selecting the relevant editing command from the context menu.

The appearance of the TOC item is specified within the TOCref tags of the TOC. The numbering format, the text, the leader, and the page reference can be inserted by right-clicking within the TOCref tags and selecting the component to insert from the context menu. Each of these components can be edited by selecting it in the design and modifying its properties in the Properties sidebar.

- See also
  - Marking Items for TOC Inclusion
  - Creating the TOC Template
Example: Hierarchical and Sequential TOCs

An example SPS file to demonstrate the use of TOCs, called Chapters.sps, is in the (My) Documents folder, C:\Documents and Settings\username\My Documents\Altova \StyleVision2015\StyleVisionExamples\Tutorial\TOC. This SPS is based on a schema that defines the content model of a large chapter-based document. The schema structure is shown in the screenshot below and can be viewed in the Schema Tree window of StyleVision when you open Chapters.sps.

The document element is helpproject, which contains a child topics element. The topics element can contain an unlimited number of topic elements, each of which can in turn contain descendant topic elements. The first level of topic elements can be considered to be the chapters of the document, while descendant topic elements are sections, sub-sections, and so on.

The SPS contains three TOCs, located at the top of the document, in the following order:

1. Chapters at a glance, which lists the names of each chapter (the first-level topics).
2. Chapters and their sections, which lists each chapter with its descendants sections (first-level topics, plus each topic's hierarchy of sub-topics down to the lowest-level topic, which in the accompanying XML document, chapters.xml, is the third-level topic)
3. List of images, which is a flat list of all images in the document (except the first), listed by file name.
SPS structure

Before considering the TOCs in detail, take a look at the structure of the design. Notice that the main template (with the green $XML tags) contains the TOCs. The rest of the main template specifies, through the rest-of-contents instruction, that global and default templates are to be applied.

The TOC definitions are in the global templates for topic and image. In the global template for topic (screenshot below), a TOC level has been created on the topic element, and a TOC bookmark has been created within the header child element (but outside the para element).

Since the topic element is recursive, the TOC level and the TOC bookmark will also recurse. This means that, at the first recursion, a new hierarchically subordinate TOC level and and a new TOC bookmark is created. This process continues for each descendant topic, thus creating a hierarchy of descendant TOC levels, each with a corresponding TOC bookmark. Since the formatting of the header (the topic title) for each TOC level is to be different, we have enclosed each level within a separate branch of a condition with three branches. Each branch tests for the TOC level at which a topic occurs: first, second, or third level.

Notice that hierarchical numbering (num-lvl) has been inserted within the level. This is done by right-clicking at the required location and selecting Insert Table of Contents | Hierarchical Numbering. The effect is to insert the correct hierarchical number before each topic title in the document's text flow, for example, 3.1 or 4.2.3.

TOC descriptions

Given below is a brief description of each TOC and the points to note about them.

Chapters at a glance: Select the TOC bookmark in the global template for topic. In the Properties sidebar (screenshot below), notice that the entry text has been set to be constructed using an XPath expression. When you click the Edit button in the value field of the Text from property, you will see that the XPath expression has been defined as para. This means that the contents of the para child of header (since the TOC bookmark has been inserted within the header element) will be used as the text of the TOC item.
The TOC template itself (screenshot below) contains one level reference (levelref), and the TOCref within that levelref has been set to select TOC bookmarks named MyTOC within the scope of the current level only—which is the first level. As a result, TOC items will be created only for first-level topics.

Notice also that the numbering has been defined as hierarchical numbering.

**Chapters and their sections:** In this TOC (screenshot below), notice that three nested levelrefs have been defined, each containing a TOCref for which the scope is the current level.

Since each TOC item is contained in a div block, formatting properties (including indentation) can be set for the block.

**List of images:** The list of images is a flat list. First of all, consider within which levels images will occur in the instantiated document. The image element is a child of the para element. Since levels have been created on topic elements, image elements will occur within the first, second, and/or third levels of the document. There is therefore no need to create any level for the image.
In the global template for image, the condition (see screenshot below) enables separate processing for (i) the first image (which is presented in this example), and (ii) the other images (which, for purposes of economy, are not presented in this example).

```
image (simple)
```

Notice that the TOC bookmark is placed only within the second branch of the condition; this means that the images selected in the first branch are not bookmarked. Also, the sequential numbering (num-seq) of the images, inserted with Insert Table of Contents | Sequential Numbering, will start with the second image (because the first image is selected in the first branch of the condition). Another feature to note is that the numbering can be formatted, as has been done in this case. To see the formatting, right-click (num-seq), and select Edit Format. In the dialog box that pops up, you will see that the formatting has been set to 01, indicating that a 0 will be inserted in front of single-digit numbers.

In the TOC template for images (screenshot below), notice that there is a single TOCref identifying bookmarks named images, and that this TOCref is within a single levelref. The scope of the TOCref (editable in the Properties window when the TOCref is selected) has been set to: current level and below. The current level, determined by the levelref, is the first level. The levels below will be the second, third, and so on. In this way, all images from the first level downward are selected as items in the TOC.

```
TOC
  div List of Images div
  div (num-seq): (text ref)(....)(page ref) div
```

Since the selected numbering is sequential, the images are numbered sequentially in a flat list.

- See also
  - Marking Items for TOC Inclusion
  - Creating the TOC Template
Auto-Numbering in the Document Body

Repeating instances of a node can be numbered automatically in the main body of the document using the Auto-Numbering feature. For example, in a Book element that contains multiple Chapter elements, each Chapter element can be numbered automatically using the Auto-Numbering feature. This is an easy way to insert numbering based on the structure of the XML document.

Note: The Auto-Numbering feature refers to numbering within the main body of the document. It does not refer to numbering within tables of contents (TOCs), where numbering is considered to be a property of the TOC item.

Auto-Numbering can be either sequential (flat) or hierarchical. Sequential numbering provides ordinary numbering on a single level. Hierarchical numbering is based on the TOC-level hierarchy created in the document and creates numbering according to the element's position in the TOC-level hierarchy.

A wide variety of formatting is available for the numbers. In the case of hierarchical numbers, individual number tokens can be formatted separately. For example, a three-token number could be given the format: A.1.i., where each of the three tokens has a different number format. Number formatting is assigned differently for sequential and hierarchical numbering, and therefore have separate descriptions, each in their respective sections below.

Sequential numbering (num-seq)

Sequential (or flat) numbering can be inserted within a TOC Bookmark in the document design (see screenshot below). Create sequential numbering as follows:

1. Place the cursor within the node that has to be numbered and create the TOC bookmark (right-click, and select Insert Table of Contents | TOC Bookmark). The TOC bookmark will be created. In the screenshot below, we wish to number the topic element, so the TOC bookmark has been created within the topic element. The exact location within the topic element depends on where in the layout you want the numbering. (In the screenshot below, the numbering is placed immediately to the left of the chapter header (title).)

2. Place the cursor within the tags of the TOC bookmark, right-click, and select Insert Table of Contents | Sequential Numbering. This inserts the Auto-Numbering placeholder for sequential numbering, (num-seq) (highlighted within the TOC bookmark 'TopicHeader' in the screenshot below).

3. If the TOC bookmark is going to be referenced from within a TOC template, then you can enter TOC bookmark properties as required. However, if the TOC bookmark is going to be used only for sequential numbering, there is no need to name it. If you wish to name it, right-click it and select the Edit Group command.
In the example shown in the screenshot above, sequential numbering has been set on the topic node. The result is that each topic element receives a sequential number, as shown in the screenshot below. Note that the numbering is essentially the position of each topic element within the sequence of all sibling topic elements at that level of the XML document hierarchy.

Note: If sequential numbering must be continued on another set of nodes, then use a TOC bookmark with the same name on both nodesets.

To format the sequential numbering, right-click the num-seq placeholder and select the Edit Format command. This pops up the Format Sequential Auto-Number dialog (screenshot below).

Select the format you want from the dropdown box of the Available numbering styles combo box (see screenshot above) and click OK to apply the selected format.

Hierarchical numbering (num-lvl)
Hierarchical numbering can be inserted within a TOC level in the design. To create hierarchical numbering in a document, you must therefore first structure the document in TOC levels. Do this as described in the section Structuring the Design in Levels. The following points should be borne in mind:

- Levels must be created either on the node to be numbered or within it.
- Levels must be nested according to the hierarchy of the numbering required (see screenshot below).
- The hierarchical numbering placeholder must be inserted within the corresponding level in the design (see screenshot below).
In the screenshot above, there are two levels. The topic element is recursive, and a level has been created on two topic elements (by right-clicking the node tag and selecting Template Serves as Level). One topic element (highlighted in the screenshot above) is nested within the other. As a result, the levels also are nested. Within each level, a hierarchical numbering placeholder (num-lvl) has been inserted (right-click within the level and select Insert Table of Contents | Hierarchical Numbering).

The result of the design shown in the screenshot above will look like this.

1: Altova StyleVision 2007
2: About this Documentation
3: Introduction
   3.1: What Is an SPS?
   3.2: Product Features
   3.3: Setting up StyleVision
4: User Interface
   4.1: Main Window
   4.2: Design Entry Helpers

The first level is shown in bold, the second in normal.

To format hierarchical numbering, right-click the num-lvl placeholder and select the Edit Format command. This pops up the Format Hierarchical Auto-Number dialog (screenshot below).
First select the number of tokens in the Token combo box. This number should be the same as the number of TOC levels in the document. Each token can then be separately formatted. In the lower of the two display boxes, select the token to be formatted. (In the screenshot above, the second token has been selected.) Next, in the Formatting combo box, select the formatting style you want. In the screenshot above, lowercase formatting has been selected for the second token, and this is reflected in the display box at the top of the dialog. Additionally, levels can be omitted by entering the required number of levels to be omitted in the Omit Levels box.

Note that formatting is defined on hierarchical numbering one level at a time. So the hierarchical numbering placeholder num-lvl at each level must be separately formatted.

Click OK when done.

See also
- Table of Contents (TOC)
- Structuring the Design in Levels
- Creating TOC Bookmarks
- Creating the TOC Template
Cross-referencing

A cross-reference is a reference to another part of the document. In an SPS a cross-reference is created in two parts: First, by setting the target of the cross-reference. Second, by defining the link to the target. Setting a target consists of creating a TOC bookmark within a TOC level. The link to the target is a Text Reference within a TOC reference (TOCref). The Text Reference generates the output text and serves as the link. Building a cross-reference therefore consists of the following three steps:

Step 1: Levels
The document is structured into TOC levels as described in the section Structuring the Design in Levels. TOC levels will be used during referencing to specify the scope of the referencing. Only those TOC bookmarks having the specified name and falling within the specified scope will be targeted. In the screenshot below, a level has been created on the n1:Office element.

Step 2: Creating TOC bookmarks
Within a level, a TOC bookmark is created by placing the cursor at the required location, right-clicking, and selecting Insert Table of Contents | TOC Bookmark. The TOC bookmark is given a name and an XPath expression that generates the output text. The XPath expression will typically identify a node in the document, the contents of which is the required text.

In the screenshot below, the TOC bookmark within the n1:Name element has a name of toc3 and an XPath expression that locates the current node. This means that the output text will be the contents of the n1:Name node.

When the XML document is processed, an anchor is created for every n1:Name element. This anchor will have a text reference (the text of the cross-reference) that is the value of the n1:Name element.

Step 3: Creating TOC references
A TOC reference (TOCref) is inserted (context menu, Insert Table of Contents | TOC Reference) to create a link to the anchors generated by a TOC Bookmark.

In the screenshot above, the TOCref named toc3 (screenshot above) is within the same TOC level.
as the TOC bookmark it references (the Office level). You must also specify the scope of the TOCref. The scope specifies what TOC levels must be searched for TOC bookmarks of the same name as the TOCref. In the example shown above, the scope is the current level. This means that TOC bookmarks within the current level that have a name of toc3 are targeted by this reference.

The screenshot above shows an n1:Office template. When an n1:Office node is processed, an anchor is created with output text that is the content of the n1:Name node. This is because the TOC bookmark specifies in an XPath expression (via the Text from property of the TOC bookmark) that the contents of this node will be the output text. The TOCref in the next line identifies the anchor with the name toc3, and the Text reference component generates the output text of the link to the anchor (purple text in the screenshot below). The output will look something like this:

<table>
<thead>
<tr>
<th>Office name: Nanonull, Inc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference to the office name: Nanonull, Inc.</td>
</tr>
<tr>
<td>Office name: Nanonull Europe, AG</td>
</tr>
<tr>
<td>Reference to the office name: Nanonull Europe, AG</td>
</tr>
</tbody>
</table>

In the example above, the scope was set to the current level. There are two other possibilities for the scope: (i) a global scope, (ii) scope for the current level and below. With these options, it is possible to also target TOC Bookmarks in other levels of the design.

- **See also**
  - Table of Contents (TOC)
  - Structuring the Design in Levels
  - Creating TOC Bookmarks
  - Creating the TOC Template
Bookmarks and Hyperlinks

In the SPS document, bookmarks can be inserted anywhere within the design. These bookmarks are transformed into anchors in the output, which can be linked to from hyperlinks. Hyperlinks can not only link to bookmarks, but also to external resources like Web pages. StyleVision offers considerable flexibility in the way target URIs for hyperlinks can be built.

In this section, we describe:

- How bookmarks can be inserted in the SPS.
- How hyperlinks can be inserted in the SPS and how they link to the target pages.

Note: Links to external documents are supported in the FO spec but might not be supported by the FO processor you are using. You should check support for this feature if you are planning to use it.

- See also
  - Unparsed Entity URIs
Inserting Bookmarks

A bookmark (or anchor) can be inserted anywhere in the SPS, at a cursor insertion point or around an SPS component.

Bookmarks are created in the SPS via the Insert Bookmark dialog (screenshot below). In this dialog you define the name of the bookmark. The name can be a static name, or it can be a dynamic name that is (i) derived from XML document content, or (ii) generated arbitrarily with an XPath expression.

Creating a bookmark

To insert a bookmark, do the following:

1. Place the cursor at the location where you wish to create the bookmark.
2. Select the menu command Insert | Insert Bookmark, or right-click and select Insert | Bookmark.
3. In the Insert Bookmark dialog (screenshot below), select a tab according to whether the name of the bookmark should be static (Static tab), dynamically obtained from the XML document or arbitrarily generated from an XPath expression (Dynamic), or composed of both static and dynamic parts (Static and Dynamic). In the screenshot below a dynamic bookmark is created, which has a name that is a unique ID for each Name child of the context node.
4. Click OK. The bookmark is defined.

After a bookmark has been created, it can be linked to by a hyperlink.

Note: Bookmarks are created at the location specified in the design. If that location is within an element that repeats, a bookmark is created within each instance of that repeating element. If a static name is given, then each bookmark will have the same name. Therefore, it is better in such cases (of repeating elements) to give a dynamic name, which can be, for example, the name of a child element of the context node (the element within which the bookmark is created). If the node selected for the dynamic name might have the same content across multiple instances, then the uniqueness of the bookmark name can be ensured by using the generate-id() function to generate the name (see screenshot above). To reference such a bookmark, the same ID can be generated as the href value of a hyperlink. In this case make sure you use the fragment-identifier # in front of the generate-id() function. The XPath expression would be: concat('#', generate-id(nodeXXX)).
Modifying a bookmark
After a bookmark has been created, its name can be modified via the Edit Bookmarks dialog. This
dialog is accessed as follows:

1. Select the bookmark in the design.
2. In the Properties sidebar, click the Edit button of the Bookmark Name property
   (screenshot below) in the Bookmark group of properties. This pops up the Edit Bookmark
dialog, which is identical to the Insert Bookmark dialog described above (see screenshot
above).

3. In the Edit Bookmark dialog, edit the name of the bookmark in either the Static, Dynamic,
or Static and Dynamic tab.

Deleting a bookmark
To delete a bookmark, select it in the design and press the Delete key.

See also
- Defining Hyperlinks, for a description of how to create hyperlinks to bookmarks.
- Insert | Bookmark, for more about inserting bookmarks.
Defining Hyperlinks

Hyperlinks can be created around SPS components such as text or images. The targets of hyperlinks can be: (i) bookmarks in the SPS design, or (ii) external resources, such as web pages or email messages. In this section, we first discuss the content of the hyperlink (text, image, etc) and then the target of the hyperlink.

Creating hyperlinks

A hyperlink can be created in the following ways:

- Around text (static or dynamic), nodes, images, conditional templates, Auto-Calculations, and blocks of content or nodes; it cannot be created around a data-entry device such as an input field or combo box—though it can be created around a node or conditional template in which that data-entry device is. This is the content of the link, which, when clicked, jumps to the target of the link. To create a hyperlink around a component in the SPS, select that component and use the **Enclose With | Hyperlink** menu command.
- A new hyperlink can be inserted via the **Insert | Hyperlink** menu command. The content of the link will need to be subsequently added within the tags of the newly created hyperlink.

Defining the target of the hyperlink

The target of a link can be either:

- A **bookmark** in the same SPS design (in which case the target URI must be a fragment identifier),
- **Dynamically generated** to match bookmark anchors (these URIs are also fragment identifiers),
- An **external resource**: the URI can be static (directly entered), dynamic (taken from a node in an XML document), a combination of static and dynamic parts, or the value of an unparsed entity.

How these targets are defined is explained below. After the URI has been defined in the Insert/Edit
Hyperlink dialog, click **OK** to finish.

---

**Linking to bookmarks**

To link to a bookmark, do the following:

1. In the Static tab of the Insert Hyperlink dialog, click the **Bookmark** button. This pops up the Select Bookmark in Document dialog (*screenshot below*). The screenshot below shows two bookmarks: one static, one dynamic.

2. To select a static bookmark as the target URI, double-click the static bookmark and click **OK**. If you double-click a dynamic bookmark, you will be prompted to enter an XPath expression to match the selected dynamic bookmark (*see screenshot below*).
The **dynamic bookmark** is actually an XPath expression that generates the name of the bookmark; it is not itself the name of the bookmark. The Create Hyperlink to Dynamic Bookmark dialog displays the XPath expression of the dynamic bookmark and enables you to construct an XPath expression that will generate a name to match that of the targeted bookmark. Click **OK** when done.

---

**Linking to dynamically generated ID bookmarks**

Bookmarks can have **dynamically generated ID anchors**. If one wishes to link back to such a bookmark, the problem then is this: Since the names of dynamically generated anchors are generated at runtime and therefore unknown at design time, how is one to set the **href** value of a hyperlink that targets such an anchor? The answer is to use the **generate-id()** function once again, this time within the **href** value of the hyperlink. The key to understanding why this works lies in a property of the **generate-id()** function. In a single transformation, each time the **generate-id()** function is evaluated for a specific node, it always generates the same ID. Because of this the IDs generated in the bookmark and the hyperlink will be the same.

Two points should be borne in mind:

- Since the **generate-id()** function must be evaluated as an XPath expression, use the Dynamic tab of the Insert Hyperlink dialog (see screenshot below) to set the target of the hyperlink.
- The evaluated value of the **href** attribute must start with `#` (the fragment identifier). Consequently the XPath expression will be: `concat('#', generate-id(nodeXXX))`. Alternatively, in the Static and Dynamic tab, enter `#` in the static part of the address and `generate-id(nodeXXX)` in the dynamic part.
Linking to external resources
URIs that locate external resources can be built in the following ways:

- By entering the URI directly in the Static tab of the Insert Hyperlink dialog. For example, a link to the Altova home page (http://www.altova.com) can be entered directly in the Address input field of the Static tab.
- By selecting a node in the XML document source in the Dynamic tab of the Insert Hyperlink dialog. The node in the XML source can provide a text string that is either: (i) the URI to be targeted, or (ii) the name of an unparsed entity which has the required URI as its value. For example, the Altova website address can be contained as a text string in a node.
- By building a URI that has both static and dynamic parts in the Static and Dynamic tab of the Insert Hyperlink dialog. This can be useful for adding static prefixes (e.g. a protocol) or suffixes (e.g. a domain name). For example, email addresses can be created using a static part of mailto: and a dynamic part that takes the string content of the //Contact/@email node (the screenshot below creates a link on the contents placeholder of the //Contact/@email node, which is why the abbreviated self::node() selector has been used).

How to use unparsed entities is described in the section Unparsed Entity URIs.

Note: While linking to external documents is supported in the FO specification, an FOP limitation could result in external links not working in PDF documents created with FOP.

Authentic View modification of parent node's content
The value of the parent node of a hyperlink can be selected by the Authentic View user. The SPS can be designed to modify presentation based on what the Authentic View user selects.

Editing hyperlink properties
To edit a hyperlink, right-click either the start or end hyperlink (a) tag, and select Edit URL from the context menu. This pops up the Edit Hyperlink dialog (screenshot above). The Edit Hyperlink dialog can also be accessed via the URL property of the Hyperlink group of properties in the Properties window.
Removing and deleting hyperlinks
To delete a hyperlink, select the hyperlink (by clicking either the start or end hyperlink (a) tag), and press the **Delete** key. The hyperlink and its contents are deleted.

- **See also**
  - Inserting Bookmarks
  - Unparsed Entity URIs
  - Insert | Hyperlink
10.8 Example: Multiple Languages

Very often, documents and Authentic Forms will need to contain content in multiple languages or will require the user to choose a preferred language. StyleVision offers a range of features that can be used to achieve these goals. Given below are some possibilities, all of which are demonstrated in the Multiple Language examples in the Examples project delivered with StyleVision. (The Examples project should load automatically by default when you first start StyleVision. It can also be loaded by selecting the menu command Project | Open, and then browsing for the Examples.svp file in the folder: C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples.)

Using variables and conditions
The user's preferred language is entered in an editable variable. A condition with multiple branches maps each language to the correct language content. The user's language choice is used to select the correct conditional branch.

In the screenshot above, notice that the user's choice is entered as the value of the editable variable. The conditions in the table have two branches for the two language choices and test for the value of the editable variable. The Authentic View output is as in the screenshot below.
The above strategy is well-suited for forms in which the user selects the required language. For details, see the file, MultiLangByCondition.sps, which is in the Examples project.

**Using parameters and Auto-Calculations**

Another scenario would be one in which the same data is required to be output in different languages. A possible strategy for this requirement would be to use a parameter, the value of which triggers the required language output. The appropriate language output can be determined, for example, by means of an Auto-Calculation. The Auto-Calculation could output the appropriate content according to the value of the parameter.

In the screenshot above, the Auto-Calcs have XPath expressions of the form:

- `if ( $Language = 'E' ) then 'First' else
- `if ( $Language = 'G' ) then 'Vorname' else ''`

The value of the $Language global parameter can be modified in the SPS design or can be supplied via the command line at runtime. Multiple transformation runs can be made to output the same data in multiple languages.

For details, see the file, MultiLangByAutoCalcs.sps, which is in the Examples project.
Example files
For more examples, open the Examples project file, Examples.svp, which is in the folder: C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples.
Chapter 11

SPS File: Presentation
11 SPS File: Presentation

In the SPS design, a single set of styling features is defined for components. These styles are converted to the corresponding style markup in the respective outputs (Authentic View, HTML, RTF, PDF and Word 2007+ in the Enterprise Edition; Authentic View, HTML and RTF in the Professional Edition; HTML in the Basic Edition). Some presentation effects, notably interactive Web presentation effects (such as combo boxes and JavaScript event handlers), will by their nature not be available in paged media output (RTF, PDF and Word 2007+). In these cases, the paged media will use a suitable print rendition of the effect. For print output, however, StyleVision offers essential page definition options. These paged media options, such as page size, page layout, and headers and footers, are defined additionally to the styling of components, and will be used for RTF, PDF and Word 2007+ output alone.

Styling of SPS components

All styling of SPS components is done using CSS2 principles and syntax. Styles can be defined in external stylesheets, globally for the SPS, and locally on a component. The cascading order of CSS2 applies to the SPS, and provides considerable flexibility in designing styles. How to work with CSS styles is described in detail in the Working with CSS Styles sub-section of this section.

The values of style properties can be entered directly in the Styles or Properties sidebars, or they can be set via XPath expressions. The benefits of using XPath expressions are: (i) that the property value can taken from an XML file, and (ii) that a property value can be assigned conditionally according to a test contained in the XPath expression.

Additionally, in the SPS design, certain HTML elements are available as markup for SPS components. These predefined formats are passed to the HTML output. The formatting inherent in such markup is therefore also used to provide styling to SPS components. When CSS styles are applied to predefined formats, the CSS styles get priority over the inherent style of the predefined format. Predefined formats are described in the Predefined Formats sub-section of this section. Note that the inherent styles of predefined formats are converted to equivalent markup for RTF, PDF and Word 2007+ output.

Note: When defining CSS styles for an SPS component be aware that some styles may not, by their nature, be applicable to paged media output (RTF, PDF and Word 2007+). Also, when HTML selectors are used (in external stylesheets and global style rules), these will not be applicable to paged media output (RTF, PDF and Word 2007+). When such selectors are used, a comment is displayed next to the selector to the effect that the style will not be applied to RTF, PDF and Word 2007+ output.

Designing for paged media output

For StyleVision's paged media support (RTF, PDF and Word 2007+ outputs and XSLT stylesheets for RTF, PDF and Word 2007+), page definition and layout options are available. These options are used additionally to the component styling mechanism, and are described in the Designing Print Output sub-section of this section.
See also
- Usage Overview
- Design sidebars
11.1 Predefined Formats

StyleVision provides a number of pre-defined formats, each of which corresponds to an HTML element (screenshot below). When you apply a Predefined Format to a component in the Design, that component is marked up as a component having the corresponding HTML semantics. This has two effects:

- Formatting inherent to the selected predefined format is applied.
- The component is contained in the component type, paragraph, which makes it available for local styling by component type.

Assigning Predefined Formats

Predefined formats can be assigned by clicking Insert | Special Paragraph, and then the required format, or by selecting the required format from the Format drop-down list in the Toolbar (shown below).

Inherent styles

The predefined formats used in StyleVision have either one or both of the following two styling components:

- a text-styling component
- a spacing component.

For example, the predefined para (p) format has a spacing component only; it puts vertical space before and after the selected component, and does not apply any text styling. On the other hand, the predefined Heading 1 (h1) format has both a text-styling component and a spacing component.

The following styling points about predefined formats should be noted:

- The spacing component of a predefined format applies for any type of SPS component, but the text styling only if it can be applied. For example, if you select an image and apply a predefined format of Heading 1 (h1) to it, then the spacing component will take effect, but the text-styling component will not.
- The text-styling component of predefined formats does not apply to data-entry devices.
- Only one predefined format applies to a component at any given time.
- The Preformatted predefined format (pre) applies formatting equivalent to that applied by the pre tab of HTML: linebreaks and spacing in the text are maintained and a
monospaced font (such as Courier) is used for the display. In the case of run-on lines with no linebreaks, such as in a paragraph of text, the Preformatted (pre) predefined format will display lines of text without wrapping. If you wish to wrap the text, use the predefined format Preformatted, wrapping (pre-wrap).

---

**Defining additional styling for a predefined format**

Styles additional to the inherent styling can be defined for a predefined format by selecting it and applying a local style via the Styles sidebar.

---

**The Return key and predefined formats**

In Authentic View, when the Return key is pressed within the contents of an element having a predefined format, the current element instance and its block are terminated, and a new element instance and block are inserted at that point. This property is useful, for example, if you want the Authentic View user to be able to create a new element, say a paragraph-type element, by pressing the Return key.

- **See also**
  - Defining CSS Styles Locally
Output Escaping

A character in a text string is said to be escaped when it is written as a character reference or entity reference. Both types of references (character and entity) are delimited by an ampersand at the start and a semicolon at the end. For example:

- the hexadecimal (or Unicode) character reference of the character A is \&\#x41;
- the decimal character reference of the character A is \&\#65;
- the HTML (and XML) entity reference of the character & is \&amp;
- the hexadecimal (or Unicode) character reference of the character & is \&\#x26;
- the decimal character reference of the character & is \&\#38;
- the HTML (and XML) entity reference of the character < is \&lt;

Output escaping

Output escaping refers to the way characters that are escaped in the input are represented in the output. A character is said to be output-escaped when it is represented in the output as a character or entity reference. Note that a character can only be output-escaped when it is escaped in the input (see table below for examples). In an SPS, output-escaping can be enabled or disabled for:

- Fragments of static text,
- The contents placeholder, and
- Auto-Calculations

This is done with the disable-output-escaping attribute of the Text group of properties. The default value of this property is no, which means that output-escaping will not be disabled. So characters that are escaped in the input will be escaped in the output by default (see table below for examples).

To disable output escaping, do the following:

1. Select the (i) static text, or (ii) fragment of static text, (iii) contents placeholder, or (iv) Auto-Calculation for which you wish to disable output escaping.
2. In the Properties sidebar, select the Text group of properties for the Text item, and set the disable-output-escaping attribute to yes for the various outputs individually or for all outputs. The available values are:
   - For HTML (to set disable-output-escaping to yes for HTML output).
   - For Authentic (to set disable-output-escaping to yes for Authentic output). Note that disabling output escaping for Authentic View is enabled only in Enterprise editions of Authentic View (that is, in the Enterprise editions of StyleVision, Authentic Desktop, Authentic Browser, and XMLSpy).
   - For RTF (to set disable-output-escaping to yes for RTF output).
   - For PDF (to set disable-output-escaping to yes for PDF output).
   - For Word 2007+ (to set disable-output-escaping to yes for Word 2007+ output).
   - For all (to set disable-output-escaping to yes for all outputs).

When output escaping is disabled for a particular output format (for example, HTML output), the selected text will not be escaped in that output format, but will be escaped in the other output formats.
Given below are some examples of text with output escaping disabled and/or enabled.

<table>
<thead>
<tr>
<th>Static text</th>
<th>disable-output-escaping</th>
<th>Output text</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>no</td>
<td>&amp;</td>
</tr>
<tr>
<td>&amp;</td>
<td>yes</td>
<td>&amp;</td>
</tr>
<tr>
<td>&amp;</td>
<td>no</td>
<td>&amp;</td>
</tr>
<tr>
<td>&amp;</td>
<td>yes</td>
<td>&amp;</td>
</tr>
<tr>
<td>&lt;</td>
<td>no</td>
<td>&lt;</td>
</tr>
<tr>
<td>&lt;</td>
<td>yes</td>
<td>&lt;</td>
</tr>
<tr>
<td>&lt;</td>
<td>no</td>
<td>&lt;</td>
</tr>
<tr>
<td>&lt;</td>
<td>yes</td>
<td>&lt;</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>no</td>
<td>&lt;&lt;</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>yes</td>
<td>&lt;&lt;</td>
</tr>
</tbody>
</table>

**Note:** Disable-Output-Escaping is supported in Authentic View only in the Enterprise Editions of Altova products.

---

**Using disabled output-escaping across output formats**

If output-escaping is disabled, the text string can have significance in one output but no significance at all in another output. For example, consider the following input text, which has escaped characters (highlighted):

```
&lt;b&gt;This text is bold.&lt;/b&gt;
```

If output-escaping is disabled, this text will be output as:

```
<b>This text is bold.</b>
```

If output-escaping is disabled for HTML output and this output is viewed in a browser (as opposed to a text editor), the markup will be significant for the HTML browser and the text will be displayed in bold, like this:

```
This text is bold.
```

However, if viewed in another output format, such as PDF, the markup that was significant in HTML will not necessarily be of significance in this other output format. In the particular case cited above, the unescaped text (output escaping disabled) will be output in PDF format as is, like this:

```
&lt;b&gt;This text is bold.&lt;/b&gt;
```

As the example above demonstrates, the output text obtained by disabling output-escaping might
be interpretable as code in one output format but not in another. This should be clearly borne in mind when using the Disable-Output-Escaping property.

- **See also**
  - Inserting XML Content as Text
  - Auto-Calculations
  - Design View Symbols
Value Formatting (Formatting Numeric Datatypes)

Value Formatting enables the contents of numeric XML Schema datatype nodes (see list below) to be displayed in a format other than the lexical representation of that datatype. (For example, the lexical representation of an xs:date datatype node is YYYY-MM-DD, with an optional timezone component, such as +02:00.) The Value Formatting is displayed in Authentic View and, depending on the formatting definition, may also be available for display in the HTML, RTF, PDF and Word 2007+ output. Value Formatting can also be used to format the result of an Auto-Calculation if the result of the Auto-Calculation is in the lexical format of one of the numeric datatypes (see list below) for which Value Formatting is available.

In the sub-sections of this section, we describe:

- how the Value Formatting mechanism works, and
- the syntax for defining the Value Formatting.

Note: Value Formatting does not change the format in which the data is stored in the XML document. In the valid XML document, the data is always stored in the lexical format appropriate to the datatype of the node. Value Formatting is applied to the display in Authentic View and, optionally (if available), to the display in the output.

Numeric datatypes for which Value Formatting is available

Value Formatting is available for the following datatypes:

- xs:decimal; xs:integer; the 12 built-in types derived from xs:integer
- xs:double and xs:float when values are between and including 0.000001 and 1,000,000. Values outside this range are displayed in scientific notation (for example: 1.0E7), and cannot have Value Formatting applied to them.
- xs:date; xs:dateTime; xs:duration
- xs:gYear; xs:gYearMonth; xs:gMonth; xs:gMonthDay; xs:gDay

See also

- Formatting Dates
The Value Formatting Mechanism

Value Formatting can be applied to:

- A numeric datatype node, such as `xs:decimal` or `xs:date` that is present in the SPS as contents or an input field.
- An Auto-Calculation that evaluates to a value which has the lexical format of a numeric datatype.

Defining Value Formatting

To define Value Formatting for a node or Auto-Calculation in the SPS, do the following:

1. Select the contents placeholder or input field of the node, or the Auto-Calculation.
2. In the Properties sidebar, select the item, and then the Content group (or AutoCalc group) of properties. Now click the Edit button of the Value Formatting property. Alternatively, right-click and select Edit Value Formatting from the context menu. The Value Formatting dialog appears (screenshot below). It is different according to whether the selected component was a node or an Auto-Calculation. If the selected component was a node, then a dialog like the one below appears. The node represented in the screenshot below is of the `xs:date` datatype.

```
Value Formatting

Formats for type 'date'

- Unformatted (Standard format = YYYY-MM-DD)
- Format as XML Schema value
- Format as Inline XBRL value

Options for XML Schema value

Data type in the XML instance is: 

Format in output documents as: [DD Month YYYY]

[ ] Insert Field... [ ] Field Options... [ ] Go to Error

[ ] Apply same format to XSLT output (not all formatting options can be applied)
```

Note that the screenshot above contains the line: Formats for type 'date' and that the standard format for the `xs:date` datatype is given alongside the Unformatted check box. For a node of some other datatype, this information would be correspondingly different. If the selected component was an Auto-Calculation, the following dialog appears.
3. You now specify whether the display of the component's value is to be unformatted or formatted. If you wish to leave the output unformatted, select the Unformatted radio button. Otherwise select the Format as XML Schema Value radio button. (If the value is unformatted, the output has the standard formatting for the datatype of the selected node or the datatype of the Auto-Calculation result. If you specify Formatting as XML Schema Value for an Auto-Calculation, you have to additionally select (from a dropdown list) the datatype of the expected Auto-calculation result.

4. Enter the Value Formatting definition. This definition can be entered in three ways: (i) by selecting from a dropdown list of available options for that datatype (see the 'Format in Output Documents' input field in the screenshots above); (ii) by entering the definition directly in the input field; and (iii) by using the Insert Field and Field Options buttons to build the definition correctly. See Value Formatting Syntax for a full description of the various formatting options.

Errors in syntax
If there is an error in syntax, the following happens:

- The definition is displayed in red.
- An error message, also in red, is displayed below the input field.
- The OK button in the Value Formatting dialog is disabled.
- The Go to Error button in the Value Formatting dialog is enabled. Clicking it causes the cursor to be placed at the point in the format definition where the syntax error is.
Mismatch of data and datatype formats
If the data entered in an XML node does not match the lexical format of that node's datatype, or if the result of an Auto-Calculation does not match the lexical format of the expected datatype, then the formatting will be undefined and will not be displayed correctly in the output.

Applying Value Formatting to the output
The Value Formatting that you define applies to Authentic View, which is supported in the Enterprise and Professional editions.

Some Value Formatting definitions—not all—can also, additionally, be applied to HTML, RTF, and PDF output. To do this, check the Apply Same Format to XSLT Output check box. If this option is not checked, or if it is not available, then only Authentic View will display the Value Formatting, while the output will display the value in the standard format for the datatype of the component (the lexical format).

- **See also**
  - Value Formatting Syntax
  - Formatting Dates
Value Formatting Syntax

The syntax for Value Formatting is:

```
([prefix character/s]field[suffix character/s]{{field-
option1,field-option2,...}})+
```

where

- `prefix character/s` and `suffix character/s` are optional specifiers used to control alignment and the display of positive/negative symbols;
- `field` can be any datatype-specific formatting or text; and
- `{field-option(s)}` is an optional qualifier, that enables additional formatting options.

Explanation of definition syntax

The Value Formatting definition is constructed as follows:

- The definition is composed of one or more fields. For example, the definition `DD Month YYYY` has three fields.
- Fields can be run together, or they can be separated by the following characters: space, hyphen, comma, colon, period, or by a text string in single or double quotes. For example, in the definition: `DD-Month' in the year 'YYYY`, the fields `DD` and `Month` are separated by a hyphen, and the fields `Month` and `YYYY` are separated by a text string enclosed in single quotes.
- A field can have optional prefix and/or suffix character/s. For example: `<+###,##0.00`.
- A field can have one or more optional field-options. The field-option/s for each field must be contained in a single set of curly braces, and must follow the field without any intervening space. Multiple field-options for a single field are separated by `"`,` (comma). For example, in the definition: `DD Month{uc,ro} YYYY`, the curly-brace-enclosed `uc` and `ro` are field-options for the field `Month`.

Examples

Example of Value Formatting for an `xs:decimal` datatype:

```
"$"(##0.00)
```

Examples of the output would be:

$ 25.00
$ 25.42
$267.56

Example of Value Formatting for an `xs:date` datatype:

```
DD Month{uc,ro} YYYY
```

where `uc` and `ro` are field-options for making the `Month` field uppercase and read-only, respectively

An example of the output would be:

24 SEPTEMBER 2003
Field types

A field type represents a component of the data and the way that component is to be formatted. The formatting inherent in the field type can be modified further by prefix and suffix modifiers as well as by field options. The following tables list the available field types. Note that, in the drop-down menu of the Value Formatting dialog, there are type-specific and field-only Value Formatting definitions. You can select one of these and modify them as required by adding prefix modifiers, suffix modifiers, and/or field options.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>Space if no digit at position</td>
</tr>
<tr>
<td>0</td>
<td>Zero if no digit at position</td>
</tr>
<tr>
<td>.</td>
<td>Decimal mark</td>
</tr>
<tr>
<td>,</td>
<td>Digit group separator</td>
</tr>
<tr>
<td>Y</td>
<td>Year</td>
</tr>
<tr>
<td>y</td>
<td>Year (base = 1930); see Note below</td>
</tr>
<tr>
<td>MM</td>
<td>Month, must have length of 2</td>
</tr>
<tr>
<td>DD</td>
<td>Day, must have length of 2</td>
</tr>
<tr>
<td>W</td>
<td>Week number</td>
</tr>
<tr>
<td>d</td>
<td>Weekday number (1 to 7)</td>
</tr>
<tr>
<td>i</td>
<td>Day in the year (1 to 366)</td>
</tr>
<tr>
<td>hh</td>
<td>Hour (0 to 23), must have length of 2</td>
</tr>
<tr>
<td>HH</td>
<td>Hour (1 to 12), must have length of 2</td>
</tr>
<tr>
<td>mm</td>
<td>Minute, must have length of 2</td>
</tr>
<tr>
<td>ss</td>
<td>Second, must have length of 2</td>
</tr>
<tr>
<td>AM</td>
<td>AM or PM</td>
</tr>
<tr>
<td>am</td>
<td>am or pm</td>
</tr>
<tr>
<td>AD</td>
<td>AD or BC</td>
</tr>
<tr>
<td>ad</td>
<td>ad or bc</td>
</tr>
<tr>
<td>CE</td>
<td>CE or BCE</td>
</tr>
<tr>
<td>ce</td>
<td>ce or bce</td>
</tr>
<tr>
<td>Field Type</td>
<td>Explanation</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Weekday</td>
<td>Weekday (Sunday, Monday...)</td>
</tr>
<tr>
<td>WEEKDAY</td>
<td>Weekday (SUNDAY, MONDAY...)</td>
</tr>
<tr>
<td>weekday</td>
<td>Weekday (sunday, monday...)</td>
</tr>
<tr>
<td>Wkd</td>
<td>Weekday (Sun, Mon...)</td>
</tr>
<tr>
<td>WKD</td>
<td>Weekday (SUN, MON...)</td>
</tr>
<tr>
<td>wkd</td>
<td>Weekday (sun, mon...)</td>
</tr>
<tr>
<td>Month</td>
<td>Month (January, February...)</td>
</tr>
<tr>
<td>MONTH</td>
<td>Month (JANUARY, FEBRUARY...)</td>
</tr>
<tr>
<td>month</td>
<td>Month (january, february...)</td>
</tr>
<tr>
<td>Mon</td>
<td>Month (Jan, Feb...)</td>
</tr>
<tr>
<td>MON</td>
<td>Month (JAN, FEB...)</td>
</tr>
<tr>
<td>mon</td>
<td>Month (jan, feb...)</td>
</tr>
</tbody>
</table>

Notes on field length and entry length

The following points relating to the length of data components should be noted:

**Length of date fields:** When fields such as MM, DD, HH, hh, mm, and ss are used, they must have a length of 2 in the definition. When the y or Y fields are used, the number of y or Y characters in the definition determines the length of the output. For example, if you specify YYYY, then the output for a value of 2006 would be 006; for a definition of YYYY, it would be 002006. See also Base Year below.

**Extending field length:** The * (asterisk) symbol is used to extend the length of a non-semantic numeric field (integers, decimals, etc). In the case of decimals, it can be used on either or both sides of the decimal point. For example, the Value Formatting *0.00* ensures that a number will have zeroes as specified in the formatting if these digit locations are empty, as well as any number of digits on both sides of the decimal point.

**Entry lengths in Authentic View:** The display in Authentic View of the contents of a node is based on the Value Formatting definition for that node. Therefore, the Authentic View user will not be able to insert more characters than are allowed by the Value Formatting definition. This is a useful way to restrict input in Authentic View. Note, however, that if the length of a pre-existing value in the XML document exceeds the length specified in the formatting definition, then the entire value is displayed.

**Note:** If a field does not render any text, this might be because of your region setting in Windows. For example, Windows returns an empty string for the AM/PM field if the regional language setting is German.
Prefix and suffix modifiers
Prefix and suffix modifiers are used to modify the textual alignment and positive/negative representations of fields. The following table lists the available prefix and suffix modifiers.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Suffix</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td></td>
<td>Left aligned; default for text. For numbers, which are aligned right by default, this is significant if there are a fixed number of leading spaces.</td>
</tr>
<tr>
<td>&gt;</td>
<td></td>
<td>Right aligned; default for numbers.</td>
</tr>
<tr>
<td>?</td>
<td></td>
<td>Minus symbol adjacent to number if negative; nothing otherwise. This is the default for numbers.</td>
</tr>
<tr>
<td>&lt;?</td>
<td></td>
<td>Minus symbol left-aligned if negative; nothing otherwise. Number left-aligned, follows minus sign.</td>
</tr>
<tr>
<td>&lt;?&gt;</td>
<td></td>
<td>Minus symbol left-aligned if negative; nothing otherwise. Number right-aligned.</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>Minus symbol adjacent to number if negative; space otherwise. Located before number (prefix), after number (suffix).</td>
</tr>
<tr>
<td>&lt;-</td>
<td>&gt;-</td>
<td>Minus symbol if negative; space otherwise. Number and sign adjacent. Left-aligned (prefix); right-aligned (suffix).</td>
</tr>
<tr>
<td>&lt;-&gt;</td>
<td></td>
<td>Minus symbol left-aligned if negative; space otherwise. Number right-aligned.</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>Plus or minus sign always, located adjacent to number; before number (prefix), after number (suffix).</td>
</tr>
<tr>
<td>&lt;+</td>
<td>++</td>
<td>Plus or minus sign always, located adjacent to number; left-aligned (prefix), right-aligned (suffix).</td>
</tr>
<tr>
<td>&lt;+&gt;</td>
<td></td>
<td>Plus or minus sign always, left-aligned; number right-aligned.</td>
</tr>
<tr>
<td>(</td>
<td>)</td>
<td>Parentheses if negative; space otherwise. Adjacent to number.</td>
</tr>
<tr>
<td>&lt;(</td>
<td>)</td>
<td>Parentheses if negative; space otherwise. Adjacent to number. Left-aligned.</td>
</tr>
<tr>
<td>&lt;(&gt;</td>
<td></td>
<td>Parentheses if negative; space otherwise. Left parentheses left-aligned; number and right parentheses adjacent and right-aligned.</td>
</tr>
<tr>
<td>[</td>
<td>]</td>
<td>Parentheses if negative; nothing otherwise. Adjacent to number.</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>Extendable number of digits to left (prefix) or to right (suffix)</td>
</tr>
<tr>
<td>_</td>
<td>_</td>
<td>Space</td>
</tr>
<tr>
<td>^</td>
<td>^</td>
<td>Fill character (defined in options)</td>
</tr>
<tr>
<td>th</td>
<td></td>
<td>Ordinality of number in EN (st, nd, rd, or th)</td>
</tr>
</tbody>
</table>
Field options
Field options enable advanced modifications to be made to fields. The following options are available:

<table>
<thead>
<tr>
<th>Option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>uc</td>
<td>Make uppercase</td>
</tr>
<tr>
<td>lc</td>
<td>Make lowercase</td>
</tr>
<tr>
<td>left</td>
<td>Left aligned</td>
</tr>
<tr>
<td>right</td>
<td>Right aligned</td>
</tr>
<tr>
<td>ro</td>
<td>Read (XML) only; no editing allowed</td>
</tr>
<tr>
<td>edit</td>
<td>The field is editable (active by default)</td>
</tr>
<tr>
<td>dec=&lt;char&gt;</td>
<td>Specify a character for the decimal point (default is point)</td>
</tr>
<tr>
<td>sep=&lt;char&gt;</td>
<td>Specify a character for the digit separator (default is comma)</td>
</tr>
<tr>
<td>fill=&lt;char&gt;</td>
<td>Specify fill character</td>
</tr>
<tr>
<td>base=&lt;year&gt;</td>
<td>Base year for year fields (see note below)</td>
</tr>
<tr>
<td>pos</td>
<td>Show only positive numbers; input of negative numbers allowed</td>
</tr>
</tbody>
</table>

Field options should be used to generate number formatting for European languages, which interchange the commas and periods of the English language system: for example, 123.456,75.

The Value Formatting to use to obtain the formatting above would be: ###,###.##{dec=,,sep=.}

Notice that the field retains the English formatting, while it is the field options \texttt{dec} and \texttt{sep} that specify the decimal symbol and digit separator. If the decimal symbol and digit separator are not specified, these characters will default to decimal symbol and digit separator of the regional settings of the Windows OS (\texttt{Control Panel | All Items | Region | Format}).

Note on Base Year
When using \textit{short year formats} (such as \texttt{yy} and \texttt{YY}), the base year specifies a cut-off for a century. For example, the base year field option could be used in the definition \texttt{DD-MM-YY\{base=1940\}}. If the user enters a value that is equal to or greater than the last two digits of the base year, which are considered together as a two-digit positive integer, then the century is the same as that of the base year. If the value entered by the user is less than the integer value of the last two digits, then the century is the century of the base year plus one. For example if you
set base=1940, then if the Authentic View user enters 50, the value entered in the XML document will be 1950; if the user enters 23, the value entered in the XML document will be 2023.

Note the following points:

- Although two digits are commonly used as the short year format, one-digit and three-digit short year formats can also be used with a base year.
- Datatypes for which short year formats can be used are: xs:date, xs:dateTime, xs:gYear, and xs:gYearMonth.
- If the Value Formatting is being set for an Auto-Calculation component, make sure that the correct datatype is selected in the Value Formatting dialog. (The selected date datatype should be that of the result to which the Auto-Calculation evaluates.)
- If the yy field type is used, the default base year is 1930. Explicitly setting a base year overrides the default.
- If the yy field type is used without any base year being set, then the Authentic View user will be able to modify only the last two digits of the four-digit year value; the first two digits remain unchanged in the XML document.

See also

- The Value Formatting Mechanism
- Formatting Dates
11.4 Working with CSS Styles

The SPS design document is styled with CSS rules. Style rules can be specified:

- In **external CSS stylesheets**, External CSS stylesheets can be added via the **Design Overview** sidebar and via the **Style Repository** sidebar.
- In **global styles** for the SPS, which can be considered to be defined within the SPS and at its start. (In the HTML output these global styles are defined within the `style` child element of the `head` element.) Global styles are defined in the **Style Repository** sidebar.
- **Locally**, on individual components of the document. In the HTML output, such rules are defined in the `style` attribute of individual HTML elements. Local style rules are defined in the **Styles** sidebar.

Each of the above methods for creating styles is described in detail in the sub-sections of this section (*links above*).

---

**Terminology**

A CSS stylesheet consists of one or more style rules. A rule looks like this:

```
H1 { color: blue }
```

or

```
H1 { color: blue;
     margin-top: 16px; }
```

Each rule has a selector (in the examples above, `H1`) and a declaration (`color: blue`). The declaration is a list of properties (for example, `color`) with values (`blue`). We will refer to each property-value pair as a style definition. In StyleVision, CSS styles can be defined in the **Styles** sidebar (local styles) and **Style Repository** sidebar (global styles).

---

**Cascading order**

The cascading order of CSS applies. This means that precedence of rules are evaluated on the basis of:

1. **Origin.** External stylesheets have lower precedence than global styles, and global styles have lower precedence than local styles. External stylesheets are considered to be imported, and the import order is significant, with the latter of two imported stylesheets having precedence.
2. **Specificity.** If two rules apply to the same element, the rule with the more specific selector has precedence.
3. **Order.** If two rules have the same origin and specificity, the rule that occurs later in the stylesheet has precedence. Imported stylesheets are considered to come before the rule set of the importing stylesheet.
**CSS styles in modular SPSs**
When an SPS module is added to another SPS, then the CSS styles in the referring SPS have priority over those in the added module. When multiple modules are added, then CSS styles in those modules located relatively lower in the module list have priority. For more information about modular SPSs, see the section, Modular SPSs.

**CSS support in Internet Explorer**
Versions of Internet Explorer (IE) prior to IE 6.0 interpret certain CSS rules differently than IE 6.0 and later. As a designer, it is important to know for which version of IE you will be designing. IE 6.0 and later offers support for both the older and newer interpretations, thus enabling you to use even the older interpretation in the newer versions (IE 6.0 and later). Which interpretation is used by IE 6.0 and later is determined by a switch in the HTML document code. In an SPS, you can specify whether the HTML and Authentic View output documents should be styled according to Internet Explorer's older or newer interpretation. You should then set CSS styles according to the selected interpretation. For more details, see Properties: CSS Support.

**Note:** For more information about the CSS specification, go to [http://www.w3.org/TR/REC-CSS2/](http://www.w3.org/TR/REC-CSS2/).

- **See also**
  - Style Repository sidebar
  - Styles sidebar
  - CSS Support
  - Modular SPSs
External Stylesheets

This section describes how external CSS stylesheets can be managed from within the StyleVision GUI. It consists of the following parts:

- Adding an external CSS stylesheet to the SPS
- Viewing the contents of an external CSS stylesheet and modifying the media applicability
- Changing the precedence
- Switching between the full CSS stylesheet set and a single CSS stylesheet

External CSS stylesheets can be managed from two sidebars: the Style Repository sidebar and the Design Overview sidebar. If an aspect of the external stylesheets is viewable in both sidebars (for example, the relative precedence of multiple stylesheets), then changes made in one sidebar will automatically be reflected in the other.

---

Adding an external CSS stylesheet to the SPS

To assign an external CSS stylesheet to the SPS, do the following:

1. In Design View, select the External item in the Style Repository window (screenshot below).

![Style Repository](image)

2. Click the Add button at the top left of the Style Repository toolbar (see screenshot above).

3. In the Open dialog that pops up, browse for and select the required CSS file, then click Open. The CSS file is added to the External item as part of its tree structure (see tree listing and screenshot below).

4. To add an additional external CSS stylesheet, repeat steps 1 to 3. The new CSS stylesheet will be added to the External tree, below all previously added external CSS stylesheets.

**Note:** You can also add an external CSS stylesheet via the Design Overview sidebar.

---

Viewing and modifying the tree of external CSS stylesheets

The tree of external CSS stylesheets is structured as follows (also see screenshot below):

- CSS-1.css (File location appears on mouseover)
- Media (can be defined in Style Repository window)
- Rules (non-editable; must be edited in CSS file)
  - Selector-1
  - Property-1
The media to which that particular stylesheet is applicable can be edited in the Style Repository window. Do this by clicking the down arrow to the right of the item and selecting the required media from the dropdown list. The rules defined in the external CSS stylesheet are displayed in the Style Repository window, but cannot be edited. The Stylesheet, Rules, and individual Selector items in the tree can be expanded and collapsed by clicking the + and - symbols to the left of each item (see screenshot below).

To delete an external stylesheet, select the stylesheet and click the Reset button in the Style Repository toolbar.

Changing the precedence of the external CSS stylesheets
The external CSS stylesheets that are assigned in the Style Repository window will be imported into the HTML output file using the @import instruction. In the HTML file, this would look something like this:

```html
<html>
  <head>
    <style>
```


The order in which the files are listed in the HTML file corresponds to the order in which they are listed in the External tree of the Style Repository and in the CSS Files tree of the Design Overview sidebar. To change the order of the CSS stylesheets in the Style Repository, select the stylesheet for which the precedence has to be changed. Then use the Move Up or Move Down buttons in the Style Repository toolbar to reposition that stylesheet relative to the other stylesheets in the tree. In the Design Overview sidebar, click the Edit button of a CSS stylesheet and select the Move Up or Move Down command as required.

**Important:** Note that it is the lowermost stylesheet that has the highest import precedence, and that the import precedence decreases with each stylesheet higher in the listing order. The order of import precedence in the listing shown above is: (i) `ExternalCSS-3.css`; (ii) `ExternalCSS-2.css`; (iii) `ExternalCSS-1.css`. When two CSS rules, each in a different stylesheet, use the same selector, the rule in the stylesheet with the higher import precedence applies.

---

**Switching between all CSS files and a single CSS file**

You can choose to either: (i) let rules in all CSS files apply with the cascading rules determining precedence, or (ii) let rules in a single selected CSS file apply. You can select the option you want in the Design Overview sidebar (see screenshot below). Click the Edit button of any of the listed CSS files and select either the Mix Styles command or Select One command. This option is also available in the Style Repository (on any of the external stylesheets).
If you click the **Select One CSS File with XPath** command, a dialog pops up in which you can enter the XPath expression (screenshot below). The XPath expression must evaluate to the name of one of the CSS files in the SPS, exactly as these names are listed in the top pane of the dialog. If you enter the filename as a string, note that, like all strings in XPath expressions, the string must be entered within single quotes.

**Select CSS File Using XPath**

Enter an XPath expression that evaluates to a single CSS file that is present in the design. The possible values are shown below.

<table>
<thead>
<tr>
<th>Module.css</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main.css</td>
</tr>
</tbody>
</table>

**XPath for CSS file:** 'Main.css'  

**OK**  
**Cancel**

Valid output conditions:

- **When a single CSS file is selected**: In the Authentic and HTML outputs, all rules from the selected CSS file are applied and these rules are supported on all design components. In the RTF, PDF, and Word 2007+ outputs, only class selector rules from the selected CSS file are applied. Non-class rules are applied from all CSS files, with conflicts being resolved on the basis of the priority of the CSS file. In the RTF, PDF, and Word 2007+ outputs, these rules can be applied to the following design components: **Auto-Calculations**, the (contents) placeholder, **paragraph (block) components**, and **table cells**.

- **When styles are mixed from all CSS files**: In the Authentic and HTML outputs, all rules from all the CSS file are applied and are supported on all design components. Conflicts are resolved on the basis of the priority of the CSS file. In the RTF, PDF, and Word 2007+ outputs, only non-class selector rules are applied, with conflicts being resolved on the basis of priority.

**See also**

- **Style Repository**
- **Global Styles**
- **Local Styles**
- **CSS Support**
Global Styles

Global styles are defined for the entire SPS design in the Style Repository and are listed in the Style Repository under the Global heading. They are passed to Authentic View and the HTML output document as CSS rules. In the HTML document, these CSS rules are written within the `<html/head/style>` element.

In the Style Repository, a global style is a single CSS rule consisting of a selector and CSS properties for that selector. Creating a global style, therefore, consists of two parts:

- Adding a new style and declaring the CSS selector for it
- Defining CSS properties for the selector

Supported selectors

The following selectors are supported:

- **Universal selector**: written as `*`
- **Type selectors**: element names, such as `h1`
- **Attribute selectors**: for example, `[class=maindoc]`
- **Class selectors**: for example, `.maindoc`
- **ID selectors**: for example, `#header`

Adding a global style

To add a global style to the SPS design, do the following:

1. In Design View, select the Global item in the Style Repository window (screenshot below).

2. Click the **Add** button at the top left of the Style Repository toolbar (see screenshot above). A global style is inserted in the Global tree with a `*` selector (which selects all HTML elements); the universal selector is the default selector for each newly inserted global style.

3. To change the selector from the default universal selector, double-click the selector and edit it.
4. Now set the CSS property values for the selector. How to do this is explained in the section Setting Style Values.

5. To add another global style, repeat steps 1 to 4. The new global style will be added to the Global tree, below all previously added global styles.

Note:

- Global styles can also be inserted before a selected global style in the Global tree by clicking the Insert button in the Style Repository window. The Add and Insert buttons are also available via the context menu that appears when you right-click a global selector.
- A global style with a selector that is an HTML element can be inserted by right-clicking an item in the Global tree, then selecting Add Selector | HTML | HTMLElementName.

Editing and deleting global styles
Both, a style's selector as well as its properties can be edited in the Style Repository window.

- To edit a selector, double-click the selector name, then place the cursor in the text field, and edit.
- For information about defining and editing a style's property values, see Setting Style Values. (The style properties can be displayed in three possible views. These views and how to switch between them are described in Views of Definitions.

To delete a global style, select the style and click the Reset button in the Style Repository toolbar.

Changing the precedence of global styles
Global styles that are assigned in the Style Repository window are placed as CSS rules in the /html/head/style element. In the HTML file, they would look something like this:

```html
<html>
  <head>
    <style>
      h1 { color:blue; font-size:16pt; }
      h2 { color:blue; }
    </style>
  </head>
</html>
```
The order in which the global styles are listed in Authentic View and the HTML document corresponds to the order in which they are listed in the Global tree of the Style Repository. The order in Authentic View and the HTML document has significance. If two selectors select the same node, then the selector which occurs lower down the list of global styles has precedence. For example, in the HTML document having the partial listing given above, if there were an element `<h1 class="green">`, then three global styles match this element: that with the `h1` selector and the two `.green` class selectors. The color property of the `.green` selector with the color lime will apply because it occurs after the `.green` selector with the color green and therefore has a higher precedence. (Class selectors always have a higher precedence than node selectors, so both `.green` selectors will have a higher precedence than the `h1` selector regardless of their respective positions relevant to the `h1` selector.) The font-size of the `h1` style will, however, apply to the `<h1>` element because there is no selector with a higher precedence that matches the `<h1>` element and has a font-size property.

To change the precedence of a global style, select that style and use the Move Up and Move Down buttons in the Style Repository toolbar to reposition that global style relative to the other global styles in the tree. For example, if the `.green` global style were moved to a position before the `.red` style, then the color property of the `.red` style would have precedence over that of the `.green` style.

Note, however, that class selectors always have precedence over type selectors. So, if the selector order were changed to `.red .green h1 h2`, then `h1` and `h2` would still be green.

See also
- Style Repository
- External CSS Stylesheets
- Local Styles
- CSS Support
Local Styles

When styles are defined locally, the style rules are defined directly on the component. These local rules have precedence over both global style rules and style rules in external CSS stylesheets that select that component. Locally defined styles are CSS styles and are defined either via the Format toolbar or in the Styles sidebar. (This is as opposed to global styles, which are defined in the Style Repository sidebar.)

Local styles via the Format toolbar

You can select content in the design and apply local styles via the Format toolbar (screenshot below).

You can apply predefined HTML formatting (such as div, h1, pre, etc), text styling, background color, text alignment, lists, and hyperlinks. See the section, Format toolbar, for details.

Local styles via the Styles sidebar

Defining a style locally via the Styles sidebar consists of three parts:

1. The component to be styled is selected in Design View. Any component in the design except node tags can be styled. The component selected in Design View then appears in the Styles-For column of the Styles sidebar (see screenshot below). In the screenshot below, a content component was selected in Design View and consequently appears in the Styles-For column.
Very often, the component selected in Design View might contain other components. In this case all the components in the selection are displayed, organized by component-type, in the Styles-For column of the Styles sidebar. The screenshot below shows the different component-types contained in the Design View selection. To the left of each component-type is the number of instances of that component-type in the selection. For example, in the screenshot below, there are 16 text components and two Auto-Calculation components (among others) in the Design View selection. You can also select a range of components by keeping the Shift key pressed while selecting the second, end-of-range component.

2. After making the selection in Design View, you select, in the Styles-For column, the component-type you wish to style. If there is more than one instance of the component-type you select, then the styles you define will be applied to all these instances. So, for example, if you select the 16-texts item of the screenshot above, then the styles you define (see Step 3 below) will be applied to all 16 text components. If you wish to style, say, four of these text components differently, then you must select and style each of the four components separately. If two components of the same component-type have been
styled differently and both are selected in Design View, then the styles of both instances are displayed in the Style Definitions pane. In the screenshot above, for example, while one Auto-Calculation has a normal font-weight, the other has a bold font-weight. When the 2-Autocalcs item is selected in the Styles-For pane, both font-weights are displayed.

3. After selecting, in the Styles-For column, the component-type to style, styles are defined in the Style Definitions pane. How to do this is described in the section Setting Style Values.

See also

- Setting Style Values, for a description of how to define styling properties.
- Global Styles, for information about using global styles.
- External CSS Stylesheets, for information about using external CSS stylesheets.
- Styles sidebar, for a description of the Styles sidebar.
- CSS Support, describes settings for the output document that will adjust the level of CSS support.
Setting Style Values

For the component-type selected in the Styles-For column, style properties are defined in the Style Definitions pane of the Styles sidebar (screenshot below). You can select more than one component-type in the Styles-For column if you like—by selecting additional component-types with the Ctrl-key pressed, or by selecting a range of component-types in the Styles-For column with the Shift-key pressed. When multiple component-types are selected, any style value you define in the Style-Definitions pane is applied to all instances of all the selected component-types.

Style property groups

The available style properties in the Style-Definitions column are organized into groups as shown in the screenshot below.

The display of properties can be modified using the List Non-Empty, Expand All, and Collapse.
All toolbar buttons. Each group of style properties can be expanded to access style properties or sub-groups of style properties (see screenshot below).

**Entering style values**

Style property values (style values for short) can be entered in the following ways, all of which are show in the screenshot below:

- Entered directly in the Value column. To do this, select a property, double-click in its Value column, enter the value using the keyboard, and press Enter or click anywhere in the GUI.
- By selecting a value from the dropdown list of the combo box for that property. Click the down arrow of the combo box to drop down the list of style-value options. In the screenshot below, the options for the (background-)repeat property are displayed. Select the required value from the dropdown list.
- By using the icon on the right-hand side of the Value column for that style property. Two types of icon are available, and these are available only for properties to which they are relevant: (i) a color palette for selecting colors (in the screenshot below, see the (background-)color property), and (ii) a dialog for browsing for files (in the screenshot below, see the (background-)image property).

- Values for styles can also be assigned via an XPath expression.

**Modifying or deleting a style value**

If a style value is entered incorrectly or is invalid, it is displayed in red. To modify the value, use any of the applicable methods described in the previous section, Entering Property Values.
To delete a style value (or, in other words, to reset a style value), click the Reset button in the toolbar of the Styles sidebar. Alternatively, you can double-click in the Value column of the property, and delete the value using the **Delete** and/or **Backspace** key, and then pressing **Enter**.

- **See also**
  - Styles sidebar
  - Defining CSS Styles Locally
  - Defining CSS Styles Globally
  - External CSS Stylesheets
  - CSS Support
Style Properties Via XPath

Styles can be assigned to design components via XPath expressions. This enables style property values to be taken from XML data or from the XPath expression itself. Also, by using the `doc()` function of XPath 2.0/3.0, nodes in any accessible XML document can be addressed. Not only can style definitions be pulled from XML data; this feature also enables style choices to be made that are conditional upon the structure or content of the XML data. For example, using the `if...else` statement of XPath 2.0/3.0, two different background colors can be selected depending on the position of an element in a sequence. Thus, when these elements are presented as rows in a table, the odd-numbered rows can be presented with one background color while the even-numbered rows are presented with another (see below for example). Also, depending on the content of a node, the presentation can be varied.

Style properties for which XPath expressions are enabled

XPath expressions can be entered for the following style properties:

- All properties available in the Styles sidebar
- The Common, Event, and HTML groups of properties in the Properties sidebar

Static mode and dynamic (XPath) mode for property values

For those properties where XPath expressions are enabled, two mode are available:

- Static mode, where the value of the property is entered directly in the Value column of the sidebar. For example, for the background-color of a design component, the value `red` can be entered directly in the sidebar.
- Dynamic, or XPath mode, where an XPath expression is entered. The expression is evaluated at runtime, and the result is entered as the value of the property. For example, for the background color of a design component, the following XPath expression can be entered: `/root/colors/color1`. At runtime, the content of the node `/root/colors/color1` will be retrieved and entered as the value of the background-color property.

Switching between static and dynamic (XPath) modes

For each property for which XPath expressions are enabled, static mode is selected by default. To switch a property to dynamic (XPath) mode, select that property and click the XPath icon in the toolbar of the sidebar (screenshot below).
If a static value was present for that property, it is now cleared and the mode is switched to dynamic. The Edit XPath Expression dialog appears. It is in this dialog that you enter the XPath expression for the property. Click OK when finished.

After you enter an XPath expression for the property, an Edit XPath expression button appears in the Value column for that property (screenshot above). Click this button to subsequently edit the XPath expression. If you wish to switch back to static mode, click the XPath icon in the toolbar. This will clear the XPath expression and switch the property to static mode.

**Note:** There are two important points to note. First, only one mode (static or dynamic), and the value/expression for that mode, is active at any time. Any value/expression that previously existed for the other mode is cleared; so switching to the other mode will present that mode with an empty entry field. (In order to go back to a previous value/expression, use the Undo command.) Second, if you reselect a property after further editing the SPS, then that property will be opened in the mode it was in previously.

---

**Creating and editing the XPath definition**

The XPath definition is created and edited in the Edit XPath Expression dialog. This dialog is accessed in two ways:

- Each time you switch to the dynamic mode of a property from static mode (by clicking the XPath icon in the toolbar of the sidebar), the Edit XPath Expression dialog appears. You can now create the XPath expression. (Note that clicking the toolbar icon when already in dynamic mode switches the mode to static mode; it does not pop up the Edit XPath Expression dialog.)
- The Edit XPath Expression dialog also pops up when you click the Edit XPath Expression button in the Value field of a property that already has an XPath expression.
defined for it. The dialog will contain the already defined XPath expression for that property, which you can now edit.

After you enter or edit the XPath expression in the entry field, click OK to finish.

---

**Values returned by XPath expressions**

The most important benefits of using XPath expressions to set a property value are that: (i) the property value can be taken from an XML file (instead of being directly entered); and/or (ii) an XPath expression can test some condition relating to the content or structure of the XML document being processed, and accordingly select a value. XPath expressions return values in the following two categories:

- **XML node content**
  The XPath expression can address nodes in: (i) the XML document being processed by the SPS, or (ii) any accessible XML document. For example the expression `Format/@color` would access the `color` attribute of the `Format` child of the context node. The value of the `color` attribute will be set as the value of the property for which the XPath expression was defined. A node in some other XML document can be accessed using the `doc()` function of XPath 2.0. For example, the expression `doc('Styles.xml')//colors/color-3` would retrieve the value of the element `color-3` in the XML document `Styles.xml` and set this value as the value of the property for which the XPath expression was defined.

- **XPath expression**
  The value of the property can come from the XPath expression itself, not from the XML document. For example, the background color of an element that is being output as a row can be made to alternate depending on whether the position of the row is odd-numbered or even-numbered. This could be achieved using the XPath 2.0/3.0 expression: `if (position() mod 2 = 0) then 'red' else 'green'`. Note that the return value of this expression is either the string `red` or the string `green`, and it will be set as the value of the property for which the XPath expression was defined. In the example just cited, the property values were entered as string literals. Alternatively, they could come from an XML document, for example: `if (position() mod 2 = 0) then doc('Styles.xml')//colors/color-1 else doc('Styles.xml')//colors/color-2`. Conversely, the XPath expression could be a straightforward string, for example: `'green'`. However, this is the same as entering the static value `green` for the property.

---

**Limitations**

The Style Properties Via XPath feature has the following limitations:

1. For RTF output, **colors** are entered by StyleVision in a table in the RTF document. This table is created at the time the XSLT-for-RTF is generated (including for RTF Preview) and contains, by default, the 140 standard colors supported by most web browsers. If values for color properties are obtained from the XML data or XPath expression, and are not one of the 140 standard colors, the next best standard color will be used. For example, `#FF0001` is not one of the 140 standard colors; so the value `#FF0000` (red) will be used.

2. For RTF, PDF, and Word 2007+ output, values obtained for the **class property** from the
XML data or a literal in the XPath expression will be applied to the following design components: Auto-Calculations, the (contents) placeholder, paragraph (block) components, and table cells.

- See also
  - Working with CSS Styles
  - XPath Dialog
  - Styles
  - Properties
  - CSS Support
Composite Styles

A Composite Style is a group of CSS text-styling properties that have been associated with an attribute of an XML instance document node. Additionally, any group of CSS text-styling properties stored in the stylesheet is also considered to be a Composite Style. Composite Styles can then be specified on the following design components:

- Auto-Calculations
- The (contents) placeholder
- Paragraph (block) design elements
- Table cells

Advantages of Composite Styles

Composite Styles offer the following advantages:

- Styling properties are in the XML data and can therefore be edited by the user. In Authentic View, the RichEdit feature enables a toolbar-based, graphical editing of Composite Styles. See the section, Composite Styles in Authentic, for more information about setting up RichEdit for Composite Styles.
- The styling properties of the design components listed above can be a combination of properties stored in the XML data and properties assigned in the SPS.
- In the SPS design phase, the SPS designer can quickly switch between the multiple Composite Styles associated with an element.

Entering the Composite Style in the XML attribute

A Composite Style (composed of multiple styling properties) is entered as the attribute-value of an element in the source XML document. For example, the desc-style attribute in the XML source document listing below contains a default Composite Style:

```xml
<Desc desc-style="font-family:Verdana; font-size:12pt; color:blue">
```

You can also set more than one Composite Style on an element. In this case, each Composite Style must be entered in a separate attribute:

```xml
<Desc styleBlue="font-family:Verdana; font-size:12pt; color:blue" styleRed ="font-family:Verdana; font-size:12pt; color:red">
```

When multiple Composite Styles are available on an element, you can switch among Composite Styles when setting a value for the Composite Style property of a design component (see below).

**Note:** The attributes that will be used to access the Composite Styles must be defined in the source schema in order for the XML document to be valid.

Supported CSS text-styling properties
The following CSS styles can be used in Composite Styles:

- font-family
- font-size
- font-weight
- font-style
- color
- background-color
- text-align
- text-decoration

---

**Setting an attribute as the Composite Style value**

If you set the Composite Style of a design component to be an attribute, then the Authentic View user can edit this Composite Style. The Authentic View user can place the cursor anywhere within the text output of the design component and use the RichEdit toolbar of Authentic View to edit the Composite Style of that design component.

To set an attribute as the Composite Style of a design component, do the following:

1. In Design View, select the design component to which you wish to assign an attribute as Composite Style. In the screenshot below, the (contents) placeholder of the `Desc` element has been selected.

   ![Schema Tree and Properties Window](image)

   In Authentic View the user can now use the RichEdit toolbar to modify the Composite Style of this design component.

2. In the combo box of the Composite Style property of the Content component (see Properties sidebar at bottom right of screenshot above), the attributes of the context element are displayed. Select the attribute you wish to set as the Composite Style of the design component. (Note that there is also an empty entry in the combo box should you wish to apply no Composite Style. In this case, the RichEdit feature of Authentic View will not be enabled in the output of this design component.)

In Authentic View the user can now use the RichEdit toolbar to modify the Composite Style of this design component.

---

**Setting an XPath expression as the Composite Style value**
You can also enter an XPath expression as the value of the Composite Style property. In this case, however, since the Composite Style is stored in the SPS (not in the XML source document), the Authentic View will not be able to edit the Composite Style.

To set an XPath expression as the value of the Composite Style property, click the XPath icon in the toolbar of the Properties sidebar, and then enter the XPath expression in the XPath dialog that pops up. The XPath expression will be evaluated as an attribute value template; the returned value will be the value of an HTML style attribute (and its equivalent in non-HTML output formats).

For example, consider the following XPath expression created on the (contents) placeholder of the n1:Person element.

```
if (number(n1:Shares) gt 1000) then 'color:red' else 'color:green'
```

What this expression will do is this: If the n1:Person element has a child element n1:Shares with a number value greater than 1000, then the contents of the n1:Person element is output in red; otherwise, all n1:Person elements are output in green. The value returned by the XPath expression is passed to the output document as the value of an HTML style attribute (or its equivalent in non-HTML output formats).

In the XSLT stylesheet generated from the SPS, this XPath expression will be evaluated as an attribute value template, something like this:

```
<span style="{if (number(n1:Shares) gt 1000) then &apos;color:red&apos; else &apos;color:green&apos;}">
```

In the HTML output, one of the following lines would be generated depending on how the condition is evaluated:

```
<span style="color:red">
```

or

```
<span style="color:green">
```

**Note:** Attribute value templates are XSLT constructs that allow the value of an attribute to be read as an XPath expression. They are delimited by curly braces and allow the value of the attribute to be assigned dynamically.

**See also**
- Working with CSS Styles
- XPath Dialog
- Styles
- Properties
- CSS Support
11.5 Text-Styling Flexibility in Authentic

An SPS can be set up so that Authentic View users can also style text. They do this by selecting a text fragment in Authentic View and setting styling properties for the selected text. These styling properties can be pre-defined by you (the SPS designer) or they can be defined by the Authentic View user. In either case, the Authentic View user can be given the option of styling text in Authentic View.

The following text-styling options are available:

- **Composite Styles**: A set of CSS style properties (the Composite Style) is defined on the attribute of an element in the source XML document. By setting this attribute as the Composite Style property-value of a design component, the Composite Style becomes editable with the RichEdit feature of Authentic View. See the section Composite Styles for a description of how this mechanism works.

- **RichEdit**: If an element is created in the SPS design as a RichEdit component, then the Authentic View user can select text fragments within that element and style it using the RichEdit styling properties of Authentic View. 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. See the section RichEdit for details.

- **Text State Icons**: You (the SPS designer) can create an Authentic View toolbar button and associate this button with an element name. We call such a toolbar button a Text State Icon. When the Authentic View user selects a text fragment in Authentic View and clicks a Text State Icon, the element associated with the Text State Icon is created around the highlighted text (a Text State Icon is enabled only if the schema allows it around the selected text). Consequently, the text formatting defined for this element (in a global template for this element) will be applied to the selected text fragment.

You can use a combination of all three text-styling options in your SPS.

You should note the following points:

- All three text-styling options depend on and require styling-related elements and/or attributes to be present in the XML document. Consequently, the schema on which the SPS is based must allow the required elements and/or attributes at the hierarchical levels on which they are required.

- Text styling applied by the Authentic View user will appear in both Authentic View as well as the output formats.

**Note**: The styling options listed above are derived from styling properties stored in the XML file and are additional to styling properties assigned in the SPS.

- **See also**
  - Working with CSS Styles
  - CSS Support
Composite Styles

The Composite Styles feature gives Authentic View users the ability to style the entire output of the following design components:

- Auto-Calculations
- The (contents) placeholder
- Paragraph (block) design elements
- Table cells

The mechanism for doing this is explained below.

Note: For a further more general description of Composite Styles, see the section Composite Styles. The description in this section describes how Composite Styles can be set up so that the Authentic View user can edit them.

---

Entering the Composite Style in the XML attribute

A default Composite Style (composed of multiple styling properties) is entered as the attribute-value of an element in the source XML document. For example, the desc-style attribute in the XML source document listing below contains a default Composite Style:

```xml
<Desc desc-style="font-family:Verdana; font-size:12pt; color:blue">
```

You, the SPS designer, can now allow the Authentic View user to edit these styling properties. This gives the Authentic View user control over the styling of the design component's text output. You enable user-editing of a Composite Style by setting the Composite Style of the design component to be the attribute containing the default Composite Style (see below).

Note: The attributes that will be used to access the Composite Styles must be defined in the source schema in order for the XML document to be valid.

---

Supported CSS text-styling properties

The following CSS styles can be used in Composite Styles:

<table>
<thead>
<tr>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>font-family</td>
</tr>
<tr>
<td>font-size</td>
</tr>
<tr>
<td>font-weight</td>
</tr>
<tr>
<td>font-style</td>
</tr>
<tr>
<td>color</td>
</tr>
<tr>
<td>background-color</td>
</tr>
<tr>
<td>text-align</td>
</tr>
<tr>
<td>text-decoration</td>
</tr>
</tbody>
</table>

---

Setting the attribute as the Composite Style of a design component

If you set the Composite Style of a design component to be an attribute, then the Authentic View user can edit this Composite Style. The Authentic View user can place the cursor anywhere within the text output of the design component and use the RichEdit toolbar of Authentic View to edit the Composite Style of that design component.
To set an attribute as the Composite Style of a design component, do the following:

1. In Design View, select the design component to which you wish to assign an attribute as Composite Style. In the screenshot below, the (contents) placeholder of the Desc element has been selected.

2. In the combo box of the Composite Style property of the Content component (see Properties sidebar at bottom right of screenshot above), the attributes of the context element are displayed. Select the attribute you wish to set as the Composite Style of the design component. (Note that there is also an empty entry in the combo box should you wish to apply no Composite Style. In this case, the RichEdit feature of Authentic View will not be enabled in the output of this design component.)

In Authentic View the user can now use the RichEdit toolbar to modify the Composite Style of this design component.

- See also
  - Text-Styling Flexibility in Authentic
  - Working with CSS Styles
  - Composite Styles
RichEdit

If an element is created in the SPS design as a RichEdit component, then the Authentic View user can mark text fragments within that element and style it using the RichEdit styling properties of Authentic View, as well as set paragraph-level formatting, such as text alignment.

RichEdit enables the Authentic View user to specify the following:

- **Character styles**: font, font-weight, font-style, font-decoration, font-size, color, background color.
- **Paragraph styles**: text alignment.

This description of the RichEdit feature given below is organized as follows:

- The RichEdit mechanism
- Creating an element in the SPS design as a RichEdit component
- Using RichEdit in Authentic View

---

**The RichEdit mechanism**

When an Authentic View user selects a text fragment in Authentic View and applies RichEdit styling to it, the RichEdit styling element and the attribute that holds the styling information is created around the selected text fragment. The RichEdit style properties that the Authentic View user selects are inserted as the value of this styling attribute.

So, if there is a text fragment in the source XML document like this:

```xml
<p>... Altova StyleVision 2012 features a unique graphical design interface ...</p>
```

and a part of this text fragment is given a RichEdit style property of bold, then the text fragment in the source XML will look like this:

```xml
<p>... <span style="font-weight: bold">Altova StyleVision 2012</span> features a unique graphical design interface ...</p>
```

The RichEdit styling element in the example above is `span` and its attribute that is to contain the RichEdit styling properties is `style`. They could be called anything. For example, instead of calling the styling element `span`, you could call it `Style`, and instead of calling the styling attribute `style`, you could call it `css`. In this case, the text fragment would then be marked up like this:

```xml
<p>... <Style css="font-weight: bold">Altova StyleVision 2012</Style> features a unique graphical design interface ...</p>
```

The important thing is that whatever name you choose for the styling element and attribute, **this styling element and attribute must be defined in the schema** and must be allowed within every element containing the text that is to be styled.

When the text is processed with an XSLT stylesheet, the style properties are passed to the output as markup appropriate to the output format.

RichEdit also enables Authentic View users to apply block-level formatting (such as text-
You can select the element and attribute that will contain the block-level formatting, similarly to how this is done for text fragments as described above. Certain block-level formatting properties, such as block-level text alignment, will then become available to the Authentic View user when the user edits an element that contains the element that has been defined for paragraph styles in the RichEdit Configuration dialog. As with character styles, the schema must be defined to allow the RichEdit paragraph-styling element inside any element in which you wish to make this paragraph styling available.

Creating an element as a RichEdit component
To create an element as a RichEdit component, do the following:

1. Drag the element from the Schema Tree and drop it at the desired location in the design.
2. From the menu that pops up, select Create RichEdit. The RichEdit Configuration dialog (screenshot below) pops up. (If a RichEdit character style has already been created for the document, you must right-click a RichEdit component in the design and select Configure RichEdit Elements/Attributes to pop up the RichEdit Configuration dialog.)
3. In the RichEdit Configuration dialog, and in the Character Styles pane, enter the name of the styling element and its attribute that is to contain the RichEdit styling properties for text fragments. You can also select the required element and attribute from the schema tree. Click the respective Select buttons to open the schema tree.

4. To enable block-level formatting (text alignment), do the following. In the Paragraph Styles pane, select the element and attribute that will contain the block-level formatting. In the Create Paragraph Type combo box, you can select the predefined format of the paragraph; this predefined format will be passed to the output.

5. When done, click OK. The element is created as a RichEdit component (see screenshot below), and an uneditable RichEdit global template for character styles that has the name of the styling element (Style in the screenshot below) is created in the design. If paragraph styles have also been specified, then an uneditable RichEdit global template for paragraph styles is also created.

Note the following:

- The uneditable RichEdit global template is created when the first RichEdit component is created in the SPS. No additional RichEdit global templates are created subsequently.
- When elements are created as RichEdit components subsequent to the creation of the first RichEdit component, the RichEdit Configuration dialog does not appear. All RichEdit components are indicated by having within them a RichEdit text box (see the overview element in the screenshot above). This RichEdit text box appears instead of the usual contents placeholder.
- The RichEdit global template can be reconfigured in that the styling element and attribute for character styles and paragraph styles can be changed. Do this by right-clicking a RichEdit component, selecting Configure RichEdit Element/Attribute, and entering the desired element and attribute names. The new element names (for character styles and paragraph styles) will appear in the title bar of the RichEdit global templates.

All schema elements that have been created as RichEdit components can now be styled with RichEdit properties in Authentic View.

---

**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.

To apply character styles, select the text to be styled. To apply paragraph styles, place the cursor within the paragraph to be styled. Then specify the required styling via the buttons and controls of the RichEdit toolbar. If the selected text is not already enclosed within the tags of the styling element, it will be enclosed now.

- **See also**
  - Text-Styling Flexibility in Authentic
  - Working with CSS Styles
  - CSS Support
Text State Icons

A Text State Icon is an Authentic View toolbar button that is associated with an element name. In Authentic View, when a Text State Icon is clicked, the selected text fragment is enclosed by the element associated with the Text State Icon. The text formatting defined for this element (in the element's global template) will therefore be applied to the selected text fragment. In this way, the Authentic View user can apply text styles using Text State Icons.

As the SPS designer, you will create the Text State Icons and define the styles of the elements associated with the Text State Icons.

Creating a Text State Icon

To create a Text State Icon for the Authentic toolbar, first select the command Authentic | Custom Toolbar Buttons, then click the Add button at the top right of the Authentic Custom Buttons dialog (see screenshot below) and select Add Text State Icon.

Text State Icons take the following parameters:

- **Element Name**: This is the element that will enclose the selected text fragment when the Text State Icon is clicked in Authentic View.
- **Bitmap**: The location of an image for the Text State Icon. The file path is relative to the SPS.
- **Text**: If no bitmap is available, the text entered in this field will be used as the button text of the Text State Icon.
• **Tooltip:** This is an optional guide for the Authentic View user on mouseover.

In the example above, we have used familiar bitmaps as the Text State Icons to mark up text with the **bold** and **italic** elements. If you wish, you could use any bitmap or button text that you think might be easier for the Authentic View user to relate to. For example, important text fragments could be associated with the **bold** element, while comments could be associated with the **italic** element. The Text State Icons could then be created as shown in the screenshot below.

In the above screenshot, note that the **Comment** Text State Icon is associated with the **italic** element. So, when text is selected in Authentic View and then the **Comment** Text State Icon is clicked, that text fragment will be enclosed by the **italic** element and will be processed in the way defined for the **italic** element (described in the next section).

---

**Defining a style rule for the element associated with a Text State Icon**

The style properties of an element associated with a Text State Icon can be defined in a global template (see screenshot below).
This screenshot shows the global templates of two elements (bold and italic), each of which has been associated with a Text State icon. The styling has been done by selecting the contents component and defining the desired font style in the Styles sidebar.

Consequently, whenever the Authentic View user creates these elements around text fragments, the respective global template (with its text styles) is applied to that text fragment.

- **See also**
  - Text-Styling Flexibility in Authentic
  - Working with CSS Styles
  - CSS Support
11.6 Designing Print Output

Properties for paged media output (PDF, RTF, and Word 2007+ in the Enterprise Edition; and RTF in the Professional Edition) can be defined in the Page Layout group of properties in the Properties sidebar. The following can be designed for print media:

- The document can be divided into sections, each of which can have separate page definitions. The properties that can be defined are listed below.
- Page dimensions (height and width) and a page orientation (portrait or landscape) can be defined.
- The margins for the body of the page and the available vertical space for headers and footers can be defined. Also, multiple pages can be defined to be facing (that is, with mirror margins) or to have the same left and right margins repeating for each page.
- Headers and footers can be defined for each section.
- Numbering styles and numbering starts can be defined for each section separately, or page numbering can run on from one section to the next.
- For each section, the number and width of columns on a page can be specified.

**Note:** When a large document is processed by FOP (Apache's FO processor), data in memory is discarded each time a new document section is processed. If you wish to reduce memory use, therefore, it is best to structure your document into multiple document sections.

---

**Properties sidebar**

Page properties can be defined individually for each section of the document in the Page Layout group of properties in the Properties sidebar (see screenshot below). These properties for a given section are accessed via the Edit Properties link of the Initial Section and Document Section items in the design (screenshot below).

![Screenshot of Properties sidebar](image)

**Document Section**

<table>
<thead>
<tr>
<th>Edit Properties...</th>
<th>Add Header/Footer...</th>
<th>Hide Headers/Footers</th>
</tr>
</thead>
</table>

Clicking the Edit Properties link pops up the Properties window, with the Page Layout properties active within it (screenshot below).
How to set the values of these properties is discussed in the section, **Page Properties**.

**See also**
- Working with CSS Styles
Document Sections

An SPS can be designed to have multiple document sections, with each document section having its own page definition settings. For example, a report which contains tables of data and text that summarizes this data can be divided into two document sections: one document section can contain the descriptive text and have portrait orientation, while the other document section with the tables of data can have landscape orientation. For each document section, the whole range of page properties (see screenshot below) can be defined. Additionally, each document section can also have different headers and footers.

When an SPS is created, it is created with one document section, called the Initial Document Section. This document section is the first document section of the document (whether a single-sectioned or multiple-sectioned document) and cannot be deleted. Initial Document Section properties include properties and styles for the entire document; these are described in the subsection, Initial Document Section.

Note: When a large document is processed by FOP (Apache's FO processor), data in memory is discarded each time a new document section is processed. If you wish to reduce
memory use, therefore, it is best to structure your document into multiple document sections.

---

**Inserting document sections**

To add a new document section, do the following:

1. Place the cursor at the location in the document where you want the new document section to start.
2. In the context menu (right-click), select **Insert Page / Column / Document Section | New Document Section**. Alternatively, select this command from the **Insert** menu. A new document section will be inserted in the design and is indicated by a document section title bar (see screenshot below; the **Hide/Show Headers/Footers** hyperlink shown in the screenshot below appears after a Header or Footer has been added to a document section). In the output, a new document section will start on a new page.

3. The new document section will have the page layout properties that were assigned to the Initial Document Section at the time the new document section was created. These page layout properties for the document section can be edited via the **Edit Properties** hyperlink of the Document Section. If required, separate **headers and footers** can be added for the document section (via the **Add Header/Footer** hyperlink). How to define page layout properties and headers and footers are described in the respective subsections of this section. When a header or footer is added, it is shown in the design within that document section. The display of headers and footers in the design can be toggled on and off with the **Hide/Show Headers/Footers** hyperlink; this hyperlink appears after a Header or Footer has been added to a document section.

4. If you wish to create a new page for the section immediately after the Initial Document Section, go to the page layout definitions of the Initial Document Section and give the **Render a Section Break** property a value of **Yes**. This starts the first non Initial Document Section on a new page. Note the converse effect also, that is, when the property has a value of **No**. In this case, the first non Initial Document Section starts directly after the Initial Document Section, without rendering a page break. This is useful if the Initial Document Setting is blank—for example, if it contains only design processing templates that produce no output. A property value of **No** ensures that the first page of the print output document will not be blank (which would have been the case if a blank page with a break after it were to be rendered for the Initial Document Section).

**Notes**

Note the following points:

- In the PDF and RTF output generated by XSLT 1.0 SPSs, only document sections that are immediate children of the Main Template are allowed. Document sections created inside conditional templates, other templates, or at any level other than that of child of the Main Template, will create only a page break in the PDF output. This restriction does not apply to PDF and RTF output generated by XSLT 2.0 or XSLT 3.0 SPSs.
- In the output document, every document section starts on a new page.
- Page margin properties are also applied to the HTML page.
- When multiple document sections are present in a design, values of the mirror margins property and the associated margin-left and margin-right properties are taken from the
initial document section. The values of these properties in subsequent document sections are ignored.

---

**Deleting a document section**

To delete a document section, in the title bar of the document section, right-click the words *Document Section*, and in the menu that pops up select the command **Edit | Delete**. The document section will be deleted, and this will be indicated by the deletion of the title bar. By deleting the document section you will be deleting the page layout properties and headers and footers created for the document section. The content of the document section, however, will not be deleted.

- **See also**
  - Initial Document Section
  - Page Layout Properties
  - Headers and Footers
  - Document Properties and Styles
Initial Document Section

Whether the document has one document section or more, properties for the document as a whole are defined in those of the Initial Document Section (the first document section of the document, *screenshot of title bar below*). Cover pages are also created in Initial Document Sections.

To edit the properties of the document, click the *Edit Properties* hyperlink in the Initial Document Section title bar. This pops up the Properties dialog of the Initial Document Section (*screenshot below*). This dialog has three tabs, for: (i) basic page layout properties, (ii) (HTML) document properties, and (iii) document styles.
Page layout properties
Page layout properties for the initial document section apply to the first document section of the
document; in single-section documents, they apply to the entire document. When a new
document section is created, it is created with the page layout properties of the Initial Document
Section at that time. The properties of the new document section can be edited subsequently. If a
property of the Initial Document Section is changed, this change will not be passed to other
document sections that already exist. New document sections will be created with the latest
values of the Initial Document Section. The various page layout properties are described in the
section Page Layout Properties. The Watermarks button brings up the Watermarks dialog in
which you can define a watermark for the section's pages. A different watermark can be defined
for each section.

HTML document properties
Properties of the output HTML document are specified in the Document Properties tab.

Document styles
The styles that are defined in the Document Styles tab apply to the entire document. If a
document has more than one document section, design elements within each document section
inherit style properties from the Initial Document Section. To over-ride inherited styles on a given
design element, specify the required style values on the individual design elements. To do this
click the design element, and, in the Styles sidebar, specify the desired styles.

Cover pages
If a cover page is required, it should be designed at the beginning of the Initial Document Section.
To ensure that the rest of the document starts on a new page, insert a page break (Insert | Insert
Page / Column / Document Section | New Page) below the cover page template. If the page
layout properties of the cover page are to be different than those of the following pages, then the
entire Initial Document Section should be used for the cover page. The following pages should
then start with a new document section.

An example is shown below.
Click the Preview RTF, Preview PDF, or Preview Word 2007+ tab to see the result in the preview window.

- **See also**
  - Document Sections
  - Page Layout Properties
  - Headers and Footers
  - Keeps and Breaks
Page Layout Properties

Page properties are assigned individually for each document section of a document design, in the Page Layout group of properties of that document section. If a design has only one document section, then the page properties of that document section are the page properties of the entire document.

Accessing the page properties of a document section

To access the page properties of a document section, click the Edit Properties link of the Initial Document Section or Document Section item in the design (see screenshot below).

This pops up the Properties window, with the Page Layout properties active within it (screenshot below).
Alternatively, clicking the Document Section title bar makes the Page Layout group of properties of that document section active in the Properties window.

---

**Page size**

Three properties determine the size of pages of a document section: (i) page height, (ii) page width, and (iii) size. Page size can be set in one of two ways:

- You can select a predefined page size from the combo box in the Paper Format property field. In this case, values for the Page height and Page width fields are automatically filled in depending on what value has been selected in the combo box.
- You can specify your own values for the Page height and Page width properties. In this case, the Paper Format field will contain the value custom size.

Valid length dimensions (for the Page height and Page width properties) are inches (\text{in})
centimeters (cm), millimeters (mm), picas (pc), points (pt), pixels (px), and ems (em). Note that (i) a unit is mandatory; (ii) there must be no space between the number and the unit; (iii) there is no default unit. Entering an invalid unit or no unit causes the property value to be displayed in red.

---

**Page margins**

The top, bottom, left and right margins of a page can be defined with the four margin properties, *Margin body from page top*, *Margin body from page bottom*, *Margin body from page left*, and *Margin body from page right*, respectively. To specify a margin, enter the required number in the relevant margin property field followed by any of the valid length units: inches (in), centimeters (cm), millimeters (mm), picas (pc), points (pt), pixels (px), and ems (em). Note that (i) a unit is mandatory; (ii) there must not be a space between the number and the unit; (iii) there is no default unit. Entering an invalid unit or no unit causes the property value to be displayed in red.

The body area is defined by the page margins you set (see screenshot below).

---

**Header and footer margins**

The *Margin header from page top* and *Margin footer from page bottom* properties specify the distance from the top of the page to the top of the header and from the bottom of the page to the bottom of the footer, respectively (see screenshot below).
The vertical extents of headers and footers are determined by the actual content of the headers and footers. You should ensure that the vertical extent of a header plus the header margin does not exceed the Margin body from page top. Otherwise, the header will be too large to be contained in the space defined for it. Similarly, ensure that the sum of the vertical extent of the footer and footer margin does not exceed the value of the Margin body from page bottom.

If the actual header or footer is too large for the space assigned for it, then the value of the Oversized Headers/Footers property comes into play and can modify the treatment of headers and footers in RTF and Word 2007+ output. If the Oversized Headers/Footers property has been set to Overlap Body Text, then the oversized header or footer will superimpose, or be superimposed by, body text. If the option Reduce Body Height has been set, then the vertical extent of the body text is reduced so as to accommodate the oversized header or footer. In the case of PDF output, the option to reduce body height is not available and oversized headers or footers will always either superimpose, or be superimposed by, body text.

**Note:** In the PDF output, you can also specify separate Margin body from page top and Margin body from page bottom values for the first and last page of each document section. These settings are available in the Page Layout (PDF) group of properties in the Properties window of the document section design element.

---

**The Multiple Pages setting**

The Multiple Pages setting has two options:

- If you set Multiple Pages to Normal, then all pages in the output will have the same value for all left margins and the same value for all right margins (see screenshot below).
- If, on the other hand, you set Multiple Pages to Mirror Margins, then the document pages are treated as facing pages (see screenshot below). This means that for even-numbered pages (left-hand-side pages), the left margin (Margin body left property) is the outer margin while the right margin (Margin body right property) is the inner margin. For odd-numbered pages (right-hand-side pages), the left margin (Margin body left property) is the inner margin while the right margin (Margin body right property) is the outer margin.

![Margins for Non-Facing Pages](image1)
![Mirror Margins for Facing Pages](image2)

The settings made for the Margin body left and Margin body right properties are applied by StyleVision to the odd-numbered pages; these margins will be reversed for even-numbered pages; the inner-margin value of odd-numbered pages becomes the outer-margin value of the even-numbered pages.
Note: If an SPS design has multiple document sections, then the value of the mirror margins setting is taken from the initial document section. The left and right margin values are also taken from the initial document section. The values of these properties in subsequent document sections will be ignored.

Page orientation
Page orientation can be set to portrait or landscape.

Columns
Columns and their widths are specified with two properties: Column count and Column gap, which specify, respectively, the number of columns and the space between two columns. The width of a column is thus the width of the page body minus the sum of the column gaps, divided by the number of columns.

Text will fill the columns on a page one by one. Only after all the columns have been filled will a new page be started. A column break can be forced by inserting a new column at the desired point in the design. To do this, right-click at the location where the column break is required and select the context menu command Insert Page / Column / Document Section | New Column.

Page numbering
There are two relevant properties: Numbering format and Numbering starts at. These work as follows:

- The required page number format is set by selecting one of the pre-defined options from the drop-down menu for Numbering format. (The Numbering format selection also applies to the page total, if this is inserted.)
- The page numbering for a document section can be set to start with any positive integer. This integer is specified in the Numbering starts at property. If the numbering is to continue from the previous document section, then this field should be left blank or set to auto.

Note: Page numbers can be inserted in a document by inserting a page number placeholder (with the command Insert Page / Column / Document Section | Page Number). The total number of pages in the output document is inserted with the command Insert Page / Column / Document Section | Page Total.

Note: MS Word does not always update page numbers automatically. To manually update page numbers in MS Word, press Ctrl+A and F9.
Page numbering in the RTF output
In order to display page numbering in the RTF output in MS Word, you must select, in MS Word, the entire contents of the document (with Edit | Select All or Ctrl+A), and then press F9. This will cause the page numbering to be displayed—if you have inserted page numbering.

Page totals
To output the total number of pages at various locations in your document, use the page total feature.

Page starts for document sections
For each document section that is not the Initial Document Section, the Section Starts On property specifies whether the document section should start on the next page (irrespective of whether it is odd-numbered or even-numbered), or whether it should specifically start on an odd-numbered or even-numbered page. For example, if the previous document section ends on an odd-numbered page and the current document section is specified to start on an odd-numbered page, then the even-numbered page that occurs directly after the end of the previous document section will be left blank. Note that it is the underlying document page-numbering that determines whether a page is odd-numbered or even-numbered. The page numbering that the user specifies is irrelevant for determining the document section start-page.

Page break after the Initial Document Section
If you wish to start the section immediately after the Initial Document Section on a new page, give the Render a Section Break property (of the Initial Document Section's page layout properties) a value of Yes. This starts the second document section (that is the first section after the Initial Document Section) on a new page. Note that this property is available only for the Initial Document Section, not for any other section.

Note the converse effect also, that is, when the property has a value of No. In this case, the first non-Initial-Document-Section starts directly after the Initial Document Section—without rendering a page break. This is useful if the Initial Document Setting is blank—for example, if it contains a set of templates that produces no output. A property value of No (for Render a Section Break) ensures that the first page of the print output document will not be blank (which would have been the case if a page break were to be rendered after an Initial Document Section that produced no output).

Page layout properties for PDF
For PDF output, you can specify different page margins for the first page of the section. These
properties work in the same way as for other pages in the section and are as described in the section, Page Margins.

- See also
- Document Sections
- Document Properties and Styles
- Headers and Footers
- Keeps and Breaks
Headers and Footers: Part 1

Headers and footers can be added for each document section of the document, including the Initial Document Section.

Adding a header or footer for a document section
To add a header or footer for a document section, click the Add Header/Footer link of the Initial Document Section or the Document Section title bar (see screenshot below).

Document Section

Adding a header or footer for a document section
To add a header or footer for a document section, click the Add Header/Footer link of the Initial Document Section or the Document Section title bar (see screenshot below).

From the menu that pops up (screenshot below), select the required item. A header or footer can be added separately for odd or even pages, or a single header or footer can be added for all pages. Additionally, a separate header/footer template can be created for the first page (Add Header First, Add Footer First) and/or for the last page of a document section (Add Header Last, Add Footer Last, PDF output only). This is useful if the first and/or last page of a document section must have a different header/footer: for example, when the first page is a cover page (an empty header/footer template could be created in this case).

On clicking the required header or footer, a template for the header or footer is created within that document section in Design View. The Design View display of header/s and footer/s in a document section can be toggled on and off by clicking the Hide Headers/Footers link in the Initial Document Section or Document Section item (see screenshot above).

Note: Headers and footers for the last page are supported for PDF output only.

Designing the header/footer in Design View
The header/footer template is designed just like any other template. Components can be dragged from the schema tree or entered statically, and then styled. An example is shown below. When a header is added, the template will look something like this:
Change the header as required. Note that you can use both static and dynamic content, and even images.

Click the Preview RTF, Preview PDF, or Preview Word 2007+ tab to see the results in the Preview windows. The illustration below shows Page 2 of the Organization Chart document in the PDF Preview window, with the header as defined above. To display page numbering in the RTF output, you must select Edit | Select All (or Ctrl+A) in MS Word, and then press F9. Also see displaying page-numbering in RTF output.

The following points should be noted:

- The vertical extent of the header and footer should not exceed the respective margin body (top or bottom) less the extent of the margin header or margin footer, respectively (see Page Layout Properties for details). The vertical extent of the header or footer, consequently, is determined by the top/bottom body margins and margin header/footer.
- You can define a header/footer either (i) for all pages in the document section, or for (ii) for even and odd pages in the document section separately. Additionally, separate first page and last page headers/footers can be inserted. (See Headers and Footers: Part 2 for
more information.)
- Page numbering in a document section starts with the number you specify in the **Page Layout Properties**.

---

**Deleting a header or footer**
To delete a header or footer, right-click the header/footer title bar and, from the menu that pops up, click **Edit | Remove**.

- **See also**
  - Document Sections
  - Page Layout Properties
  - Keeps and Breaks
Headers and Footers: Part 2

In this section, we describe how to create the following types of headers and footers:

- Different headers/footers for odd-numbered and even-numbered pages
- Different headers/footers for different document sections
- Simulating headers/footers inside a page
- Headers/footers with subtotals

Note: In the PDF output, you can also specify separate Margin body from page top and Margin body from page bottom values for the first and last page of each document section. These settings are available in the Page Layout (PDF) group of properties in the Properties window of the document section design element.

Different headers/footers for odd-numbered and even-numbered pages

For each document section, odd-numbered and even-numbered pages can be assigned different headers/footers.

To create different headers for odd-numbered and even-numbered pages, click the Add Header/Footer link in the title bar of the respective document section, and select Add Header Odd and Add Header Even from the menu that pops up (screenshot below). This creates two header templates, one for odd-numbered pages, the other for even-numbered pages. Enter the content of the two headers in the templates.

Separate footers for odd-numbered and even-numbered pages can be created in a similar way to that described above for headers.

Different headers/footers for different document sections

Different headers/footers can be created for each document section of the document. To do this, click the Add Header/Footer link of the Initial Document Section or the Document Section title.
bar. This pops up the Add Header/Footer menu shown in the screenshot above. Note the following points:

- Headers/footers for odd-numbered and even-numbered pages can be added separately, or a common header/footer can be added for all pages in the document section.
- An additional first-page and/or last page header/footer can be added. These headers/footers will be used on the first and/or last page of the document section instead of other headers/footers that might be defined for that document section.
- Page numbering for the document section can either run on from the previous document section or start at a designated number.
- The Page Total is the page count of the entire document not that of the current document section.

Simulating headers/footers inside a page
Headers and footers can be designed manually inside a layout container. The approach would be to design a single page as a layout container. The header and footer are created within static tables located, respectively, at the top and bottom of the page. If more than one page is to be designed, then multiple layout containers can be used, each separated from the next by a page break (Insert | Page / Column / Document Section | New Page).

Headers/footers with subtotals and running totals
When a document contains a list of numerical items that must be totalled and the list extends over multiple pages, subtotals of each page and/or running totals might be required to appear in the headers and/or footers of each page. The Subtotals.sps example, which is in the (My) Documents folder, demonstrates how running totals can be created and included in headers and footers.

The following strategy was used to design this SPS:

- Because the listing is in a table and because a table cannot be made to auto-fit a printed page, the number of rows that must be accommodated on a page must be specified. These numbers are given in two variables that have been defined on the top level template, that for the $XML template; they are named RowsOnFirstPage and RowsPerPage.
- The page count is derived by dividing the total number of list items by the number of rows per page (adjusted to take account of the different number of rows on the first page). The page count is stored in a variable called CountOfPages (defined on the $XML template).
- A user-defined template is created for the sequence 1 to $CountOfPages, and a static table is created within this template. Defined on this template are two variables that calculate the which row is to be the first row ($RowFrom) and the last row ($RowTill) on each page. The rows in the table are generated by a user-defined template, which selects the items in the XML file (file elements) on the basis of their position with respect to the $RowFrom and $RowTill values. If the position of the file element is an integer value that lies in the range delimited by values of the $RowFrom and $RowTill variables of the current page, then a row will be generated for the current file element.
- The running totals are generated with Auto-Calculations and inserted into rows at the top
and bottom of the tables. Note that the XPath expressions to generate running totals at the top and bottom of pages are different from each other.

- Headers and footers are created in tables, respectively, above and below the main table on the page. The Auto-Calculations to generate the running totals are inserted in the header and footer templates.
- A page break is inserted at the end of each page.

See also
- Document Sections
- Page Layout Properties
- Keeps and Breaks
**Keeps and Breaks**

In PDF documents (*Enterprise edition only*), keeps and breaks for pages and columns can be set in the XSL-FO group of styles (Styles sidebar, *screenshot below*). This group of styles enables you to specify whether the current design document block (the block within which the cursor is currently placed) should have a page/column break placed before or after it, or whether it should be kept with adjacent blocks. Whether table headers and footers are omitted (or repeated) at page breaks can also be set using the `table-omit` properties. For more information about these properties, see the XSL-FO specification.

For the printed version of HTML pages, settings for page breaks and widows/orphans (leading/trailing lines on a page) can be made via the relevant properties in the *Paged Media* group of styles (*see screenshot above*).

**See also**
- [Page Layout Properties](#), for information on defining page properties.
- [Headers and Footers](#), for information creating headers and footers.
- [Styles sidebar](#)
Footnotes

You can insert footnotes in a document by adding the Footnote component \(\text{(Insert | Insert Footnote)}\) at the location where you want the footnote number to be. Footnotes are available in paged media output (PDF, RTF, and Word 2007+ in the \(\text{Enterprise Edition}\); and RTF in the \(\text{Professional Edition}\)).

Note the following points:

- The text of the footnote must be placed within the tags of the footnote component, and the footnote text can be formatted.
- In the output, the footnote number appears at the location where the footnote was added. The footnote text appears at the bottom of the page, together with the corresponding footnote number.
- In the output, footnote text will be formatted according to the formatting of the text within the footnote component in the design.
- In the output, footnotes are numbered automatically through to the end of the document.
- In the case of multiple output documents, numbering is re-started for each output document.

In the screenshot below, two footnote components \(\text{(Insert | Insert Footnote)}\) have been inserted. The footnote text has been placed within the tags of the component, and the text has been formatted.

The screenshots below show the output. The screenshot at left shows the complete page, while the screenshots at right show closeups of the footnote numbers \(\text{(top)}\) and footnote texts \(\text{(bottom)}\).
**Summary of News Items**

**Note:** Formatting of footnote numbers is not supported.
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 PDF 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 earlier versions of 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.

Using fonts in StyleVision
- The default document font used in the PDF output is Helvetica (Acrobat PDF Reader versions prior to 4.0) or Arial MT (Acrobat PDF Reader version 4.0 or later). So, if no font is specified for any text in the design document, Helvetica/ArialMT (a Base 14 font) will be used.
- If you specify Helvetica or Times in the SPS, and if the Acrobat PDF Reader used to view the PDF output is version 4 or higher, then the PDF Reader will use Arial MT and Times New Roman PS MT, respectively, from its own font resources folder. Acrobat PDF Reader does this even if Helvetica and Times are active on the system.
- In StyleVision, you can change the default font for the entire document by first selecting all the text in the document. Then, in the Styles sidebar, select the texts item in the Styles For column, and with the Font group selected, assign the desired font to the font-family attribute. This font is assigned to each text node in the document. An alternative font can then be set for individual text nodes as required.
- If in the XSL Options tab of the Options dialog (Tools | Options) you select FOP 0.93/1.0 compliant, then only the font-family values Helvetica, Times, Courier, and Symbol (exactly as spelled here) will be passed through to the XSLT-for-FO. Entering any other font-family value on a node will remove the font-family attribute from that node in the resulting FO document—leading to the default Helvetica/ArialMT being output for that node.
- If in the XSL Options tab of the Options dialog (Tools | Options) you select Full usage of
XSL-FO standard, then any font-family you enter in the Text Style window will be passed to the XSLT-for-FO document. If the selected font is not available to the formatter, an error may result. In order to avoid such errors, add alternative fonts and a generic fallback. For example:

```xml
<font-family="Bodoni, Garamond, serif"
```

In this example, Garamond is a second choice, and the generic serif is the fallback font. If neither Bodoni nor Garamond is available, then the generic serif font is used. The three generic fonts are: serif (Times or Times New Roman PS MT, depending on the Acrobat Reader version), sans-serif (Helvetica or ArialMT), and monospace (Courier).

**Note for HTML use:** CSS2 allows alternative choices, as well as a generic font as a fallback (serif, sans-serif, monospace, cursive, and fantasy).

---

**Making fonts available to the formatter**

Most formatters already have available to them the Base 14 fonts. It is important to know the names by which the formatter recognizes these Base 14 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.

---

**Font support in FOP**

There are two steps involved in setting up custom fonts:

1. Informing FOP how to find the font files.
2. Informing FOP how to use the fonts.

**How to find the required font files**

More recent versions of FOP can automatically detect system fonts or can scan user-specified directories for fonts and register these with FOP. See the [Fonts page at the FOP website](http://fop.xml.org/docs/Fonts.html).

If you are using an earlier version of FOP, you will need to first generate a font metrics file (an XML file) for each required font from the font's 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](http://fop.xml.org/docs/Fonts.html). Next, in the FOP configuration file, the required metrics file for each font must be specified in the `metrics-url` attribute of the respective `font` element.

```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>
```

**Telling FOP how to use the fonts**

The FOP configuration file—an XML file called `fop.xconf` and located in the `conf` folder in the
FOP installation folder—must be edited so that FOP knows how to use the fonts 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:

```
<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,

| metrics-url | URL of the metrics file. If not absolute, relative to the path in the `font-base` element of the configuration file. Optional. |
| embed-url   | URL of the font file (usually located in `\%WINDIR\Fonts`). Required if the Viewer does not have the file on its local system. |
| name        | Maps this font name to the font defined with `metrics-url` and/or `embed-url`. |
| style       | Maps this font style to the font defined with `metrics-url` and/or `embed-url`. |
| weight      | Maps this font weight to the font defined with `metrics-url` and/or `embed-url`. |

For more details, see `Registering Fonts with FOP` at the FOP website.

---

**Editing the batch file (fop.bat) to read the configuration file**

After the configuration file has been edited as described above, the FOP executable (`fop.bat`), which is in the FOP installation folder, must be edited so that FOP reads the configuration file. Do this by changing the last line of `fop.bat` from:

```
"%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
```

See `Starting FOP` at the FOP website.

---

**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.

**Note:** When entering font sizes, do not use an intervening space between the point size and its unit. For example, 12pt is correct, 12 pt is incorrect.

---

- **See also**
  - Setting CSS Property Values
Pixel Resolution

If you use pixels as a unit of length in your SPS, you should be aware that pixel-defined lengths are a function of screen resolution. The corresponding absolute lengths in print could be very different from what you see on screen. In this section, we do the following:

1. Discuss why pixel-defined sizes have two forms: (i) an abstract form, defined in pixels; (ii) an actual size, obtained by resolving the abstract form in terms of a specific screen resolution.
2. Explain StyleVision functionality to deal with this issue.

From pixels to points

A few key points are essential to understanding the factors that affect the abstract and actual dimensions of pixel-defined lengths:

- The pixel is a relative unit: its size depends on screen resolution. The higher the resolution, the smaller the pixel. Screen resolution is given with dpi (dots per inch). A dot in the case of screens is a pixel. So, if screen resolution is 72 dpi, then there are 72 pixels (dots) in one inch of a line of pixels. If screen resolution is 96 dpi, then there are 96 pixels in an inch. How many pixels there are in an inch of screen length depends on the screen resolution. The most commonly available screen resolutions today are 72 dpi, 96 dpi, and 120 dpi. The higher the dpi, the smaller will be the pixels.
- The length unit known as the point is an absolute unit of length, used most commonly in the printing industry: 72 points make up an inch.
- From the above it can be seen that only when screen resolution is 72 dpi will one pixel be equal to one point. For other screen resolutions, the absolute length can be calculated. The table below lists the absolute length (in points) of 100px at various screen resolutions.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Pixels</th>
<th>Points</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>72 dpi</td>
<td>100</td>
<td>100</td>
<td>1.00</td>
</tr>
<tr>
<td>96 dpi</td>
<td>100</td>
<td>75</td>
<td>0.75</td>
</tr>
<tr>
<td>120 dpi</td>
<td>100</td>
<td>60</td>
<td>0.60</td>
</tr>
</tbody>
</table>

To convert pixels to points for each screen resolution, we can use a factor given by the ratio of points in one inch (72) to pixels in one inch (dpi), that is, 72 divided by dpi. The conversion factors for various resolutions are given in the table above. For example: 72 div 96 = 0.75 and 72 div 120 = 0.60. Multiplying the length in pixels by the appropriate conversion factor gives the absolute length in points. So 100px on a 96 dpi screen is 75pt.

Since 72 points make an inch, you can obtain the length in inches by dividing the length in points by 72. For example, 100 points is equal to 100 div 72 inches = 1.389in.

Screen resolution and absolute length
In the previous section we have seen that only when the screen resolution is 72 dpi will the absolute length in points be the same as the number of pixels used to define that length. Screen resolutions on Windows systems, however, are typically not 72 dpi but 96 dpi. This means that the number of points of a pixel-defined length on such a screen will be 75% the number of pixels. For example, in the StyleVision Design View screenshot below, the two characters measure 100px and 75pt, respectively.

The reason they are the same height is that the screen resolution in which they appear is 96 dpi. At this resolution 100px is equal to 75pt in absolute terms.

The point to note is that when specifying pixels as a unit of length, be aware of your monitor's screen resolution (normally 96 dpi on Windows systems); it determines the absolute length of the pixel-defined length that you see on screen.

---

**Print output resolution**

When an SPS is transformed into a print format, such as PDF, RTF, or Word 2007+, non-absolute pixel lengths must be converted to absolute lengths, such as points or inches (because lengths on paper cannot be defined in terms of pixels). The question is: What factor (or screen resolution) should be used to convert from pixels to points?

In StyleVision, you can select the output dpi for each SPS individually. So, for example, if you select 96 dpi, then a 100px character will be rendered in print output as a 75pt character (see the table above). If you select 72 dpi, then the same character will be rendered in print output as a 100pt character. This system applies to all lengths defined in pixels.

**Note:** Conversion to an absolute measure is not carried out for HTML output. For HTML, the original pixel units are passed to the HTML file unchanged.

---

**Setting print output resolution of an SPS**

To set the print output resolution of an SPS, click File | Properties. In the Properties dialog that pops up, click the Resolution tab (screenshot below).
In the Resolution tab, select the required print resolution. The conversion factor for each listed dpi option is given in the conversion table above (in the section, From Pixels to Points). Multiplying the pixel count by the conversion factor gives the absolute length in points. Note that the conversion to absolute units is applied only to print output formats. The HTML output format will retain the original pixel definitions. Note also that this setting does not change the screen resolution of your monitor.

- **See also**
  - Setting CSS Property Values
PDF Bookmarks

PDF Bookmarks are navigational links that are displayed in a pane in PDF reader applications (screenshot below). They usually reflect the structure of the PDF document and allow the user to navigate to different parts of the document.

![Bookmarks Screenshot](image)

StyleVision gives you the flexibility to set up the structure of a document's PDF Bookmarks exactly as you want. By default, all the Tables of Contents (TOCs) defined in the document will be displayed as PDF Bookmarks. Depending on your FOP processor, items that are common to the various TOCs in your document may or may not be merged. If you wish to set up PDF Bookmarks that are different from the TOC structure, you can use the PDF Bookmarks feature to customize PDF Bookmarks. If you create PDF Bookmarks, then these override input from the TOC settings and are the settings adopted for the document's PDF Bookmarks.

TOCs and PDF Bookmarks

What bookmarks are created in the PDF output depends on the TOCs in the document and whether a PDF Bookmark Tree has been created. The following options are available:

- **One TOC in the design**: Bookmarks in the PDF will be automatically generated using the TOC structure.
- **Multiple TOCs in the design**: Bookmarks in the PDF will be automatically generated for each TOC, with each TOC being appended to the previous TOC. Note that some FOP processors, such as FOP 0.93, merge items that are common to TOCs.
- **A PDF Bookmark Tree is created in the design**: If a PDF Bookmark Tree is created in the design, this tree is used for the bookmarks in the PDF; any TOC/s in the design are overridden. If a PDF Bookmark Tree is created in the design but is empty, then no bookmarks will be generated in the PDF—even if TOCs are present; the PDF Bookmark pane will be empty.
Creating the PDF Bookmark Tree
This section describes how bookmarks can be created in the PDF output using the PDF Bookmark Tree. (If no PDF Bookmark Tree is present in the document, then bookmarks are generated automatically from the TOCs in the design. For information about creating TOCs, see the section Table of Contents, Referencing, Bookmarks.)

A PDF Bookmark Tree (template) is a template that is defined on a document's Initial Document Section. If the design has multiple document outputs, then PDF Bookmark Trees can be specified for each of the multiple documents, on their respective Initial Document Sections.

Add the PDF Bookmark Tree template by clicking the Add Header / Footer link in the title bar of the Initial Document Section, and then selecting the Add PDF Bookmark Tree command (see screenshot above). The PDF Bookmark Tree template will be added (see screenshot below).

You can now drag nodes from the Schema Tree sidebar into the PDF Bookmark Tree to create a customized PDF Bookmarks structure.

For every node that you drop into the PDF Bookmark Tree, you will be prompted as to whether you wish to create a PDF Bookmark or a template. If you select PDF Bookmark, you will be
prompted to enter a target for the bookmark. These targets must be bookmarks (anchors) that already exist in the design. How to create such bookmarks (anchors) is described in the section, Inserting Bookmarks. After you have entered the target bookmark, a PDF Bookmark component is added to the tree (see screenshot below).

Each PDF Bookmark component contains a hyperlink component (see screenshot above) that has two key attributes:

- **Content** which defines the text of the PDF Bookmark. You can delete the default content placeholder and insert any text or suitable component instead.
- **A URL property**, which specifies the target of the bookmark. If the target bookmark is within the same document, then the URL must be a fragment identifier (and must therefore start with #). For more information, see Defining Hyperlinks.

---

**PDF Bookmark targets**

The URL property of a PDF Bookmark’s hyperlink component is the target bookmark you assigned when the PDF Bookmark was created. This target can be edited by right-clicking the hyperlink component and selecting **Edit URL** from the context menu that pops up.

We recommend that the targets are unique (that is, no PDF Bookmark should point to the same design bookmark). Otherwise the output might be dependent on the FOP version. (For example, FOP 0.95 ignores nested PDF Bookmarks if they have the same destination. Also FOP 0.95 ignores PDF Bookmarks which point to undefined document locations. FOP 1.0, however, does display such bookmarks in the generated Bookmark Tree.) PDF Bookmarks which point to external locations (e.g. http or file urls) are currently not supported by Apache FOP (both 0.95 and 1.0).

---

**PDF Bookmark Tree structure**

Ideally PDF Bookmarks in the PDF are structured to display the hierarchy of the document. This structure can be specified in the PDF Bookmark Tree by nesting PDF Bookmarks. For example, in the screenshot below, the **n1:Office** element (highlighted in blue) has been dropped into the PDF Bookmark Tree after the hyperlink component of the Organization Chart bookmark but inside
the bookmark itself. The Office PDF Bookmark is therefore nested inside the Organization Chart PDF Bookmark.

Note the following points about the example shown in the screenshot above:

- The XML document contains one `n1:orgChart` element and two `n1:Office` elements.
- The `n1:orgChart` element has a static bookmark. A fragment identifier in the PDF Bookmark’s hyperlink component targets this static bookmark.
- The name of the Organization Chart PDF Bookmark is a static text string.
- The bookmarks for the two `n1:Office` elements are dynamically created using XPath expressions that select the names of the two offices from a child node, `n1:Name`. The URL of the Office PDF Bookmark component is a dynamically generated fragment identifier, also obtained with the help of the child node, `n1:Name`.
- The name of each Office PDF Bookmark is generated with an Auto-Calculation that uses the respective a child node, `n1:Name`, of each `n1:Office` element.

The PDF Bookmarks generated by this PDF Bookmark Tree are as in the screenshot below.

Note: If either the target bookmarks or the PDF Bookmark URLs are incorrect, errors or incorrect PDF Bookmarks are possible, depending upon the FO processor being used (see PDF Bookmark Targets above).
Deleting the PDF Bookmark Tree
Delete the PDF Bookmark Tree either by selecting the tree and clicking the Delete key, or by right-clicking the tree and selecting the command Edit | Delete.

See also
- Initial Document Section
- Page Layout Properties
- Headers and Footers
Watermarks

A watermark is an image or text that is displayed on the background of each page of a document section.

The following options are available:

- Watermarks must be defined separately for each document section. Each section can therefore have a different watermark.
- For each document section, you must define (with XPath) some condition to be fulfilled in order for that section's watermarks to be enabled. For example, the condition could be, say, that the drafts attribute of some element must have the value true. If the condition is not fulfilled, or if the condition evaluates to false(), that watermark is disabled.
- A watermark can be a text that you customize in StyleVision or it can be an image you select.
- In the design, you can specify the location and appearance of watermarks on the page.

Note: Watermarks are available only for print-output media, not for HTML or Authentic View.

Creating a watermark

A watermark can be created separately for each document section. To create (or to edit) a watermark, click the Edit Properties link of the Initial Section or Document Section items in the design (screenshot below).

In the Properties dialog that pops up (screenshot below), click the Watermark button at the bottom of the dialog.
This pops up the Edit Watermark dialog (*screenshots below*), in which you specify the properties of the watermark/s of that document section. According to whether you wish to use an image or text as the watermark, select either the *Image* or *Text* tab. (If you wish to define both an image as well as a text as the watermarks of that section, enter the details of each type in their respective tabs. In this case, both types of watermark will be created in the output.)

![Edit Watermark dialog](image)

**Parameters for image watermarks**
The *Image* tab (*see screenshot above*) contains the parameters for defining an image watermark for that section. An XPath expression specifies the condition for enabling the image watermark. If you do not wish to specify any condition, then enter `true()` in the *Condition* text box. This causes the condition to evaluate to `true()`. If the condition evaluates to `false()`, or if no condition is specified, then the image watermark is not enabled. Next, browse for the image, so that the filepath to the image is entered in the *Image* text box.

You can then set the transparency of the image and its size and position on the page. The *Fill the entire page* option expands the image till one dimension (height or width) is filled. Click **OK** when done.
Parameters for text watermarks

The Text tab (see screenshot below) contains the parameters for defining a text watermark for that section. An XPath expression specifies the condition for enabling the text watermark. If you do not wish to specify any condition, then enter `true()` in the Condition text box. This causes the condition to evaluate to `true()`. If the condition evaluates to `false()`, or if no condition is specified, then the text watermark is not enabled.

Next, enter the text you wish to use as the watermark, then specify its formatting (font family, font size, transparency, and color) and orientation. Note that the text will be stretched to extend across the page in all orientations. Click OK when done.

Removing or disabling a watermark

To remove or disable a watermark, in the Edit Watermark dialog (see screenshots above), either delete the condition or set the XPath expression of the condition to `false()`. Note that the Image tab and Text tab each have a separate condition, for their respective watermarks.

See also
- Initial Document Section
• Page Layout Properties
Chapter 12

SPS File: Additional Functionality
12 SPS File: Additional Functionality

Additional to the content editing, structure, advanced, and presentation procedures described in this documentation, StyleVision provides a range of miscellaneous additional features. These are listed below and described in detail in the sub-sections of this section.

- **Global Resources.** Global resources provide flexibility in selecting resources. For example, multiple resources (such as files and databases), can be assigned to an alias. When an alias is used as a source (XML, XSD, etc) of an SPS, the resource can be switched among the multiple resources assigned to the alias.

- **Authentic Node Properties.** Individual nodes in the XML document have Authentic View-specific properties. Nodes can be defined to be non-editable, to be displayed with markup tags, to display user information on mouseover, etc.

- **Replace Parent Node OnClick With.** The value of the parent node of a button or hyperlink can be selected by the Authentic View user. The SPS can be designed to modify presentation based on what the Authentic View user selects.

- **Additional Validation.** A node can be tested using an XPath expression to return a Boolean value that determines whether user input for that node is valid. This test is in addition to document validation against a schema.

- **Working with Dates.** In Authentic View, a graphical date-picker ensures that dates are entered in the correct XML Schema format. Furthermore, dates can be manipulated and formatted as required.

- **Unparsed Entity URIs.** URIs can be stored in unparsed entities in the DTD on which an XML document is based. The Unparsed Entity URI feature enables images and hyperlinks to use these URIs as target URIs.

- **Using Scripts.** StyleVision contains a JavaScript Editor in which JavaScript functions can be defined. These functions are then available for use as event handlers anywhere within the SPS, and will take effect in the output HTML document.

- **HTML Import.** An HTML file can be imported into StyleVision and an XML, XSD, and SPS files can be created from it.

- **New from XSLT.** An SPS can be created from an XSLT-for-HTML or an XSLT-for-FO. Template structure and styling in the XSLT will be created in the SPS. You can then modify the SPS components and add content and formatting to the SPS.

---

**See also**

- Properties sidebar
- Authentic View
12.1 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 XML file can be generated by an Altova MapForce mapping and used in StyleVision as an XML Working File in an SPS.

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 StyleVision, 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 StyleVision.

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.

See also:

- **Tools | Global Resources**, for the menu command to access the Altova Manage Global Resources dialog.
- **Tools | Active Configuration**, for the menu command to change the active configuration of the application.
- **Toolbars and Status Bar**, for information about the Global Resources toolbar.
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.
(see screenshot above).

**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.

**See also:**

- [Tools | Global Resources](#), for the menu command to access the Altova Manage Global Resources dialog.
- [Tools | Active Configuration](#), for the menu command to change the active configuration of the application.
- [Toolbars and Status Bar](#), for information about the Global Resources toolbar.
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.
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).

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 analysed 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.

---

**See also:**
- [Tools | Global Resources](https://www.altova.com), for the menu command to access the Altova Manage Global Resources dialog.
- [Tools | Active Configuration](https://www.altova.com), for the menu command to change the active configuration of the application.
- [Toolbars and Status Bar](https://www.altova.com), for information about the Global Resources toolbar.
Folders

In the Global Resource dialog for Folders (screenshot below), add a folder resource as described below.

![Global Resource dialog](image)

**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.
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.

---

**See also:**

- [Tools | Global Resources](#), for the menu command to access the Altova Manage Global Resources dialog.
- [Tools | Active Configuration](#), for the menu command to change the active configuration of the application.
- [Toolbars and Status Bar](#), for information about the Global Resources toolbar.
In the Global Resource dialog for Databases (screenshot below), you can add a database resource as follows:

**Databases**

Define the alias (its name and configurations) 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 StyleVision, 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.

   ![Choose Root Object](image)

   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.

---

**See also:**
- [Tools | Global Resources](#), for the menu command to access the Altova Manage Global Resources dialog.
- [Tools | Active Configuration](#), for the menu command to change the active configuration of the application.
Toolbars and Status Bar, for information about the Global Resources toolbar.
Using Global Resources

There are several types of global resources (file-type, folder-type, and database-type). Particular scenarios in StyleVision allow the use of particular types of global resources. For example, you can use file-type or folder-type global resources for a Working XML File or a CSS file. Or you can use a database-type resource to create a new DB-based SPS. Some scenarios in which you can use global resources in StyleVision 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.

See also:

- Defining Global Resources, for information about defining Global Resources.
- Tools | Global Resources, for the menu command to access the Altova Manage Global Resources dialog.
- Tools | Active Configuration, for the menu command to change the active configuration of the application.
- Toolbars and Status Bar, for information about the Global Resources toolbar.
- Projects in StyleVision, for information about using global resources in projects.
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).

![Open Global Resource Dialog](image)

- **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:

- Adding and modifying schema sources and Working XML Files and Template XML Files
- Saving as Global Resource
- Adding modules and CSS files
- Adding global resources to a project
**Schema, Working XML File, Template XML File**

In the Design Overview sidebar (screenshot below), the context menus for the Schema, Working XML File, Template XML File contains an entry that pops up the Open dialog in which you can assign the schema or Working XML File via a global resource. Clicking the **Switch to Global Resources** button pops up a dialog with a list of all file-type global resources that are defined in the Global Resources XML File currently active in StyleVision. (How to set the currently active Global Resources XML File is described in the section **Defining Global Resources**.)

If a global resource has been selected as the file source, it is displayed in the relevant entry in the Design Overview sidebar (screenshot below).

**Adding modules and CSS files from a global resource**

In the **Design Overview sidebar**, the **Add New Module** and **Add New CSS File** commands pop up the Open dialog, in which you can click Switch to Global Resources to select a Global Resource to be used. Modules and CSS files can then be changed by 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.
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 StyleVision. This is because StyleVision will open the file assuming the filetype specified in the definition of the configuration.

---

**Global Resources in projects**

Global resources can also be added to the currently active project via the **Project | Add Global Resource to Project** command. This pops up a dialog listing the file-type global resources in the currently active **Global Resources XML File**. Select a global resource and click **OK** to add it to the project. The global resource appears in the Project sidebar and can be used like any other file.

---

**See also:**

- [Defining Global Resources](#), for information about defining Global Resources.
- [Tools | Global Resources](#), for the menu command to access the Altova Manage Global Resources dialog.
- [Tools | Active Configuration](#), for the menu command to change the active configuration of the application.
Assigning Databases

When an SPS is created from a database (DB) with the File | New from DB 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 DB Data Selection.

See also:

Defining Global Resources, for information about defining Global Resources.
Tools | Global Resources, for the menu command to access the Altova Manage Global Resources dialog.
Tools | Active Configuration, for the menu command to change the active configuration of the application.
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 Configuration Combo Box](screenshot)

In this way, by changing the active configuration, you can change source files that are assigned via a global resource.

See also:

- [Defining Global Resources](DefiningGlobalResources), for information about defining Global Resources.
- **Tools | Global Resources**, for the menu command to access the Altova Manage Global Resources dialog.
- **Tools | Active Configuration**, for the menu command to change the active configuration of the application.
12.2 Authentic Node Properties

Authentic node properties are properties you set for the display of a node in Authentic View. For example, a node can be displayed with XML markup tags, be defined to be non-editable, and/or to have user information displayed when the cursor is placed over the node. Authentic node properties can be set on various SPS components. What properties are available for a component depends upon the type of component it is.

To assign Authentic node properties, select the required component in the design. Then, in the Authentic group of properties in the Properties sidebar (screenshot below), specify the required Authentic node settings. Alternatively, you can right-click the node-template and select Edit Authentic Properties.

---

Defining Authentic Properties

The following node settings can be made to control the behavior of individual nodes in the Authentic View display.

**Add children**

This setting is available when the selected node is an element. It allows you to define what child elements of the selected element are inserted when the selected element is added. The options are: all child elements, mandatory child elements, and no child element.

**Additional validation**

In addition to validation of the XML document against a schema, additional validation can be specified for individual nodes and Auto-Calculations. You can set an XPath expression to define the validity range of the XML content of the node or Auto-Calculation. If the XML value of the node is invalid, this is made known to the Authentic View user by means of an error message when the XML document is validated (F8). See Additional Validation for details.

**DB rows displayed and fetched**

The DB Rows Displayed and DB Rows Fetched properties define, respectively, how many DB
rows will be displayed in Authentic View and how many DB records are originally loaded. The \textit{DB Records Fetched} property thus enables you to speed up the loading and display time.

\textbf{Content is editable}
Defines whether the node is editable or not. By default the node is editable. This setting is available when the selected node is an element, attribute, or contents. Auto-Calculation results cannot be edited because the value is computed with the XPath expression you enter for the Auto-Calculation; this option is therefore not available for Auto-Calculations.

\textbf{Mixed markup}
This setting is available when the selected node is an element or attribute, and enables you to specify how individual nodes will be marked up in the mixed markup mode of Authentic View. The following options exist: large markup (tags with node names); small markup (tags without node names); and no markup.

\textbf{Show "add Name" when XML Element is missing}
Determines whether a prompt ("Add [element/attribute name]") will appear in Authentic View when the selected element or attribute is missing. By default, the prompt will be displayed. This setting is available when the selected node is an element or an attribute.

\textbf{User info}
Text entered in this text box appears as a tooltip when the mouse pointer is placed over the node. It is available when the selected node is an element, attribute, contents, or an Auto-Calculation. If both the element/attribute node as well as the contents node has User Info, then the User Info for the contents node is displayed as the tooltip when the mouse is placed over the node.

- \textbf{See also}
  - Additional Validation
  - Properties sidebar
  - Editing in Authentic View
12.3 **Replace Parent Node OnClick With**

With the *Replace Parent Node OnClick With* property (in the Authentic group of properties, assignable in the Properties sidebar, see screenshot below), you can specify text content of the parent node of a **button** or **hyperlink**.

![Properties screenshot](image)

When the Authentic View user clicks the design component (a button or hyperlink), the text you, the SPS designer, have specified will be inserted as the XML content of the design component's parent node. This enables you to allow the Authentic View user to make certain selections (by clicking a button or hyperlink) that will alter the presentation of the XML document in Authentic View.

---

**Usage mechanism**

This feature can be used as follows:

1. In the design, insert the design component (button or hyperlink) within the node for which you wish to insert text content.
2. Select the design component in the design, and, in the Properties sidebar, select that design component (in the **Properties For** column), and enter a value for the *Replace Parent Node OnClick With* property (which is in the Authentic group of properties; see screenshot above). This value can be entered as a text value directly or via an XPath expression. (Click the XPath icon in the toolbar of the Properties sidebar to enter the value as an XPath expression). In the screenshot above, the value is an XPath string expression `'default'`. Now, if the Authentic View user clicks the design component in Authentic View, the text *default* will be entered as the text content of the parent node of the design component. This result can also be obtained if you enter a text value (not an XPath expression) of *default* directly as the value of the *Replace Parent Node OnClick With* property.
3. Now that you know what content the parent node will have when the Authentic View user clicks the design component, you can set up processing based on the text content of the parent node. For example, using a **condition**, you can specify that when the content of the parent node is *default*, then some default formatting is applied; or, when the content is *hide*, the text of a node is hidden. In this way you can use this feature to achieve powerful effects that might otherwise have required the deployment of **scripts**.
Note: For this feature to work correctly, in the schema, the parent node must be defined to allow text content.

Usage with variables
The parent node can also be a variable that has been defined on an ancestor node. The value of the variable can then be set up by the SPS designer using the Replace Parent Node OnClick With property. When the Authentic View user clicks the design component (button or hyperlink), the value defined for the Replace Parent Node OnClick With property of the design component is assigned to the variable. You can now set up alternative processing options based on the value of the variable.

A simple example is shown below. When the value of the variable Var2 is set to Boolean true(), a button enabling the value to be changed to Boolean false() is displayed (see screenshot below). This processing is specified in the first branch of a condition.

Writing to a variable
Two condition branches, both with a button:

The Otherwise branch of the condition specifies processing that displays a button enabling the value of the variable to be changed to Boolean true() (screenshot below). You can define additional processing according to the value of the variable Var2 (true() or false()).
In this way you, the SPS designer, can allow Authentic View users to select from a range of options that then produce specific and corresponding processing of the document.

- See also
  - Buttons and Hyperlinks
  - Conditions
  - Variables
12.4 Additional Validation

The Additional Validation setting is available when the selected component is an element or attribute node, contents [contents placeholder], a data-entry device, or an Auto-Calculation. You can set an XPath expression to define the validity range of the XML content of the node or Auto-Calculation. An XML value that falls outside this defined range is invalid. If the XML value of the node is invalid, this is made known to the Authentic View user by means of an error message when the XML document is validated (F8). The error message that is displayed is the text you enter into the Error message field of the Additional Validation setting.

Setting Additional Validation

To set additional validation, do the following:

1. Select the component for which additional validation is required.
2. In the Properties sidebar, select the Authentic group of properties, and click the Edit button [edit button] of the Additional Validation property (screenshot below). This pops up the Additional Validation dialog.

3. In the Additional Validation dialog (screenshot below), add a row for an Additional Validation entry by clicking the Add button near the top left of the pane.

4. In the XPath expression column, enter an XPath expression to define the validity range of the XML data in that component.
5. Enter an error message to display when the data is invalid.
6. Click **OK** to finish.

- **See also**
  - Properties sidebar
12.5 Unparsed Entity URIs

If you are using a DTD and have declared an unparsed entity in it, you can use the URI associated with that entity for image and hyperlink targets in the SPS. This is useful if you wish to use the same URI multiple times in the SPS. This feature makes use of the XSLT function `unparsed-entity-uri` to pass the URI of the unparsed entity from the DTD to the output, and is therefore only available in the outputs (HTML, RTF, PDF, Word 2007+); not in Authentic View.

Using this feature requires that the DTD, XML document, and SPS documents be appropriately edited, as follows:

1. In the DTD, the unparsed entities must be declared, with (i) the URI, and (ii) the notation (which indicates to StyleVision the resource type of the entity).
2. In the XML document, the unparsed entity must be referenced. This is done by giving the names of the required unparsed entities.
3. In the SPS, unparsed entities can be used to target images and hyperlinks by correctly accessing the relevant dynamic node values as unparsed entities.

---

Declaring and referencing unparsed entities

Given below is a cut-down listing of an XML document. It has an internal DTD subset which declares two unparsed entities, one with a GIF notation (indicating a GIF image) and the other with an LNK notation (indicating a hyperlink). The `img/@src` and `link/@href` nodes in the XML code reference the unparsed entities by giving their names.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE document SYSTEM "UEURIDoc.dtd" [ 
<!ENTITY Picture SYSTEM "nanonull.gif" NDATA GIF>
<!ENTITY AltovaURI SYSTEM "http://www.altova.com" NDATA LNK> ]>
<document>
  <header>Example of How to Use Unparsed Entity URIs</header>
  <para>...</para>
  <img src="Picture"/> 
  <link href="AltovaURI">Link to the Altova Website.</link>
</document>
```

---

**SPS images and hyperlinks that use unparsed entities**

Images and hyperlinks in the SPS that reference unparsed entity URIs are used as follows:

1. Insert the image or hyperlink via the **Insert** menu.
2. In the Edit dialog of each, select the Dynamic tab properties (screenshot below), and enter an XPath expression that selects the node containing the name of the unparsed entity. In the XML document example given above, these nodes would be, respectively, the `//img/@src` and `//link/@href` nodes.
3. Then check the Treat as Unparsed Entity check box at the bottom of the dialog. This causes the content of the selected node to be read as an unparsed entity. If an unparsed entity of that name is declared, the URI associated with that unparsed entity is used to locate the resource (image or hyperlink).

When the stylesheet is processed, the URI associated with the entity name is substituted for the entity name.

**Note:** Note that if the URI is a relative URI, the XSLT processor expands it to an absolute URI applying the base URI of the DTD. So if the unparsed entity is associated with the relative URI "nanonull.gif", then this URI will be expanded to file:///c:/someFolder/nanonull.gif, where the DTD is in the folder someFolder.

- **See also**
  - Using Graphics
  - Defining Hyperlinks
12.6 New from XSLT, XSL-FO or FO File

An SPS design can be based on existing XSLT files that were designed for HTML output or XSLT files with XSL-FO commands for output in PDF or FO files. This means that SPS files do not have to be designed from scratch, but can take an already existing XSLT file as a starting point.

Steps for creating an SPS from XSLT
The steps for creating an SPS file from an XSLT, XSLT-for-FO, or FO file are as follows.

1. Select the command File | New | New from XSLT, XSL-FO or FO File.
2. In the Open dialog that appears, browse for the file you want.
3. In the next dialog you will be prompted to select a schema on which the SPS is to be based. Select the schema you want.
4. An SPS based on the structure and formatting in the XSLT or FO file will be created and displayed in Design View.
5. You can now modify the SPS in the usual way. For example, you could drag in nodes from the Schema Tree, modify the styling and presentation or add additional styling, and use StyleVision functionality such as Auto-Calculations and Conditional Templates.
6. You can save the SPS and use a Working XML File to preview various output formats. Subsequently you can generate stylesheets and output files using the Save Generated Files command.

Example
The example discussed below is located in the (My) Documents folder, C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2015\StyleVisionExamples\Tutorial\NewFromXSLT. This folder contains the files: SimpleExample.xslt, SimpleExample.xsd, and SimpleExample.xml.

The XML file is shown below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="YearlySales.xsd">
  <ChartType>Pie Chart 2D</ChartType>
  <Region id="Americas">
    <Year id="2005">30000</Year>
    <Year id="2006">90000</Year>
    <Year id="2007">120000</Year>
    <Year id="2008">180000</Year>
    <Year id="2009">140000</Year>
    <Year id="2010">100000</Year>
  </Region>
  <Region id="Europe">
    <Year id="2005">50000</Year>
  </Region>
</Data>
```
The XSLT file is as follows:

```
<?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">
  <xsl:template match="/">
    <html>
      <head>
        <title>Simple Example for New From XSLT</title>
      </head>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="Data">
    <xsl:for-each select="Region">
      <h1 style="color:red">
        <xsl:apply-templates select="@id"/>
      </h1>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Follow the steps 1 to 4 listed above to obtain the SPS in Design View. The SPS will look something like this:
Notice that the two templates in the XSLT have been created in the SPS. Now switch to the HTML Preview (screenshot below), and notice that the h1 element's styling (color:red) has been also passed to the SPS.

In Design View select the h1 element and change its color to black (in the Styles sidebar, in the Color group of properties). Then, from the Schema Tree, drag the Year element and create it as a table at the location shown in the screenshot below. Reverse the contents of the two columns so that the Year ID is in the first column.
You can make additional changes in the content, structure, and presentation properties of the document, then preview the output and save files using the **Save Generated Files** command.

- **See also**
  - SPS File: Content
  - SPS File: Structure
  - SPS File: Presentation
12.7 **User-Defined XPath Functions**

The SPS designer can define customized **XPath 2.0/3.0** functions. A user-defined XPath function can be re-used in any design component that accepts an XPath expression, for example, in Auto-Calculations, conditions, and combo boxes.

---

**Defining and editing user-defined XPath functions**

User-defined XPath functions are created (and subsequently accessed for editing) in either the Schema Tree sidebar or the Design Tree sidebar (*see screenshot below*). All the user-defined XPath functions in an SPS are listed under the XPath Functions item in both the Schema Tree and Design Tree sidebars and can be accessed via either sidebar.

To create a user-defined XPath function, click the icon of the XPath Functions item. This pops up the XPath Functions dialog (*screenshot below*). If you wish to edit a function that has already been created, double-click its entry in the list of XPath functions. The XPath Functions dialog (*screenshot below*) will appear and the function definition can be edited.
After a user-defined XPath function is created, it is available for use anywhere in the design.

**Namespace of user-defined XPath functions**

User-defined XPath functions are created in the namespace: http://www.altova.com/StyleVision/user-xpath-functions. This namespace is bound to the prefix sps;, so user-defined XPath functions must be called using this namespace prefix. For example, sps:MyFunction().

**Enabling and disabling user-defined XPath functions**

Each user-defined XPath function can be enabled or disabled by, respectively, checking or unchecking the check box to the left of the function's entry in the list of user-defined XPath functions (see screenshot below).
This feature is useful if two functions have the same name. Such a situation could arise, for example, when an imported SPS module contains a function having the same name.

---

### Calling a user-defined XPath function

A user-defined XPath function can be called in an XPath expression at any location in the design. For example, the user-defined XPath function `sps:MyFunction` defined above can be called, for example, with the following XPath expression in an Auto-Calculation:

```
sps:MyFunction()/@name.
```

This XPath expression would be evaluated as follows:

1. The `sps:MyFunction()` function is evaluated. Let’s say the function is defined as follows:

   ```xml
   $XML/Trades/Stock[@name=$XML/Trades/Selection/Stock]
   ```

   When the function is evaluated it returns the `/Trades/Stock` element that has a `name` attribute with a value that matches the content of the `/Trades/Selection/Stock` element.

2. The result of Step 1 is returned to the XPath expression in the function call. Now the value of the `name` attribute of this `/Trades/Stock` element is returned as the value of the Auto-Calculation.

---

### Deleting a function

To delete a function, select it in the XPath Functions list in the Schema Tree or Design Tree sidebar and then click the Remove Item icon in the toolbar of the sidebar. Alternatively, you can right-click the XPath function and select Remove Item from the context menu.
Defining an XPath Function

A user-defined XPath function requires: (i) a name (a text string), and (ii) a definition (an XPath expression).

Additionally, you can specify one or more parameters for the function. A user-defined XPath function can also have an optional return type (specified by selecting a type from the dropdown list of the Return Type combo box). A return type is useful if you wish to check that the datatype of the returned value conforms to the selected datatype. Note that the return value is not converted to the selected datatype. If there is a type mismatch, an error is returned. If no return type is specified, no datatype check is carried out.

After a user-defined XPath function is created, it is available for use anywhere in the design. In the XSLT stylesheet, it is created as an xsl:function element that is a child of the xsl:stylesheet element, as shown in the listing below.

```xml
<xsl:stylesheet>
  ...
  <xsl:function name="sps:Stock">
    <xsl:sequence select="$XML/Trades/Stock[@name=$XML/Trades/Selection/Stock]"/>
  </xsl:function>
  <xsl:function name="sps:Average" as="xs:decimal">
    <xsl:param name="a" as="xs:integer"/>
    <xsl:param name="b" as="xs:integer"/>
    <xsl:param name="c" as="xs:integer"/>
    <xsl:sequence select="avg( ($a, $b, $c) )"/>
  </xsl:function>
</xsl:stylesheet>
```

The sps:Stock function shown in the screenshot below and listed above returns the /Trades/Stock element that has a name attribute with a value that matches the content of the /Trades/Selection/Stock element. The sps:Average function listed above returns the average of three input parameter-values. The function definition uses the avg() function of XPath 2.0/3.0. The return datatype is specified to be of the xs:decimal type, which is the datatype returned by the avg() function when evaluating input values of datatype xs:integer. If the return type is specified, then the datatype of the return value is checked to see if it conforms with the specified type. If it doesn’t, an error is returned.
Defining the function

To define a function, click the icon of the XPath Functions item in the Schema Tree or Design Tree. This pops up the XPath Functions dialog (screenshot above). If you wish to edit a function that has already been created, double-click its entry in the list of XPath functions. Then enter a name for the function and a definition in the Function Body pane. Parameter definitions can be entered if required (see the next two sections, Parameters and Sequences and Parameters and Nodes, for details). A return type for the function can also be specified (see above).

The most important point to bear in mind when writing the XPath expression that defines XPath function is that there is no context node for the XPath expression. If the XPath expression must locate a node then the context node for the expression can be provided in one of the following ways:

1. The XPath expression starts with the document root. The document root is specified in the first location step of the XPath expression as $XML. For example, the XPath expression $XML/Trades/Stock[1] locates the first Stock child element of the /Trades element. The variable $XML (which locates the document root of the main schema) is defined globally by StyleVision in all SPS designs.
2. The context node can be passed as a parameter. See the section Parameters and Nodes below for an explanation.
In the following cases, errors are returned:

- If a parameter is defined but is not used in the body of the definition.
- If the datatype of the value returned by the function does not match the return type defined for the function.
- If any function in the SPS contains an error, an XSLT error is generated for the whole design, even if the function containing the error is not called. Note, however, that a function can be disabled by unchecking its check box in the list of user-defined XPath functions. When disabled in the design, the function is not included in the XSLT document generated from the design. In this way, an XPath expression containing an error can be excluded from the XSLT and no XSLT error will be generated.
Reusing Functions to Locate Nodes

In the previous section we saw how an XPath function can be built to locate a node. The `sps:Stock` function which is defined as shown in the screenshot below returns the `/Trades/Stock` element that has a `name` attribute with a value that matches the content of the `/Trades/Selection/Stock` element.

We could modularize the location steps of the XPath expression `$/XML/Trades/Stock[@name=$XML/Trades/Selection/Stock]` into separate XPath functions. For example as follows:

- The function `sps:Stocks()`, with the definition: `$/XML/Trades/Stock`
- The function `sps:SelectedStock()`, with the definition: `$/XML/Trades/Selection/Stock`

The whole XPath expression can then be written in another XPath expression as:

```
sps:Stocks()[@name=sps:SelectedStock()]
```

When XPath functions are created in this way to locate a node or nodeset, these functions can be re-used in other XPath expressions across the SPS design, thus considerably simplifying the writing of complex XPath expressions.
Parameters in XPath Functions

A user-defined XPath function can be assigned any number of parameters. The function's parameters are defined in the Parameters pane of the XPath Functions dialog (see screenshot below). These parameters can then be used in the definition of the user-defined XPath function (in the Function Body pane).

User-defined XPath function mechanism

The steps below explain how an XPath function works.

1. In a function call (for example, in an Auto-Calculation), the number of arguments in the function call must match the number of parameters defined for the user-defined function (as defined in the Parameters pane of the user-defined function; see screenshot below). Additionally, the number of items submitted by each argument (in the function call) must match the Occurrence definition of the corresponding parameter. If a datatype restriction has been specified for a parameter (in the Type column of the Parameters pane), the value/s submitted by the argument must match this datatype.

2. The arguments passed to the function's parameters are then used in the XPath function (as defined in the Function Body pane; see screenshot below). The result obtained by evaluating the XPath expression is then checked against the optional Return Type definition (see screenshot below). If the datatype is as expected, the result is used in the XPath expression from which the function was called.

Order of parameters

The order of the user-defined function's parameters is important because, when the function is called, the arguments submitted in the function call will be assigned to the parameters according to the order in which they are defined in the Parameters pane (see screenshot below).
So if the `sps:Stock` user-defined XPath function is defined as in the screenshot above and if it is called with the following XPath expression:

```xml
sps:Stock($XML, Node1, Node2)
```

then these three arguments—$XML, Node1, Node2—will be assigned, in that order, respectively, to the parameters `$ContextStock`, `$Selection`, and `$StockInfo`.

Note that each argument in the function call is separated from the next by a comma. So, each argument, as delimited by the commas in the function call, will be passed to the corresponding parameter (as ordered in the Parameters pane; see screenshot above).

The order of parameters in the Parameters pane can be controlled with the Append, Insert, and Delete icons of the Parameters pane.

---

**Datatype of parameters**

Optionally, the datatypes of parameters of the user-defined function can be defined. If a datatype is specified, then the datatype of the incoming argument will be checked against the parameter's datatype, and an error will be returned if the types do not match. This feature enables the input data (from the function call's arguments) to be checked.
Occurrence
Each parameter of the user-defined XPath function can be considered to be a sequence. The Occurrence property of a parameter specifies how many items must be submitted for that parameter by the corresponding argument of the function-call.

In both function definitions and in function calls, commas are used to separate one parameter or argument from another as well as to separate items within a sequence. It is important, therefore, to note the context in which a comma is used: to separate parameters/arguments or to separate sequence items.

- In parameters/arguments, if required, parentheses are used to delimit sequences—in the function definition (parameters) or in the function call (arguments).
- In sequences, parentheses are ignored.

In this context, the following examples and points should be noted:

- **Parentheses in parameters/arguments:** Several XPath functions take a single sequence as an argument, for example, the avg() and count() functions. If this sequence is enumerated using comma separators or range operators, the sequence must be enclosed in parentheses to unambiguously show that it is a single sequence—and not multiple comma-separated sequences. For example, in the function avg((count($a), $b, $c)), the XPath 2.0 avg() function takes the single sequence (count($a),$b,$c) as its argument. Since the items of the sequence are enumerated, making up a sequence of three items, the sequence must be enclosed in parentheses and submitted as a single argument to the avg() function: avg((count($a),$b,$c)). Without the inner pair of parentheses, the definition of the avg() function would have three parameters, and that would be an error (since the avg() function expects one argument consisting of a single sequence).

- **No parentheses in parameters/arguments:** Similarly, the count() function also takes a single sequence as its one-parameter argument. However, since in our example count($a) the single sequence is not a comma-separated enumerated list, but is fetched instead by the variable/parameter $a, the argument does not need to be enclosed by an inner set of parentheses: So the expression count($a) is correct.

- **Parentheses and commas in function calls:** In a function call, parentheses must be correctly used so that each argument corresponds to a parameter (as defined in the Parameters pane of the XPath Functions dialog). For example, if a user-defined XPath function named MyAverage() is defined with the XPath 2.0 expression: avg((count($a),$b,$c)), then the following function call would be valid: MyAverage((1,2,3),4,5). The values corresponding to the three parameters $a,$b, and $c would be, respectively, the sequence (1,2,3), the singleton-sequence 4, and the singleton sequence 5. Singleton-sequences can, optionally, be enclosed in parentheses. The value returned by MyAverage() in this case would be 4.
Parameters and Sequences

It is important to note the relationship between parameters and sequences, and how parameters and sequences are used in XPath expressions. We use the following definitions to make these relationships clearer:

- A **sequence** consists of items that are either atomic values or nodes. A comma can be used to construct a sequence, by placing it between the items of a sequence and so allowing the sequence to be built.

- An XPath function can be defined to take **parameters**. For example, in the XPath 2.0 expression `count($a)`, the part within the function's parentheses is the parameter of the function and it must be a sequence of items.

- An **argument** consists of one or more items in a function call. For example, the function `count(//Person)` has one argument: `//Person`. This argument is valid because it returns one sequence of nodes, which corresponds to the signature of the `count()` function. (The signature of a function specifies the number of parameters and the expected datatype of each parameter. It also specifies what the function will return and the datatype of the returned object)

- The function `substring('StyleVisionExamples', 6, 6)`—which returns the string `Vision`—has three arguments. This is valid according to the signature of the `substring()` function, and is specified by it. When a function call has multiple arguments, these are separated by commas.

Parentheses as sequence delimiters

A key point to note when constructing XPath expressions is this: Parentheses are used to delimit sequences that use the comma separator or range operator to enumerate sequences. As a result, each parentheses-delimited sequence is read as one parameter (in function definitions) or one argument (in function calls).

Parentheses are not necessary around a path (or locator) expression (example of a path expression: `//Person/@salary`), because a path expression can be read unambiguously as one parameter or one argument. It is in fact a one-sequence parameter/argument.

Here are some examples to illustrate the points made above:

- `avg((10, 20, 30))` The `avg` function of XPath 2.0 takes one sequence of items as its single argument. Since this sequence is a comma-separated enumeration, the inner pair of parentheses are necessary in order to delimit the mandatory single sequence. Without the inner parentheses, the argument would have three arguments and therefore be invalid. (The outer pair of parentheses are the parentheses of the function.)

- `avg(/Person/@salary)` This path expression selects the `salary` attribute nodes of all `Person` elements and returns their attribute-values as the sequence to be evaluated (that is, to be averaged). No parentheses are required because the sequence is not enumerated before the argument is read. The argument is the single path (or locator) expression. The path expression is evaluated and the returned values are submitted to the function as the items of a sequence.

- `count((10 to 34))` This is an enumeration via the range operator. The range operator 'to' generates a sequence of comma-separated items (the integers from 10 to 34) before the argument is read. As a result, the `count()` function has within its argument a comma-separated sequence of 25 items. To read this as one single-sequence argument,
delimiting parentheses are required. Without such parentheses, the function call would have 25 arguments instead of one—thus invalidating the function call, since the `count()` function must, according to its signature, have only one argument.

- `count((10 to 34, 37))` The inner parentheses indicate that everything within them is the one argument of the function call—a single sequence consisting of 26 items.
- `count(/Person)` No sequence-delimiter parentheses are required around the single argument. The argument is a path expression that collects the `//Person` nodes in the XML document and returns these nodes as the items of the sequence to be counted.

**Using XPath parameters in XPath functions**

When parameters are used in the definition of a user-defined XPath function, ensure (i) that the number of arguments in a function call to this user-defined XPath function is correct, and (ii) that the arguments evaluate correctly to the expected type and occurrence.

The screenshot above defines three parameters (in the *Parameters* pane) and then uses these parameters (in the *Function Body* pane) to define an XPath function.

Each parameter that is defined in the *Parameters* can be considered to be a single sequence. The number of items allowed within the single sequence is specified with the *Occurrence* property. In
the definition above, each parameter is defined (in its Occurrence property) as a singleton-sequence (that is, a sequence of exactly one item). Each argument of the function call must therefore be a sequence of one item. The Type property specifies the datatype of the items of the sequence.

In the definition of our example XPath function (in the Function Body pane), each parameter provides one item of the sequence that is to be averaged. Since the XPath parameters together constitute a sequence, the sequence must be enclosed in parentheses to ensure that the entire sequence is read as the one parameter of the \texttt{avg()} function. If, at runtime, any of the arguments in the function call (corresponding to the three parameters) is not a singleton-sequence, an error is returned.

Given below are examples of XPath parameter usage in calls to the XPath function \texttt{ThreeAverage()} shown in the screenshot above. In Design View, you can insert an Auto-Calculation and give it the XPath expressions listed below to see the results. The function has been defined to take a sequence of three integers and average them.

- \texttt{sps:ThreeAverage(10,20,30)} returns 20. There are three valid arguments in the function call, corresponding to the three XPath parameters.
- \texttt{sps:ThreeAverage( (10),(20),(30) )} returns 20. There are three valid input arguments, corresponding to the three XPath parameters. Each input argument has been enclosed with parentheses (which are redundant, since each sequence is a singleton-sequence; however, this redundancy is not an error).
- \texttt{sps:ThreeAverage( (10),20,30 )} returns 20. There are three valid input arguments, corresponding to the three XPath parameters. The first argument has been enclosed with parentheses (redundant, but not an error).
- \texttt{sps:ThreeAverage( (10,20),(30),(40) )} returns an error because the first argument is not valid. It is not a singleton-sequence, as required by the property definition of the first \$a parameter ("Exactly one").
- \texttt{sps:ThreeAverage( (10,20,30) )} returns an error because only one input argument is submitted, inside the parentheses. Additionally, the argument is invalid because the sequence is not a singleton-sequence.

If the Occurrence property of a parameter is set to At-least-one (as in the definition shown in the screenshot below), then that parameter is defined as a sequence of one-or-more items.
In the definition above, the first parameter has been defined as a sequence of one or more items, the next two parameters as singleton-sequences. The function has been defined to count the number of items submitted by the first parameter, add the result to the sum of the two integers submitted by the other two parameters, and then divide the result by three to obtain the average. Notice the following:

- The sequence that is the parameter of the \( \text{avg()} \) function is enclosed in parentheses. This is to specify that the \( \text{avg()} \) function takes a single sequence consisting of three items as its parameter. The single sequence consists of three integers: the first submitted by the \( \text{count()} \) function; the second and third are the two parameters \( b \) and \( c \).
- The argument of the \( \text{count()} \) function is not enclosed in sequence-delimiter parentheses because the argument is unambiguously a single sequence.

Here are examples of parameter usage in calls to the XPath function \( \text{Average()} \) shown in the screenshot above.

- \( \text{sps:Average((1,2),3,4)} \) returns 3. There are three valid input arguments, corresponding to the three parameters. The first argument is enclosed in parentheses to delimit it. When the \( \text{count()} \) function operates on it, the function will return the value 2, which will be the first item of the sequence submitted to the \( \text{avg()} \) function.
- \( \text{sps:Average( 4,4,4 )} \) returns 3. There are three valid input arguments. The first argument is allowed to be a sequence of one item (see the \textit{Occurrence} property of its corresponding parameter). No parentheses are required to indicate separate arguments.
**Additional points of interest**
The following additional points should be noted:

- If an parameter is defined as having **At-least-one** occurrence, then a function such as `MyAverage()` could be defined with an XPath expression such as `avg((@$a))`. This function would accept an argument that is a single sequence of one-or-more items. The function could be called as follows: `sps:MyAverage((2,3,4))`, and it would return the value 3. The input argument must be enclosed in parentheses to ensure that the input is being read as a single sequence rather than as three singleton-sequences (which would be the case if there were no enclosing parentheses).

- If an XPath parameter $a is defined as having **None-or-one** occurrence, then a function such as `MyAverage()` could be defined with an XPath expression such as `avg(($a, $b, $c))`. This function would accept as its argument three sequences, with the possibility of the first sequence being empty. If the first sequence is to be empty, then an empty sequence must be explicitly submitted as the first input argument. Otherwise an error is reported. If the function were called as follows: `sps:MyAverage(30,20,10)`, it would return the value 20. The function could also be called with: `sps:MyAverage((),20,10)`, returning 15 (note that the empty sequence does count: as an input value of empty; for a return value of 10, the first item would have to be 0). The following, however, would generate an error: `sps:MyAverage(20,10)`, because no first empty sequence is supplied and, as a consequence, the third input argument is considered to be absent.

---

**Complex examples**
Besides providing the benefit of being able to re-use an XPath expression, user-defined XPath functions also enable the construction of complex customized XPath functions that are not available in the XPath 2.0 function set. For example, a factorial function could easily be constructed with an XPath expression that takes a singleton-sequence as its single parameter. If the parameter $num is the number to be factorialized, then the XPath expression to create the function would be:

```
if ($num < 2) then 1 else $num * sps:Factorial($num - 1)
```

If this function were called `Factorial()`, then the factorial of, say 6, could be obtained by calling the function with: `sps:Factorial(6)`. 
Parameters and Nodes

When using parameters in XPath functions that locate nodes, it is important to bear in mind that the function has no context node, no matter from where in the design it is called. The context node can be supplied either in the XPath expression that is used to define the function (that is, in the Function Body pane) or in the XPath expression that is used to call the XPath function. In the latter case, the context can be supplied via arguments in the function call.

Consider the user-defined XPath function `Stock()`, which is defined with three parameters as shown in the screenshot below.

The definition in the function body is `$ContextStock[@name=$Selection]/$StockInfo`, which uses the three parameters but contains no context node information. The context node information can be supplied in the XPath expression that calls the function, for example in this way:

```
sps:Stock( $XML/Trades/Stock, $XML/Trades/Selection/Stock, @name )
```

The function call has three arguments, the value of each of which supplies either context or node-locator information. Alternatively, the following XPath expressions can be used as the function-call and give the same results:

```
sps:Stock( /Trades/Stock, /Trades/Selection/Stock, @name )
sps:Stock( /Trades/Stock, //Selection/Stock, @name )
```
The $XML variable, which returns the document root, can be left out in function calls from design components because in the XPath expressions of design components the context node is known.

Notice that in the function-call listed above there are three input arguments corresponding respectively to the three parameters defined for the user-defined XPath function:

- $ContextStock = $XML/Trades/Stock (the /Trades/Stock element)
- $Selection = $XML/Trades/Selection/Stock (the /Trades/Selection/Stock element)
- $StockInfo = @name

The XPath expression in the function definition is:

$ContextStock[@name=$Selection]/$StockInfo

When the input arguments are substituted, the XPath expression in the function definition becomes:

$XML/Trades/Stock[@name=$XML/Trades/Selection/Stock]/@name

It is important to note that it is the nodesets that are passed to the function, not the text strings.

It is in this way that the context node and location steps are passed to the function via parameters. The function can then be evaluated to locate and return the required nodes.
12.8 Working with Dates

If the source document contains nodes that take date values, using the `xs:date` or `xs:dateTime` datatypes in the underlying XML Schema makes available the powerful date and time manipulation features of XPath 2.0/3.0 (see examples below). StyleVision supports the `xs:date` or `xs:dateTime` datatypes by providing:

1. A graphical date picker to help Authentic View users enter dates in the correct lexical format of the datatype for that node. The date picker, since it is intended for data input, is available only in Authentic View.
2. A wide range of date formatting possibilities via the Input Formatting feature.

These StyleVision features are described in the sub-sections of this section: Using the Date-Picker and Formatting Dates. In the rest of the introduction to this section, we show how XPath 2.0 can be used to make calculations that involve dates.

Note: Date and time data cannot be manipulated with XPath 1.0. However, with XPath 1.0 you can still use the Date Picker to maintain data integrity and use Input Formatting to provide date formatting.

---

Date calculations with XPath 2.0

Data involving dates can be manipulated with XPath 2.0 expressions in Auto-Calculations. Given below are a few examples of what can be achieved with XPath 2.0 expressions.

- The XPath 2.0 functions `current-date()` and `current-dateTime()` can be used to obtain the current date and date-time, respectively.
- Dates can be subtracted. For example: `current-date() - DueDate` would return an `xdt:dayTimeDuration` value; for example, something like `P24D`, which indicates a positive difference of 24 days.
- Time units can be extracted from durations using XPath 2.0 functions. For example: `days-from-duration(xdt:dayTimeDuration('P24D'))` would return the integer 24.

Here is an XPath 2.0 expression in an Auto-Calculation. It calculates a 4% annual interest on an overdue amount on a per-day basis and returns the sum of the principal amount and the accumulated interest:

```
if (current-date() gt DueDate)
then (round-half-to-even(InvoiceAmount +
  (InvoiceAmount*0.04 div 360 *
   days-from-duration((current-date() -
   DueDate)))), 2))
else InvoiceAmount
```

Such a calculation would be possible with XPath 2.0 only if the `DueDate` element were defined to be of a date type such as `xs:date` and the content of the element is entered in its lexically correct form, that is, `YYYY-MM-DD[±HH:MM]`, where the timezone component (prefixed by `±`) is optional. Using the Date Picker ensures that the date is entered in the correct lexical form.
• Formatting Dates
• Using the Date-Picker
• Insert Date Picker
• Auto-add Date Picker
Using the Date-Picker

The Date Picker (screenshot below) is a graphical calendar in Authentic View for entering dates in the correct lexical format for nodes of xs:date and xs:dateTime datatype.

The lexical format is entered appropriately according to the datatype:

- For xs:date, the format of the entry is `YYYY-MM-DD[±HH:MM]`, where the timezone component (prefixed by ±) is optional according to the XML Schema specification. A value for the timezone component can be selected in the Date Picker.
- For xs:dateTime, the format of the entry is `YYYY-MM-DDTHH:MM:SS[±HH:MM]`. The timezone component (prefixed by ±) is optional according to the XML Schema specification. A value for the timezone component can be selected by the user.

Inserting and deleting a Date Picker in the design

A Date Picker can be inserted in the SPS design: (i) for any node that is of datatype xs:date or xs:dateTime, and (ii) when that node is created either as contents or as an input field. A Date Picker can be inserted in one of two ways:

- By default when a node of datatype xs:date or xs:dateTime is created in the SPS. To set this default, toggle the Auto-Add Date Picker feature ON. Do this by selecting/de-selecting the command Authentic | Auto-add Date Picker. When the Auto-Addition of the Date Picker is switched on, the Date Picker is inserted when any element of datatype xs:date or xs:dateTime is created as either contents or an input field, or changed to either of these two components.
- By clicking the menu command Insert | Date Picker when the cursor is at the desired location within the xs:date or xs:dateTime node in the SPS. This command can be accessed via the Insert menu, or via the context menu when the cursor is within the xs:date or xs:dateTime node.

When the Date Picker is inserted, the Date Picker icon appears at that location. To delete a Date Picker, use the Delete or Backspace buttons.

Using the Date Picker in Authentic View

In Authentic View, the Date Picker appears as an icon (screenshot below).
To modify the date, click the icon. This pops up the Date Picker (screenshot below). To enter a new date, select the required date in the Date Picker. The date will be entered in the correct lexical format according to that node's datatype.

To enter a timezone, click the Timezone button, which is set to a default of No Timezone. The timezone will be entered in the lexical format appropriate to the node's datatype (screenshot below).

- See also
  - Insert Date Picker
  - Auto-add Date Picker
  - Working with Dates
  - Formatting Dates
Formatting Dates

A date in an XML document is saved in the format specific to the datatype of its node. For example, the value of an `xs:date` node will have the format `YYYY-MM-DD[±HH:MM]`, while the value of an `xs:dateTime` node will have the format `YYYY-MM-DDTHH:MM:SS[±HH:MM]`. These formats are said to be the lexical representations of that data. By default, it is the lexical representation of the data that is displayed in Authentic View and the output. However, in the SPS, the Value Formatting feature can be used to display dates in alternative formats in Authentic View and, in some cases, optionally in the output.

Value Formatting for dates can be used to define custom formats for nodes and Auto-Calculations of the following datatypes:

- `xs:date`
- `xs:dateTime`
- `xs:duration`
- `xs:gYear`
- `xs:gYearMonth`
- `xs:gMonth`
- `xs:gMonthDay`
- `xs:gDay`

Using Value Formatting to format date nodes

To format dates alternatively to the lexical format of the date node, do the following:

1. Select the `contents` placeholder or input field of the node. Note that value formatting can only be applied to nodes created as contents or an input field.
2. In the Properties sidebar, select the `content` item, and then the `Content` group of properties. Now click the Edit button of the `Value Formatting` property. This pops up the Value Formatting dialog (screenshot below).
By default, the *Unformatted* radio button (the standard lexical format for the node's datatype) is selected.

3. To define an alternative format, select the *Format* radio button.
4. You can now select a predefined date format from the drop-down list of the combo box (screenshot below), or define your own format in the input field of the combo box. See *Value Formatting Syntax* for details about the syntax to use when defining your own format.

---

**Using Value Formatting to format Auto-Calculations**

When Auto-Calculations evaluate to a value that is a lexical date format, Value Formatting can be used to format the display of the result. Do this as follows:

1. Select the Auto-Calculation in the design.
2. In the Properties sidebar, select the *content* item, and then the *AutoCalc* group of properties. Now click the Edit button of the *Value Formatting* property. This pops up the Value Formatting dialog (screenshot below).
By default, the *Unformatted* radio button is selected.

3. To define an alternative format, select the *Format* radio button.

4. In the Options for XML Schema value pane, in the *Datatype* combo box, select the *date* datatype to which the Auto-Calculation will evaluate. In the *Format* combo box, you can then select a predefined date format from the drop-down list (available options depend on the selected datatype), or define your own format in the input field of the combo box. See *Value Formatting Syntax* for details about the syntax to use when defining your own format.

---

**Applying Value Formatting to the output**

The Value Formatting that you define applies to Authentic View. Additionally, some Value Formatting definitions—not all—can also be applied to HTML, RTF, PDF, and Word 2007+ output. To do this, check the Apply Same Format to XSLT Output check box. If this option is not checked or if it is not available, then only Authentic View will display the Value Formatting; the output will display the value in its lexical format (for nodes) or, in the case of Auto-Calculations, in the format to which the Auto-Calculation evaluates.

---

**See also**

- Value Formatting (Formatting Numeric Datatypes)
- Using the Date-Picker
### 12.9 Using Scripts

In StyleVision, you can define JavaScript functions for each SPS in a JavaScript editor (available as a tab in the Design View). The function definitions created in this way are stored in the header of the HTML document and can be called from within the body of the HTML document. Such functions are useful when:

- You wish to achieve a complex result using multiple script statements. In this case it is convenient to write all the required scripts, as separate functions, in one location (the header) and refer to the functions subsequently in the design document.
- You wish to use a particular script at multiple locations in the design document.

How to define functions in the JavaScript Editor is described in the sub-section *Defining JavaScript Functions*.

In the GUI, all JavaScript functions which are defined for a given SPS in the JavaScript Editor are listed in the Design Tree window under the Scripts entry (screenshot below). The screenshot below indicates that four JavaScript functions, *Average*, *ImageOut*, *ImageOver*, and *Buttons*, are currently defined in the active SPS.

The functions defined in the JavaScript Editor are available as event handler calls within the GUI. When a component in the design document is selected, any of the defined functions can be assigned to an event handler property in the Event property group in the Properties sidebar (screenshot below). How to assign a JavaScript function to an event handler is described in the section *Assigning Function to Event Handlers*. 
Note: Scripts are applicable in the HTML output only. They are not applicable in Authentic View.

Scripts in modular SPSs
When an SPS module is added to another SPS module, the scripts in the added module are available within the referring SPS, and can be used as event handlers via the Properties sidebar for components in the referring SPS. For more information about using modular SPSs, see the section Modular SPSs.

See also
- Design View
- Design Tree
- Modular SPSs
Defining JavaScript Functions

To define JavaScript functions, do the following:

1. In Design View, switch to the JavaScript Editor by clicking the Design View tab and selecting JavaScript (screenshot below).

2. In the JavaScript Editor, type in the function definitions (see screenshot below).

```javascript
function DisplayTime()
{
    now = new Date();
    hours = now.getHours();
    mins = now.getMinutes();
    secs = now.getSeconds();
    result = hours + "." + mins + "." + secs;
    alert(result);
}

function ClearStatus()
{
    window.status="";
}
```

The screenshot above shows the definitions of two JavaScript functions: DisplayTime and ClearStatus. These have been described for the active SPS. They will be entered in the header of the HTML file as follows:

```html
<script language="javascript">
<!--
function DisplayTime()
{
    now = new Date();
    hours = now.getHours();
    mins = now.getMinutes();
    secs = now.getSeconds();
    result = hours + "." + mins + "." + secs;
    alert(result);
}

function ClearStatus()
{
    window.status="";
}
-->
</script>
```

These functions can now be called from anywhere in the HTML document. In StyleVision, all the defined functions are available as options that can be assigned to an event handler.
property in the Event property group in the Properties sidebar. See Assigning Function to Event Handlers for details.

See also
- Design View
- Assigning Functions to Event Handlers
Assigning Functions as Event Handlers

In the StyleVision GUI, you can assign JavaScript functions as event handlers for events that occur on the HTML renditions of SPS components. These event handlers will be used in the HTML output. The event handler for an available event—such as `onclick`—is set by assigning a global function as the event handler. In the Properties sidebar, global functions defined in the JavaScript Editor are available as event handlers in the dropdown boxes of each event in the Events property group for the selected component (screenshot below).

To assign a function to an event handler, do the following:

1. Select the component in the SPS for which the event handler is to be defined. The component can be a node or content of any kind, dynamic or static.
2. In the Properties sidebar select the Event group. This results in the available events being displayed in the Attribute column (screenshot above).
3. In the Value column of the required event, click the down arrow of the combo box. This drops down a list of all the functions defined in the JavaScript Editor.
4. From the dropdown list, select the required function as the event handler for that event.

In the HTML output, when that event is triggered on the component for which the event handler is defined, the JavaScript function is executed.

See also
- Design View
- Defining JavaScript Functions
External JavaScript Files

An SPS can access external JavaScript files in two ways:

1. **By creating a User-Defined Element or User-Defined XML Block.** These design objects can contain a `<SCRIPT>` element that accesses the external JavaScript file. Note that location of the User-Defined Element or User-Defined XML Block is within the `<BODY>` element of the design (and therefore within the `<BODY>` element of the HTML output, not within the `<HEAD>` element).

2. **By adding a script in the Javascript Editor** that accesses the external file. A script that is added in this way will be located in the `<HEAD>` element of the HTML output.

User-Defined Elements and User-Defined XML Blocks

External JavaScript files can be accessed by means of **User-Defined Elements** and **User-Defined XML Blocks**. Using these mechanisms, a `<SCRIPT>` element that accesses the external JavaScript file can be inserted at any location within the `<BODY>` element of the output HTML document.

A **User-Defined Element** could be inserted as follows:

1. Place the cursor at the location in the design where the `<SCRIPT>` element that accesses the JavaScript file is to be inserted.
2. From the **Insert** menu or context menu, select the command for inserting a **User-Defined Element**.
3. In the dialog that pops up (see screenshot above), enter the `<SCRIPT>` element as shown above, giving the URL of the JavaScript file as the value of the `src` attribute of the `<SCRIPT>` element: for example, `script type="text/javascript" src="file:///c:/Users/mam/Desktop/test.js"`.
4. Click **OK** to finish.

You can also use a **User-Defined XML Block** to achieve the same result. To do this use the same procedure as described above for User-Defined Elements, with the only differences being (i) that a **User-Defined XML Block** is inserted instead of a **User-Defined Element**, and (ii) that the `<SCRIPT>`
element is inserted as a complete XML block, that is, with start and end tags.

---

**JavaScript Editor**

The **JavaScript Editor** enables you to insert an external script in the `<head>` element of the HTML output. Do this by entering, in the JavaScript Editor, the following script fragment, outside any other function definitions that you create.

```javascript
var script = document.createElement('script');
script.type = 'text/javascript';
script.src = 'file:///c:/Users/Desktop/test.js';
var head = document.getElementsByTagName('head')[0];
head.appendChild(script)
```

The external JavaScript file that is located by the URL in `script.src` is accessed from within the `<head>` element of the output HTML document.

- **See also**
  - Defining JavaScript Functions
  - User-Defined Elements
  - User-Defined XML Text Blocks
  - Assigning Functions to Event Handlers
12.10 HTML Import

In StyleVision you can import an HTML file and create the following documents based on it:

- An SPS document based on the design and structure of the imported HTML file.
- An XML Schema, in which HTML document components are created as schema elements or attributes. Optionally, additional elements and attributes that are not related to the HTML document can be created in the user-defined schema.
- An XML document with: (i) a structure based on the XML Schema you have created, and (ii) content from the HTML file.
- XSLT stylesheets based on the design in Design View.

HTML-to-XML: step-by-step

The HTML Import mechanism, which enables the creation of XML files based on the imported HTML file, consists of the following steps:

1. **Creating New SPS via HTML Import.** When an HTML file is imported into StyleVision, a new SPS document is created. The HTML document is displayed in Design View with HTML markup tags. A user-defined XML Schema with a document element called UserRoot is created in the Schema Tree window. This is the schema on which the SPS is based. The HTML document content and markup that is displayed in Design View at this point is included in the SPS as static content.

2. **Creating the Schema and SPS Design.** Create the schema by (i) dragging components from the HTML document to the required location in the schema tree (in the Schema Tree window); and, optionally, (ii) adding your own nodes to the schema tree. In the Design Window, HTML content that has been used to build nodes in the schema tree will now be displayed with schema node tags around the content. HTML content that has no corresponding schema node will continue to be displayed without schema node tags.

3. In the Design Document, assign formatting to nodes, refine processing rules, or add static content as required. These modifications will have an effect only on the SPS and the generated XSLT. It will not have an effect on either the generated schema or XML file.

4. After you have completed the schema tree and the design of the SPS, you can **generate and save** the following:
   - an XML Schema corresponding to the schema tree you have created;
   - an XML data file with a structure based on the schema and content for schema nodes that are created with the (content) placeholder in the SPS design;
   - a SPS (.sps file) and/or XSLT stylesheet based on your design.

- **See also**
  - User-Defined Schemas
  - New from XSLT
Creating New SPS via HTML Import

To create a new SPS file from an HTML document, do the following:

1. Select the menu command File | New | New from HTML File.
2. In the Open dialog that pops up, browse for the HTML file you wish to import. Select it and click Open.
3. You will be asked whether relative paths should be converted to absolute paths. Make your choice by clicking either Yes or No.

A new SPS document is created. The document is displayed in Design View and is marked up with the predefined HTML formats available in StyleVision (screenshot below).

```
<h1>A Sample Article</h1>

This is a simple article, marked up in HTML. It is the example HTML document used to demonstrate how STYLEVISION can generate schema files, XML files, and stylesheets from an HTML document. It contains headlines, paragraphs, two lists, a table, an image, a hyperlink, horizontal rules, and inline italics. A number of these elements have attributes.

It is important to note that the HTML-to-XML conversion involves two parallel processes:

- Creation of a schema tree
- Design of a STYLEVISION Power Stylesheet (SPS)
```

Note that the HTML document is displayed within the main template. There is no global template.

In the Schema Tree sidebar, a user-defined schema is created (screenshot below) with a root element (document element) called UserRoot.

Note that there is no global element in the All Global Elements list.
SPS structure and design
The SPS contains a single template—the main template—which is applied to the document node of a temporary internal XML document. This XML document has the structure of the user-defined schema which was created in the Schema Tree window. In Design View, at this point, the HTML document components within the main template are included in the SPS as static components. The representation of these HTML components in Authentic View will be as non-editable, non-XML content. The XSLT stylesheets will contain these HTML components as literal result elements. The schema, at this point, has only the document element Root; consequently, the temporary internal XML document contains only the document element Root with no child node.

When you create HTML selections as elements and attributes in the user-defined schema, you can do this in either of two ways:

1. By converting the selection to an element or attribute. In the design, the node tags are inserted with a \(\text{(content)}\) placeholder within the tag. In the schema, an element or attribute is created. In the XML document, the selection is converted to the text content of the schema node which is created in the XML document. The contents of the node created in the XML document will be inserted dynamically into the output obtained via the SPS.

2. By surrounding the selection with an element or attribute. In the design, the selection is surrounded by the node tags; no \(\text{(content)}\) placeholder is inserted. This means that the selection is present in the SPS design as static content. In the schema, an element or attribute is created. In the XML document, the node is created, but is empty. The static text which is within the schema node tags in the design will be output; no dynamic content will be output for this node unless a \(\text{(content)}\) placeholder for this node is explicitly inserted in the design.

The significance of the \(\text{(content)}\) placeholder is that it indicates locations in the design where data from the XML document will be displayed (in the output) and can be edited (in Authentic View).

- See also
  - User-Defined Schemas
Creating the Schema and SPS Design

The schema is created by dragging selections from Design View into the user-defined schema. You do this one selection at a time. The selection is dropped on a node in the schema tree (relative to which the new node will be created, either as a child or sibling). You select the type of the node to be created (element or attribute) and whether the selection is to be converted to the new node or surrounded by it.

The selection
The selection in Design View can be any of the following:

- A node in the HTML document.
- A text string within a node.
- Adjacent text strings across nodes.
- An image.
- A link.
- A table.
- A list.
- A combination of any of the above.

In this section we explain the process in general for any selection. The special cases of tables and lists are discussed in more detail in the section Creating Tables and Lists as Elements/Attributes.

To make a selection, click an HTML document component or highlight the required text string. If multiple components are to be selected, click and drag over the desired components to highlight the selection. Note that StyleVision extends the selection at the beginning and end of the selection to select higher-level elements till the first and last selected elements belong to the same parent.

The location in the schema tree
On dragging the selection over the desired schema tree node, the mouse pointer will be changed to one of the following symbols.

- Dropping the node when the Create as Sibling symbol ↓ appears, creates the selection as a sibling node of the node on which the selection is dropped.
- Dropping the node when the Create as Child symbol → appears, creates the selection as a child node of the node on which the selection is dropped.

You should select the node on which the selection is to be dropped according to whether the selection is to be created as a sibling or child of that node.

Selecting how the node is created
When you drop the selection (see previous section), a context menu pops up (screenshot below) in which you make two choices: (i) whether the node is to be created as an element or attribute; (ii) whether the selection is to be converted to the node or whether the node is to simply surround the selection.

The following points should be noted:

- When a selection is converted to a node (element or attribute), the node tags, together with a contained (content) placeholder, replace the selection in the design. In the design and the output the text content of the selection is removed from the static content. In the output, the text of the selection appears as dynamic content of the node in the XML document.
- If an HTML node is converted to an XML node, the XML node tags are inserted within the HTML node tags.
- When a selection (including HTML node selections) is surrounded by an XML node, the XML node tags are inserted before and after the selection. In the design and the output, the text content of the selection is retained as static text.
- The inserted node tags are inserted with the necessary path (that is, with ancestor node tags that establish a path relative to the containing node). The path will be absolute or relative depending on the context of the node in the design.
- How to create nodes from table and list selections are described in Creating Tables and Lists as Elements/Attributes.

---

Adding and deleting nodes in the schema

You can add additional nodes (which are not based on an HTML selection) to the user-defined schema. Do this by right-clicking on a node and selecting the required command from the context menu. Alternatively, you can use the toolbar icons of the Schema Tree sidebar.

To delete a node, select the node and then use either the context menu or the toolbar icon. Note, however, that when a node is deleted, some paths in the design could be invalidated.

---

Modifying the design
You can modify the structure of the design by dragging components around and by inserting static and dynamic components. Styles can also be modified using the various styling capabilities of StyleVision.

See also

- User-Defined Schemas
Creating Tables and Lists as Elements/Attributes

Tables and lists in the HTML document can be converted to element or attribute nodes in the XML Schema so that they retain the table or list structure in the schema.

Converting a table to elements/attributes

To convert a table to schema nodes, do the following:

1. Select the HTML table by highlighting some text in it.
2. Drag it to the node in the schema tree as a sibling or child of which you want to create it.
3. Drop the node when the Create as Sibling symbol or Create as Child symbol appears.
4. In the context menu that now pops up (screenshot below), select the command Convert selected table/list to elements or Convert selected table/list to attributes according to whether you wish to create the contents of table cells as elements or attributes, respectively.
5. In the Convert Table dialog that pops up (screenshot below), select whether the table created in the SPS should be a static table or dynamic table.

If the static table option is selected, then for each cell in the table, a schema node is created. In the design, each node is inserted with the (content) placeholder. The data in the table cells is copied to the temporary internal XML document (and to the generated XML document). The dynamic table option is available when the structure of all rows in the table are identical. When created in the SPS, the rows of the dynamic table are represented by a single row in the design (because each row has the same structure). The table data will be copied to the XML file. The dynamic table can grow top/down (rows are arranged vertically relative to each other) or left/right (rows become columns and
extend from left to right). If you indicate that the first row/column is a header, then (i) a header row containing the column headers as static text is included in the design; and (ii) the schema element/attribute nodes take the header texts as their names. If the first row/column is not indicated as a header, then no header row is included in the design.

6. After you have selected the required option/s, click **Convert** to finish.

---

**Converting a list to elements/attributes**

To convert a list to schema nodes, do the following:

1. Select the HTML list by highlighting some text in it.
2. Drag it to the node in the schema tree as a sibling or child of which you want to create it.
3. Drop the node when the Create as Sibling symbol or Create as Child symbol appears.
4. In the context menu that now pops up (screenshot below), select the command **Convert selected table/list to elements** or **Convert selected table/list to attributes** according to whether you wish to create the contents of table cells as elements or attributes, respectively.

   ![context menu](screenshot)

5. In the Convert List dialog that pops up (screenshot below), select whether the list created in the SPS should be a static list or dynamic list.

   ![Convert List](screenshot)

If the **static list** option is selected, then for each list item, a schema node is created. In the design, each node is inserted with the text of the HTML list item included as static content of the list item. If the **dynamic list** option is selected, then each list item is represented by a single list item node in the design. In the design, the list item element is inserted with the *(content)* placeholder.

6. After you have selected the required option, click **Convert** to finish.

---

*See also*
- Working with Tables, for a description of how tables are used in an SPS.
- Creating Lists
Generating Output

After completing the SPS, you can generate the following output using the File | Save Generated Files command:

- Generated user-defined schema, which is the schema you have created in the Schema Tree sidebar.
- Generated user-defined XML data, which is an XML document based on the schema you have created and containing data imported from the HTML file.
- XSLT stylesheets for HTML, RTF, PDF, and Word 2007+ output.
- HTML, RTF, FO, PDF, and Word 2007+ output.
12.11 XML Signatures

An SPS can be designed with an XML signature configured for Authentic View. When XML signatures are enabled, 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 enabled in the Enterprise edition of StyleVision. If an SPS with an enabled signature is opened in StyleVision Professional or StyleVision Basic, a message indicating that the feature is not supported in these versions is displayed.

Note: XML signatures can be used, and will be verified, in the Authentic View of the Enterprise edition of the following Altova products: Authentic Desktop, Authentic Browser, XMLSpy, and StyleVision.

Working with XML signatures in StyleVision and Authentic View
This section provides an overview of how XML signatures are enabled in an SPS, how Authentic XML documents are signed with XML signatures, and how the signatures are verified.

1. **Enabling and configuring the XML signature:** In StyleVision, the Design Overview sidebar provides access to the Configure XML Signature dialog (screenshot below), in which you must enable the signature.

![Configure XML Signature](image)

In this dialog you also specify whether the Authentic View user can choose an own certificate/password and whether the Authentic XML document is signed automatically when it is saved. A certificate or password can be specified in the SPS by the SPS...
designer in the XML Signature Settings dialog, which is accessed via the XML Signature Settings button. The specified certificate/password is used as the default. It can be overridden by the Authentic View user's certificate/password if this has been allowed. When the Authentic View user signs the XML document, the certificate or password is used together with the XML data to generate the signature.

2. In the XML Signature Settings dialog, accessed via the Edit XML Signature Settings button of the Configure XML Signature dialog (see screenshot above), the following settings are specified:
   (i) The certificate or password to use. If a certificate is specified, the certificate's key information can be saved with the signature by checking the Append Keyinfo checkbox (at the bottom of the XML Signature Settings dialog); this obviates the need for the Authentic View user to specify a certificate during verification. If a password is specified, the password information can be saved in the SPS file. This enables the Authentic View user to verify the Authentic XML document without needing to enter the password to run the verification process.
   (ii) The placement of the signature: The XML signature can be included in the Authentic XML file, or be saved in a separate (detached) file. If the signature is in a separate file, it will have to be delivered together with the Authentic XML file and saved to the correct location relative to the Authentic XML file. Note also that if you place the XML signature within the Authentic XML file, the schema will need to be modified to accommodate the Signature element. This issue is discussed in the section, XML document validity.
   (iii) The transformation method: This specifies how the XML will be transformed before being processed to generate the XML signature. The options are described in the section describing the XML Signature Settings dialog.

3. Authentic View user actions: The following Authentic View user actions are possible:
   (i) 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 (screenshot 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.

(ii) Signing the document: The Authentic XML document can be signed either automatically or manually. Automatic signing is specified in the signature configuration and causes the 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. This issue is discussed in the section, XML document validity.

(iii) 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.

The sub-sections of this section describe in detail each of the processes discussed above:

- XML Signature Configuration
- XML Signature Settings
- Signing the Authentic XML Document
- Signature Verification in Authentic
- XML Document Validity
- Working with Certificates

See also
- PXF File: Container for SPS and Related Files
XML Signature Configuration

In the Configure XML Signatures dialog (screenshot below), the options listed below may be configured.

- **Enabling XML signatures**: Checking this option enables XML signature support in Authentic. As a result, Authentic View users can sign the Authentic XML document they have edited. Also, when a document is loaded in Authentic View, the signature-verification process is executed. If XML signature support is disabled, then signing in Authentic is not possible and the XML document, when loaded in Authentic View, will not run the signature-verification process.

- **Own certificate/password for Authentic users**: With this option the SPS designer can specify whether the Authentic View user may select an own certificate/password. (If this option is checked, the Authentic View user can select an own certificate/password via a button in the XML Signature dialog (see screenshot below), and that selection will replace the default certificate/password specified by the SPS designer.) If the SPS designer unchecks this option, a prompt appears asking for a certificate/password to be specified for the signature.

- **Show messages on load**: Select the individual options for messages. A message will be displayed each time the document is loaded and the selected condition is true.

- **Sign automatically on save**: If this option is checked, the Authentic XML document will be signed automatically when the document is saved. The certificate/password used will be that currently saved: either the certificate/password specified by the SPS designer, or the
certificate/password specified by the Authentic View user—if this option has been allowed.

Clicking the **Edit XML Signature Settings** button opens the XML Signature Settings dialog, in which you can specify: the default certificate/password to be used; where the signature will be placed; transformation options; etc.

- **See also**
  - XML Signature Settings
  - Signing the Authentic XML Document
**XML Signature Settings**

In the XML Signature Settings dialog (*screenshot below*), you can specify the default certificate/password to be used for the signature, where the signature will be placed, transformation options, whether certificate key information is saved with the XML file, etc. The individual settings are described below.

**Authentication method: certificate or password**

The signature can be based on a certificate or a password. Select the radio button of the method you wish to use.
• **Certificate**: Click the **Select** button and browse for the certificate you want. The certificate you select must have a private key. The signature is generated using the private key of the certificate. For the signature to be verified subsequently, access to the certificate (or to a public-key version of it) is required. The public key of the certificate is used to verify the signature. For more details about certificates, see the section *Working with Certificates*. Alternatively, the certificate's key information can be saved with the signature by checking the **Append Keyinfo** check box at the bottom of the dialog. In this case, no certificate will be referenced for verifying the signature; the key information saved with the signature will be used. **Note**: StyleVision's XML Signature feature supports only certificates of type RSA-SHA1 and DSA-SHA1.

• **Password**: Enter a password with a length of five to 16 characters. This password will subsequently be required to verify the signature. Alternatively, the password can be saved with the SPS file. In this case, verification will be carried out directly, without Authentic View prompting for a password.

The certificate/password you specify will be the default for the signature.

---

**Transformations**

The XML data is transformed and the result of the transformation is used for the creation of the signature. You can specify the canonicalization algorithm to be applied to the file's XML data (the SignedInfo content) prior to performing signature calculations. Significant points of difference between the algorithms are noted below:

- **Canonical XML with or without comments**: If comments are included for signature calculation, then any change to comments in the XML data will result in verification failure. Otherwise, comments may be modified or be added to the XML document after the document has been signed, and the signature will still be verified as authentic. The **Comments** option is available for detached signatures and is disabled for enveloped signatures.

- **Base64**: The root (or document) element of the XML document is considered to be Base64 encoded, and is read in its binary form. If the root element is not Base64, an error is returned or the element is read as empty.

- **None**: No transformation is carried out and the XML data from the binary file saved on disk is passed directly for signature creation. Any subsequent change in the data will result in a failed verification of the signature. However, if the **Strip Whitespace** check box option is selected, then all whitespace is stripped and changes in whitespace will be ignored. A major difference between the **None** option and a **Canonicalization** option is that canonicalization produces an XML data stream, in which some differences, such as attribute order, are normalized. Consequently, a canonicalization transformation will normalize any changes such as that of attribute order (so verification will succeed), while no-transformation will reflect such a change (verification will fail). Note, however, that a default canonicalization is performed if the signature is embedded (enveloped). So the XML data will be used as is (i.e. with no transformation), when the signature is detached, **None** is selected, and the **Strip Whitespace** checkbox is unchecked.

---

**Signature placement**
The signature can be placed within the XML file or be created as a separate file. The following options are available:

- **Enveloped**: The signature element is created as the last child element of the root (document) element.
- **Detached**: The XML signature is created as a separate file. In this case, you can specify the file extension of the signature file and whether the file name is created with: (i) the extension appended to the name of the XML file (for example, `test.xml.xsig`), or (ii) the extension replacing the XML extension of the XML file (for example, `test.xsig`). You can also specify whether, in the signature file, the reference to the XML file is a relative or an absolute path.

---

**Append key information**

The *Append Keyinfo* option is available when the signature is certificate-based. It is unavailable if the signature is password-based.

If the option is selected, public-key information is placed inside the signature, otherwise key information is not included in the signature. The advantage of including key information is that the certificate itself (specifically the public-key information in it) will not be required for the verification process (since the key information is present in the signature).

- **See also**
  - XML Signature Configuration
  - Signing the Authentic XML Document
  - XML Document Validity
  - Working with Certificates
Signing the Authentic XML Document

The Authentic XML document can be signed in two ways:

- Automatically when the Authentic XML document is saved. For this method to be used, the SPS designer must activate the **Sign Automatically on Save** option in the Configure XML Signature dialog.
- Manually via the **Sign Document** button in the XML Signature dialog (see screenshot below). This dialog is accessed by clicking the XML Signature toolbar icon or the **Authentic | XML Signature** menu command. This way of (manually) signing the XML document is described below.

Manually signing the Authentic XML document

In addition to automatic signing on save—a feature which the SPS designer enables via the SPS—the Authentic View user can sign the document manually.

The signature will be generated in accordance with the signature settings that are current at the time of signing. All of the signature settings except the certificate/password must be specified by the SPS designer in the Configure XML Signature and XML Signature Settings dialogs. The certificate/password can be specified by the SPS designer or by the Authentic View user. The following possibilities are available:

- The SPS designer specifies the certificate/password. The own certificate option is not available to the Authentic View user. See **SPS only** below for details.
- The SPS designer specifies the certificate/password. The own certificate option is available to the Authentic View user. See **SPS + Authentic View user** below for details.

**SPS only**

The SPS designer specifies the certificate/password. The Authentic View user is not given the option of selecting an own certificate/password (done by deactivating this option in the Configure XML Signature dialog). The Authentic View user signs the document using the SPS-specified certificate/password. There are two possibilities:

- If a certificate has been specified (as opposed to a password), then the Authentic View user must have access to the private-public-key certificate that has been specified. The private key of the certificate will be used to generate the signature.
- If a password has been specified in the signature settings, this password is used directly when signing. The Authentic View will not be asked for it when signing.
**Note:** Either the certificate or the password option must be set as the Authentication method in the XML Signature Settings dialog. However, no value needs to be set for the selected method. For example, the Password option can be selected but no password need be set. In this case, the Authentic View user must set a password when signing the document. Similarly, if the certificate method were selected but no certificate were specified, then the Authentic View user would need to specify a certificate for generating the signature.

---

**SPS + Authentic View user**

The SPS designer specifies the certificate/password. The Authentic View user may select an own certificate/password (this option is enabled in the Configure XML Signature dialog). The Authentic View user can now choose whether to use the certificate/password specified in the SPS file (the default certificate/password) or to use an own certificate/password. The own certificate/password is saved in memory and will be used for the current session only. When the SPS file or XML file is closed, the signature setting reverts to the default certificate/password.

To use an own certificate/password, the Authentic View user must do the following:

1. Click the XML Signature toolbar icon or the Authentic | XML Signature menu command to pop up the XML Signature dialog (*screenshot below*).

![XML Signature dialog](image)

2. Click the Select Own Certificate or Select Own Password button. (Which of these buttons appears depends on which of the two options (certificate or password) was selected in the XML Signature Settings dialog.)

3. Either browse for the certificate file or enter the password you want.

4. The Authentic View user can either sign the document right away (by clicking Sign Document) and/or click OK to finish.

On clicking OK, the user-defined certificate/password setting is saved to memory and will be used for the current session.

---

**See also**

- XML Signature Configuration
- XML Signature Settings
- Signature Verification in Authentic
- XML Document Validity
- Working with Certificates
Signature Verification in Authentic

Signature verification occurs each time the Authentic XML document is refreshed or reloaded. If the signature certificate's key information has been appended to the signature or if a signature's password is available in the SPS, then the verification is carried out directly using this authentication information. If the authentication information is not available, then a message to that effect is displayed. In this case, the signature will have to be verified manually.

Manual signature verification

To verify the signature of a signed document manually, the Authentic View user must click the XML Signature toolbar icon or the menu command Authentic | XML Signature.

If the authentication information has been saved with the signature (certificate's key information) or with the SPS (password), then the verification is carried out directly and the verification result is displayed, as in the screenshot below.

![XML Signature verified successfully](image)

If no authentication information is available, the Authentic View user will be prompted for a certificate or password. For example, in the screenshot below, the user is prompted to enter the password for signature verification.

![Signature Password](image)

Note that the XML Signature toolbar icon (or Authentic | XML Signature menu command) runs the verification process only if the document has already been signed. Otherwise, the command will display a Verification Failed message.
In this case, the option to sign the document is available.

See also
- XML Signature Configuration
- XML Signature Settings
- Signing the Authentic XML Document
- XML Document Validity
- Working with Certificates
XML Document Validity

If an XML signature is embedded in the XML document, a Signature element in the namespace http://www.w3.org/2000/09/xmldsig# is added to the XML document. In order for the document to remain valid according to a schema, the schema must contain the appropriate element declarations. StyleVision embeds signatures within the document. The signature is said to be Enveloped. The Signature element is created as the last child element of the root (or document) element.

If you do not wish to modify the schema of the XML document, the XML signature can be created in an external file. For more details, see the description of the placement options in the section, XML Signature Settings.

Given below are excerpts from XML Schemas that show how the Signature element of an enveloped signature can be allowed. You can use these examples as guides to modify your own schemas.

In the first of the two listings below, the XML Signature Schema is imported into the user's schema. The XML Signature Schema is located at the web address: http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi=http://www.w3.org/2000/09/xmldsig#
elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:element name="Root">
<xs:complexType>
<xs:sequence>
<xs:element ref="FirstChildOfRoot"/>
<xs:element ref="SecondChildOfRoot" minOccurs="0"/>
<xs:element ref="ThirdChildOfRoot" minOccurs="0"/>
<xs:element ref="xsig:Signature" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:element>
...
</xs:schema>
```

A second option (listing below) is to add a generic wildcard element which matches any element from other namespaces. Setting the processContents attribute to lax causes the validator to skip over this element—because no matching element declaration is found. Consequently, the user does not need to reference the XML Signatures Schema. The drawback of this option, however, is that any element (not just the Signature element) can be added at the specified location in the XML document without invalidating the XML document.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:element name="Root">
<xs:complexType>
...
</xs:element>
</xs:schema>
```
<xs:sequence>
  <xs:element ref="selection"/>
  <xs:element ref="newsitems" minOccurs="0"/>
  <xs:element ref="team" minOccurs="0"/>
  <xs:any namespace="##other" minOccurs="0" processContents="lax"/>
</xs:sequence>
</xs:complexType>
</xs:element>
...
</xs:schema>

W3C Specification
For more details about XML signatures, see the W3C specification for XML signatures at http://www.w3.org/TR/xmldsig-core/.

See also
- XML Signature Configuration
- XML Signature Settings
- Working with Certificates
Working with Certificates

Authorization certificates are commonly used to create and verify XML signatures. This section contains information about obtaining, importing, and exporting certificates. It is organized into the following sub-sections:

- Obtaining a certificate with a private-public-key pair
- Importing a private-public-key certificate
- The certificate stores on a Windows machine
- Exporting a public-key certificate

Obtaining a certificate with a private-public-key pair

A certificate can be obtained in the following ways:

- From a certificate authority. The certificate authority verifies the identity of the certificate's owner. Certificates obtained in this way are in contrast to self-signed certificates, which can be created by anyone with a certificate creation tool.
- By creating a self-signed certificate. Such certificates are not verified by any authority, but often provide adequate security. A number of certificate creation tools, such as Microsoft's Visual Studio, are available.

For use with XML signatures you will need a certificate with a private-public-key pair.

**Note:** StyleVision's XML Signature feature supports only certificates of type RSA-SHA1 and DSA-SHA1.

Importing a private-public-key certificate

After a private-public-key certificate has been obtained, you will need to import it to your Windows certificate store. Do this as follows:

1. Double-click the certificate file to open the Certificate Import Wizard (screenshot below), and click Next.
In the File to Import window, ensure that the certificate file is selected, then click **Next**.

3. Type in the password for the private key. You must know the password if you intend to use the private key to create an XML signature. The password for the private key will be supplied to you when you obtain the certificate. After typing in the password, click **Next**.

4. You can allow the wizard to automatically select the store in which to place the certificate—according to the certificate type—or you can select the store yourself. (It might be better to select the store yourself, so you know the location of the certificate.) Click **Next** when done.

5. Click **Finish** to complete the process.

---

**Certificate stores on a Windows machine**

The certificate store on a Windows XP machine can be accessed as follows:

1. In the Start menu, select **Run**.
2. Type in `mmc` and click **OK**. A Console window pops up (screenshot below).
3. In the Console window, select the command File | Add/Remove Snap-in.
4. In the Standalone tab of the Add/Remove Snap-in dialog that pops up, click Add.
5. In the Add Standalone Snap-in dialog that pops up, select Certificates and click Add.
7. In the Add/Remove Snap-in dialog, click OK.
8. The Console Root in the Console window will now contain a Certificates item (see screenshot above). This Certificates item contains the certificate stores of your machine.
9. Save the Console as a Microsoft Management Console File (.msc file) via the File | Save command of the Console window. You can subsequently use this MSC file (via the File | Open command of a Console window) to access the certificate stores on your machine.

---

**Exporting a public-key certificate**

If you have a certificate with a private-public key, you might wish to export this certificate with only a public key. This public-key certificate can then be sent to receivers for use in verifying signatures created with the private key of the certificate.

A public-key certificate can be exported from an existing private-public-key certificate as follows:

1. Open the certificate stores in a Console window. Do this as follows: (i) Enter `mmc` on the Start menu's Run command line; (ii) In the Console window that pops up, select File | Open, and select the MSC file in which the certificate stores were saved (see section immediately above this section).
2. Browse for the certificate that you wish to export as a public-key certificate and right-click it.
3. In the context menu that pops up, select All Tasks | Export. This pops up the Certificate Export Wizard (screenshot below).
4. Select **Next**.
5. In the Export Private Key window, select *No, do not export the private key*, and click **Next**.
6. In the Export File Format window, select the required format (leave the default DER format unchanged if you are not sure), and click **Next**.
7. In the File to Export window, browse for the location where you wish to save the file and provide a name for the file (without a file extension, which will be automatically appended). Click **Next** when done.
8. Click **Finish** to complete the export.

A public-key certificate will be created at the location you specified. This public-key certificate can be sent to receivers of XML files signed with the corresponding private key. The receiver can then import this public-key certificate to a certificate store on his or her machine and use the public key of this certificate for verification.

---

**See also**
- [XML Signature Configuration](#)
- [XML Signature Settings](#)
- [Signing the Authentic XML Document](#)
- [XML Document Validity](#)
- [Working with Certificates](#)
12.12 ASPX Interface for Web Applications

If an HTML report of DB or XML data for the Internet is to be created with an SPS, then the usual procedure for creating the report with StyleVision would be as follows:

1. If the source data is in a DB, then, with the finished SPS active in StyleVision, generate an XML file from the DB. (If the source data is in an XML file, then this step is not required.)
2. Also from the SPS, generate the XSLT-for-HTML file.
3. Transform the XML file using the generated XSLT-for-HTML file.
4. Place the resulting HTML file on the server.

For a web application, the HTML file could become outdated if the source (DB or XML) data is modified. Updating the HTML file on the Web server with the new data would require: (i) for DB-based data, the re-generation of the XML file, (ii) transforming the new XML file using the XSLT-for-HTML, and (iii) placing the result HTML file on the server.

StyleVision provides a solution to quickly update HTML web pages. This is a feature for automatically generating an ASPX application. All the required ASPX application files (the .aspx file, XSLT file, and the code files) are generated by StyleVision. These files can then be placed on the server together with the source DB file or XML file and the XSLT-for-HTML file. Each time the .aspx file—which is the web interface file—is refreshed, the following happens: (i) for DB-based data, a new XML file is generated from the DB; for XML-based data, this step is not required; (ii) the XML file is transformed using the XSLT-for-HTML file that is on the server; and (iii) the output of the transformation is displayed in the web interface page. In this way, the web interface page will quickly display the latest and up-to-date DB or XML data.

Generating files for an ASPX solution

After creating the DB-based SPS or XML-based SPS, do the following to create an ASPX solution:

1. With the SPS active in StyleVision, generate the ASPX files by clicking the command, File | Web Design | Generate ASPX Web Application. The ASPX application files will be created in the folder location you specify. The folder in which you generate the ASPX application will contain the following files among others:
   - Readme.doc
   - SPSFilename.aspx
   - SPSFilename.xslt
   - SPSFilename.cs
2. Place the DB file or XML file on the server, in the same folder as the ASPX application. The .aspx file is the entry point of the application. Refreshing this file will cause the DB or XML data that is displayed in it to be updated.

Note: You will need to have Altova’s RaptorXML application installed in order for the XSLT transformation to run correctly. If you have problems with the transformation, see the ReadMe.doc file for details about setting up RaptorXML.
How it works
The folder in which you generate the ASPX application will contain the following files among others:

- Readme.doc
- SPSFilename.aspx
- SPSFilename.xslt
- SPSFilename.cs

SPSFilename.aspx is the URL of the output document. SPSFilename.aspx executes C# code stored in the file SPSFilename.cs. This C# code reads the XML content (from files or a database as required) and passes it to RaptorXML, together with the SPSFilename.xslt file. (RaptorXML contains Altova's XSLT transformation engine. It can be downloaded from the Altova website.) RaptorXML performs a transformation of the XML content, using the provided XSLT file. The result is an HTML document, which the web application then displays in the browser. When the XML content changes, for example because of changes made to the database, browsing to SPSFilename.aspx (or refreshing the page in the browser) will automatically fetch the most recent data from the database or XML file and render an updated document.

See also
- DBs and StyleVision
- Connecting to a DB
- DB Data Selection
- The DB Schema and DB XML files
Example: Localhost on Windows 7
The procedure outlined below sets up your local host to serve an ASPX application. For more information, see the file Readme.doc in the ASPX application folder. This folder and file are generated when you select the command File | Web Design | Generate ASPX Web Application with an SPS file active.

Install RaptorXML
Make sure that the latest version of RaptorXML is installed. RaptorXML contains Altova’s transformation engine. It will be used to transform the (DB-generated) XML file.

Activate Internet Information Services (Microsoft’s web server)
If Internet Information Services (IIS) is not activated, carry out the steps below to activate it. Step 5 shows how to test whether IIS has been activated.
1. Go to Control Panel | Programs and Features | Turn Windows features on or off.
2. Set the Internet Information Services checkbox. The tri-state checkbox will change to Partly checked.
3. Additionally, set the Internet Information Services | World Wide Web Services | Application Development Features | ASP.NET checkbox.
4. Click OK. When the process is complete, you will have a folder named C:/inetpub/wwwroot. This is the web server’s root folder.
5. As a test, go to localhost in a browser. This will display the IIS welcome screen.

In StyleVision, generate the ASPX application
Generate the ASPX application as follows:
1. It is recommended that the database and the SPS file be in the same folder.
2. After the SPS file has been completed, issue the command Files | Web Design | Generate ASPX Web Application.
3. In the dialog that opens, create a folder below C:/inetpub/wwwroot and select that folder, for example: C:/inetpub/wwwroot/Test1.
4. On confirming your folder selection, StyleVision will generate the following files in it: <FileName>.aspx, <FileName>_AltovaDataBaseExtractor.cs, Web.config, and a folder App_Code containing the various files.

Note: In order to save files to C:/inetpub/wwwroot you will need to run StyleVision as an administrator. Do this by restarting StyleVision. Right-click the StyleVision executable file or a shortcut to it and select Run as Administrator.
Make ASPX aware of the generated application

Carry out the following steps to make ASPX aware of the application you have generated with StyleVision:

1. Go to Control Panel | Administrative Tools | Internet Information Services (IIS) Manager.
2. In the Connections panel, expand the tree to display the folder (for example, Test1). The folder’s icon will be a standard yellow folder at this point.
3. In the folder’s context menu, issue the command Convert to Application, then click OK in the dialog. The folder’s icon will now be a globe.
4. In the Connections panel, expand the tree to display Application Pool and select this.
5. In the context menu for DefaultAppPool (available in the main pane when you select the Application Pools item in the Connections pane), select the command Advanced Settings.
6. For the Identity property, select Custom account and enter your Windows user name and password.
7. For the Enable 32-Bit Applications property, enter True. (This is so the database drivers have access). This step applies only to 64-bit versions of Windows 7.

Run the application

In the browser, go to localhost/Test1/<FileName>.aspx (assuming Test1 is the name of the folder in which the ASPX application has been saved). The transformed HTML will be displayed in the browser. Refreshing this ASPX page will cause the latest data from the database or XML file to be displayed.

Note: If the browser hangs at this point, make sure that the RaptorXML is correctly licensed.

See also
- DBs and StyleVision
- Connecting to a DB
- DB Data Selection
- The DB Schema and DB XML files
12.13 PXF File: Container for SPS and Related Files

An SPS design that uses XSLT 2.0 or 3.0 can be saved as a Portable XML Form (PX F) file. The PXF file format has been specially developed by Altova to package the SPS design with related files (such as the schema file, source XML file, image files used in the design, and XSLT files for transformation of the source XML to an output format). The benefit of the PXF file format is that all the files required for Authentic View editing and for the generation of output from Authentic View can be conveniently distributed in a single file.

This section describing the use of PXF files is organized in two parts:

- Creating a PXF file
- Editing a PXF File
- Deploying a PXF file

Notes
Note the following points about the PXF feature:

- It is only supported for SPSs designed with XSLT 2.0 or 3.0.
- XBRL sources are not supported. The dialog asking whether to save as SPS or PXF does not appear when an SPS file contains such a schema source. When editing a PXF file, it is not possible to add such a schema source.

See also
- Usage Overview
- Save As
Creating a PXF File

To create a PXF file that will contain an SPS design plus related files, open the SPS design in StyleVision and select the command File | Save As. This pops up the Save Design dialog (screenshot below).

The SPS format is the standard Altova format for StyleVision designs. In this section we are concerned with the PXF format and so we will not consider the SPS format here. Saving a file as an SPS is described in detail in the User Reference section.

Save as PXF

Selecting the PXF option causes the familiar Save As dialog of Windows systems to pop up. Saving works exactly as described for the Save Design command—with the additional step of selecting the files you wish to include in the PXF file. After you specify the PXF filename, the Configure PXF dialog (screenshot below) will appear, in which you can select/deselect the files you wish to embed.
In the Global Configuration pane of the Design-time Files tab, you can select/deselect the design-related source files to be embedded/omitted. You can additionally choose to embed XSLT files generated from the design. In the XSLT files pane, select the output formats for which you wish to generate and embed XSLT files. If an XSLT file is included in the PXF file and the PXF file is opened in the Authentic View of an Altova product, then the toolbar button to generate and view that output format is enabled in Authentic View (screenshot below).

In the Additional Files tab (screenshot below), you can specify any additional files you wish to include that are not design-time files. These could be, for example, image files referenced in the design by a URL generated with an XPath expression. In the screenshot below, the image file NewsItems.bmp located in the Images folder is selected for inclusion in the PXF file.
To include an additional file in the PXF file, click the **Add Document** button and then browse for the file you want. The Open dialog (in which you browse for the required file) opens the folder in which the SPS is located. Files from this folder or any descendant folder may be selected. After an additional file has been added to the PXF file, it and the folder structure leading to it are displayed. The screenshot above indicates that the additional file `NewsItems.bmp` is in a folder named `Images`, which is itself contained in the folder in which the SPS file is located.

If a file is selected from a folder located in any level above the folder containing the SPS file, an error is reported.

In the SPS design, any reference to an additional file must be made with a relative path and must use the folder structure shown in the **Additional Files** pane. For example, `NewsItems.bmp` in the screenshot above must be referenced with the relative path: `Images/NewsItems.bmp`.

**Note:** In order to save PXF files, the option *Embed Images for RTF and Word 2007* (File | Properties | Images) must be selected.
See also
- Usage Overview
- Save As
**Editing a PXF File**

A PXF file can be opened in StyleVision via the File | Open command and edited. These edits can be of two types:

- The configuration of the PXF file can be edited
- The content of individual component files such as the SPS and Authentic XML can be edited in StyleVision, while other component files (such as image and CSS files) can be edited in external applications. All component files must however be explicitly updated in StyleVision.

---

**Entry point for PXF editing**

The entry point for editing the PXF configuration and for updating the PXF file is the PXF item in the Design Overview sidebar (screenshot below).

---

**Configure embedded files**

Clicking the Configure Embedded Files link in Design Overview (see screenshot above) opens the Configure Portable XML Form (PXF) dialog. The configuration options are exactly the same as described in the section, Creating a PXF File.

---

**Updating embedded files**

Clicking the Update Embedded Files link in Design Overview (see screenshot above) opens the Portable XML Form (PXF) Update dialog (screenshot below).
First, select whether the source files should be retrieved relative to their current PXF locations or from their original locations. Then check the files you wish to update and click **Update**. A new PXF file will be created and will overwrite the existing PXF file. Therefore, before you update, it is highly recommended that you back up the original PXF file.

- **See also**
  - Usage Overview
  - Save As
Deploying a PXF File

After a PXF file has been created, it can be transported, downloaded, copied, and saved like any other data file. Since the PXF file can contain all the files required to edit an XML file in Authentic View and to generate output reports, it is the only file an Authentic user needs in order to get started and to generate output.

A PXF file can be opened in the Authentic View of Altova products. To give you an idea of how a PXF file may be used, here is a list of some usage scenarios in XMLSpy:

- The PXF file is opened via the File | Open command. The embedded XML file will be displayed in Authentic View using the embedded SPS, and can be edited in Authentic View. The File | Save command saves changes to the PXF (the embedded XML is modified).

- The PXF file contains no embedded XML file and is opened via the File | Open command. If no XML file is included, then a Template XML file based on the SPS design is opened in Authentic View. The File | Save command will save this XML file as an embedded file in the PXF file.

- In the Altova product, XMLSpy, an XML file can be associated with a PXF file so that the embedded SPS of the PXF file is used for Authentic View editing. The association is done via the menu command Authentic | Assign a StyleVision Stylesheet. When changes are saved, they will be saved to the XML file; the PXF file will be unchanged.

- If an XSLT stylesheet for one of the output formats has been embedded in the PXF file, then the Authentic View user will be able to generate output in that format. This is done with the appropriate output-generation toolbar button (screenshot below). In Authentic View, individual output-generation toolbar buttons will be enabled only if the PXF file was configured to contain the XSLT stylesheet for that output. For example, if the PXF file was configured to contain the XSLT stylesheets for HTML and PDF, then only the toolbar buttons for HTML and PDF output will be enabled while those for RTF and DocX (Word 2007+) output will be disabled.

Note: If a PXF file is located on a web server and will be used with the Authentic Browser Plugin, you must ensure that the server does not block the file. You can do this by adding (via the IIS administration panel, for example) the following MIME type for PXF (.pxf) file extensions: application/x-zip-compressed.

See also
- Usage Overview
- Save As
Chapter 13

SPS File and Databases
13 SPS File and Databases

When a DB is used as the basis of an SPS—that is, as the main schema of an SPS—the SPS can be used in the following ways:

- To edit the DB in Authentic View.
- To generate an XML Schema having a structure based on the DB (if the DB does not contain a schema; only XML DBs, such as IBM DB2 version 9 upwards, contain schemas).
- To generate an XML file with data from the DB (if the required DB data is not already in XML format).
- To design and generate XSLT stylesheets for HTML, RTF, PDF, and Word 2007+ output.
- To generate DB reports (based on the SPS design) in HTML, RTF, PDF, and Word 2007+ format. These reports can be previewed in StyleVision.

When a DB is the source of a subsidiary schema in an SPS, then data from the DB can be included in the design document, but the DB itself cannot be edited in Authentic View. It is the XML document or DB associated with the main schema that can be edited.

General procedure

This section describes the procedure for working with DBs in StyleVision. After an introductory sub-section, which provides an overview of how DBs work in StyleVision, the sub-sections of this section describe the various steps in the work procedure. Note that we distinguish between two broad types of DBs: non-XML DBs and XML DBs. The term DB is used in two senses: generically, it refers to all DBs; specifically, to non-XML DBs. XML DBs are always referred to as XML DBs. The distinction should be borne in mind because the method of selecting the DB data that provides the schema and XML data for the SPS is different for these two types of DB.

- Connecting to a DB: Describes how to connect to non-XML DBs, including IBM DB2 versions below 9.
- DB Data Selection: Describes how the schema and XML data for the SPS is selected from the DB's table structure, for non-XML and XML DBs separately.
- The DB Schema and DB XML file: When DB tables (from non-XML DBs) are loaded, StyleVision generates and works with temporary XML Schema and XML data files based, respectively, on the DB structure and data. For XML DBs, the schema and XML files are not generated by StyleVision but referenced directly from the DB or, in the case of schemas, from another file location.
- DB Filters: Filtering DB Data: DB data that is loaded into the temporary XML file can be filtered.
- SPS Design Features for DB: In the SPS, special DB functionality, such as DB controls and DB Queries, are available.
- Generating Output Files: A wide range of DB report-related files can be generated by StyleVision.
**Supported databases**

The following databases are supported. The available root object for each database is also listed. 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>Root Object</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle 9i, 10g, 11g</td>
<td>schema</td>
<td></td>
</tr>
<tr>
<td>MySQL 5.x, 5.5.28</td>
<td>database</td>
<td></td>
</tr>
<tr>
<td>PostgreSQL 8.0, 8.1, 8.2, 8.3, 9.0.10, 9.1.6, 9.2.1</td>
<td>database</td>
<td></td>
</tr>
<tr>
<td>Sybase ASE15</td>
<td>database</td>
<td></td>
</tr>
<tr>
<td>IBM DB2 8.x, 9.5, 9.7, 10.1</td>
<td>schema</td>
<td></td>
</tr>
<tr>
<td>IBM DB2 for i 6.1, 7.1</td>
<td>schema</td>
<td>Logical files are supported and shown as views.</td>
</tr>
<tr>
<td>IBM Informix 11.70</td>
<td>database</td>
<td></td>
</tr>
</tbody>
</table>

- See also
  - What Is an SPS?
13.1 DBs and StyleVision

In StyleVision, you can create DB-based SPSs. These stylesheets enable you to do two things:

- Edit DBs in Authentic View, and
- Generate reports from DBs.

After you have created the SPS, you can view reports in StyleVision and generate report files in HTML, RTF, PDF, and Word 2007+ format. You can also save the following DB-related XML files that StyleVision generates:

- XML Schema based on DB structure (not applicable for XML DBs, where a schema is already available)
- XML file having structure defined in the generated schema and content from the DB (not applicable for XML DBs, where the data is already available in XML format)
- SPS that you design, and which is based on the generated schema
- XSLT stylesheet for HTML output (based on design of SPS)
- XSLT stylesheet for RTF output (based on design of SPS)
- XSLT stylesheet for XSL-FO output (based on design of SPS)
- XSLT stylesheet for Word 2007+ output (based on design of SPS)
- HTML output
- RTF output
- PDF output
- Word 2007+ output

The saved XML file can then be processed with the required XSLT stylesheet/s. This provides more flexible report-generating capabilities.

Note: The XML Schema and XML files are generated from non-XML DBs by StyleVision, and you cannot alter their structure or content for use in Authentic View. This is because the structure of these files is related to the structure of the non-XML DB. Editing the DB and creating reports from the DB depend on the unique XML structure generated by StyleVision from the DB.
Broad mechanism for working with DB-based SPSs

Given below are the steps involved in creating and using DB-based SPSs. These steps cover the two uses of DB-based StyleVision Power Stylesheets: editing the DB and creating HTML, RTF, PDF, and Word 2007+ reports from the DB.

- **Connect to the database** with StyleVision. During the connection process you can specify what data tables in the DB should be filtered out from the XML Schema.
- When the connection is made, a temporary XML Schema is generated based on the structure of the DB and that schema is displayed in the Schema Window of StyleVision in tree form. In the case of XML DBs, a pre-existing schema (either in the DB or at a file location) is referenced.
- Temporary StyleVision-internal XML files are also created. One is non-editable (see diagram above) and is used for the previews and as the source of the generated XML data file. The other is an editable XML file, which is displayed in Authentic View (see screenshot above). When changes made to this file in Authentic View are saved (with the File | Save Authentic XML Data command), the modifications are written back to the DB. The non-editable XML file is updated if necessary each time an output view is newly accessed or when the XML data is saved.
- In StyleVision, you can define top-level filters to restrict the data imported into the non-editable XML File, i.e. for the output views and the reports.
- A DB Query is used within Authentic View to restrict the list of records displayed in Authentic View. It is used only during editing.
- If editing changes have been saved to the DB, then the next time an output view window is accessed, the non-editable XML file is updated with the modified contents of the DB and the refreshed file is displayed in the preview.
- A DB-based SPS is created in the same way as the standard schema-based SPS: by dragging-and-dropping nodes into the Design Window, inserting static stylesheet components, assigning display properties, etc. These mechanisms are described in this documentation.

**See also**
- Sources and SPS, for a broader perspective on DB sources.
- SPS and Authentic View
13.2 Connecting to a Database

When you select either the File | New | New from DB or File | New | New from XML Column in DB Table command, the Open Database dialog (screenshot below) pops up. (Note that the New from DB and New from XML Column in DB Table commands are also available in the Design Overview sidebar.)

The connection to the database is made using one of the options given below:

- Using a Connection Wizard that guides you through the connection process. Using
- Using an ADO connection.
- Using an ODBC connection.
- Using a JDBC Connection.
- Using a Global Resource.

Each of these options is explained in a separate sub-section of this section.

Note: When adding an ODBC Data Source for the IBM iSeries (formerly AS/400), a default flag is set which enables query timeouts. This setting must be disabled for StyleVision to correctly load the SPS. When adding an ODBC data source (User/System DSN) for iSeries Access ODBC driver, the "iSeries Access for Windows ODBC Setup" dialog box is opened. Select the Performance tab, click the Advanced button and uncheck the Allow Query Timeout check box option.

Note: If you are using the 64-bit version of StyleVision, ensure that you have access to the 64-bit database drivers needed for the specific database you are connecting to.
**Note:** Some database security settings might require that you run StyleVision as an administrator. If you get connection errors due to security settings, right-click the StyleVision application icon on the desktop and **Run as Administrator**.

---

**Selecting the XML data and schema for the SPS**

After the connection to the database has been made, the schema and XML data for the SPS must be generated. This is done on the basis of the database structure and data. There are two possible situations:

- In the case of non-XML databases, you select the table/s for which the SPS is being created. StyleVision automatically generates: (i) a schema based on the structure of the table/s, and (ii) temporary XML files based on this schema and containing the data in the selected table/s. How to select the tables is described in the section [DB Data Selection | Non-XML Databases](#).

- In the case of XML databases, you must do two things. First, select the XML cell of the DB in which the required XML data is stored. This XML data is loaded as the Working XML File of the SPS. Second, select the schema on which the SPS will be based. How to select the XML data and schema is explained in the section [DB Data Selection | XML Databases](#).

---

**See also**

- [DBs and StyleVision](#)
- [DB Data Selection](#)
- [The DB Schema and DB XML files](#)
- [Datatype Conversions: DB to XSD](#)
Connection Wizard

When the Connection Wizard button in the Open Database dialog is selected, the Connection Wizard (screenshot below) pops up. The Connection Wizard helps you to connect to the most commonly used types of databases. In this section, we go through the steps for connecting to:

- A Microsoft Access database (a non-XML DB). Also included with this description is information about connecting to other non-XML DBs.
- An Oracle database or IBM DB2 database using ODBC (IBM DB2 databases are currently the only XML DBs that are supported).

MS Access

Carry out the following steps to connect to an MS Access database. Select Microsoft Access (ADO) (screenshot below), and click Next.

In the Connect to MS Access pane that pops up, browse for the MS Access database, and click Connect. The connection to the data source is established. For information about selecting tables in the MS Access DB, see DB Data Selection.

Note: The following points about non-XML DBs should be noted:

- The following providers (drivers) are recommended: (i) For Microsoft SQL DBs: Microsoft OLE DB Provider for SQL Server; (ii) For Oracle, MySQL, Sybase, and IBM DB2 DBs (versions prior to version 9 are non-XML), Microsoft OLE DB Provider for
**ODBC Drivers.**

- If a password is required to access the DB, it must be saved in the connection string.
- All the connection information is stored in the SPS in the form of a connection string. It is therefore important that all clients that use this SPS must have the same driver installed so that they will correctly use the connection string.
- If an ADO connection is used, hierarchical relationships in Oracle and Sybase DBs are ignored and only flat schemas are generated from these DBs.
- For an ADO connection to MS SQL Server via the driver SQL Server Native Client 10.0 the following property values must be set in the All tab of the Data Link Properties dialog: (i) Set the property value of Integrated Security to a space character; (ii) Set the property value of Persist Security Info to `true`.

---

**A note on network shares and UNC paths**

Any folder on your computer's hard drive can be marked as a 'share'. When this has been done, any other computer on the network can access it and even write to it depending on how the share has been set up. A share can therefore be used on a remote machine exactly as if it were a local folder.

When working with an SPS based on a DB which is on a hard drive (such as an MS Access DB), the folder in which the DB is located must be marked as a share. A UNC path is then used in the connection string to point to the DB. This enables the SPS to connect correctly to the DB when used by clients (such as Authentic Browser) on other machines.

There are two parts to setting up the network share mechanism for a DB-based SPS.

- The folder in which the DB is located must be set up for sharing. (In Windows XP, the sharing settings are accessed by right-clicking the folder and selecting **Sharing and Security...**.) You must also enable Advanced File Sharing (**My Computer** | **Tools** | **Folder Options**, then uncheck Simple File Sharing (in the **View** tab)).
- Accessing the shared folder and the DB from the remote location. To do this in a DB-based SPS, you have to set up the connection string with a UNC path. The format of a UNC path is: `\servername\sharename\path\file.mdb`, where `servername` is the name of the server, `sharename` is the name of the share, `path` is the path to the DB, and `file.mdb` is the name of an MS Access DB in the shared folder or a descendant folder of the shared folder.

**Note:** Network shares and UNC paths are to be used for DBs, such as MS Access, which do not require any driver and are located on a server on a network.

---

**Oracle and IBM DB2: ODBC or JDBC**

If you select **Oracle (ODBC / JDBC)** or **IBM DB2 (ODBC / JDBC)**, a dialog pops up asking whether you wish to select JDBC or ODBC for the connection (**screenshot below**).
ODBC connections are faster and light on resources, while JDBC connections support the newer features of your database. We recommend that you use ODBC connections. If you find that some DB features are not supported when you use ODBC, then try JDBC.

The process for connecting to an Oracle database or IBM DB2 database via ODBC is described below. The process for connecting via JDBC is described in the section, JDBC Connections.

Connecting to Oracle or IBM DB2 via ODBC
In the first screen of the Connection Wizard (see first screenshot in this section), select Oracle (ODBC / JDBC) or IBM DB2 (ODBC / JDBC) and click Next. In the JDBC vs. ODBC dialog that appears (screenshot above), select ODBC and click Next.

The Open Database dialog for Oracle or IBM DB2 pops up (see screenshot below, which shows the dialog for IBM DB2). This dialog lists the DSNs of the available Oracle or IBM DB2 data sources. Select the DSN you want from the list. (You can also create a new DSN, which will then be added to the list of existing DSNs and be available for selection.)
On clicking **Next**, the Connect to Oracle Database or Connect to DB2 Database dialog pops up (the screenshot below shows the Connect to DB2 Database dialog). In this dialog you provide information for connecting to the DB, including your user ID and password.

Clicking **OK** will establish the connection with the database. Also see **ODBC Connections** for
more details. How to select the required DB information is described in the section, DB Data Selection.

See also
- DBs and StyleVision
- DB Data Selection
- The DB Schema and DB XML files
- ADO Connections
- ODBC Connections
- Datatype Conversions: DB to XSD
ADO Connections
The ADO Connections option enables you to connect to a DB via ADO. In this section, the connection via ADO is described for two cases:

- A Microsoft SQL Server database (as an example of non-XML DBs).
- An IBM DB2 database (an XML DB; currently, IBM DB2 databases are the only XML DBs that are supported).

Microsoft SQL Server
To connect via ADO to any DB other than an MS Access DB (whether non-XML or XML), you build a connection string to that DB. In the Open Database dialog (screenshot below), click the ADO Connections button.

Build the ADO connection string as follows:

1. In the ADO Connections screen, click Build. This pops up the Data Link Properties dialog.
2. Select the provider (driver) to connect to the DB. (For SQL DBs, we recommend Microsoft OLE DB Provider for SQL Server; for Oracle, MySQL, Sybase, and IBM DB2 DBs, Microsoft OLE DB Provider for ODBC Drivers.) Then click Next. This takes you to the Connection tab.
3. Enter the name of the server, server log-on information, and the DB name, all of which are user-specific. If, for the log-on information, you enter a required password, note that you must check "Allow saving password" to save the password in the connection string. Otherwise the connection will fail.

4. Click Test Connection to test whether a connection can be successfully made with the connection string you have built. If a password has been entered, it will be used for testing the connection. However, the password will be saved in the connection string only if the "Allow saving password" option has been checked (see Step 3).

5. If the connection test fails, rebuild the connection string correctly. If the connection test is successful, click OK to complete. The connection string you build is entered in the ADO Connections screen. A password will be entered in the connection string only if the "Allow saving password" option is checked.

6. Click Connect. The connection is made and a dialog appears in which you select the required DB tables. How to select the required DB tables is described in the section DB Data Selection.

Note: The following points about non-XML DBs should be noted:

- The following providers (drivers) are recommended: (i) For Microsoft SQL DBs: Microsoft OLE DB Provider for SQL Server; (ii) For Oracle, MySQL, Sybase, and IBM DB2 DBs (versions prior to version 9 are non-XML), Microsoft OLE DB Provider for ODBC Drivers.
- If a password is required to access the DB, it must be saved in the connection string.
- All the connection information is stored in the SPS in the form of a connection string.
is therefore important that all clients that use this SPS must have the same driver installed so that they will correctly use the connection string.

- If an ADO connection is used, hierarchical relationships in Oracle and Sybase DBs are ignored and only flat schemas are generated from these DBs.
- For an ADO connection to MS SQL Server via the driver SQL Server Native Client 10.0 the following property values must be set in the All tab of the Data Link Properties dialog: (i) Set the property value of Integrated Security to a space character; (ii) Set the property value of Persist Security Info to true.

---

**IBM DB2**

To connect to an IBM DB2 database via ADO, do the following:

1. In the Open Database dialog, select ADO Connections (*screenshot below*).

![ADO Connections](screenshot)

2. Click the **Build** button. The Data Link Properties dialog pops up.
3. In the Provider tab of the Data Link Properties dialog, select IBM OLE DB Provider for DB2 (*screenshot below*), and click **Next**.
4. In the Connection tab (screenshot below), enter the server name and the DB name.
5. Enter your user name and password.
6. Test the connection to the DB by clicking the **Test Connection** button. If the test fails, you will have to correct the connection data.
7. After the connection has been successfully tested, check the Allow Saving Password check box. This step is necessary to save the password information in the connection string. The Connection tab of the Data Link Properties should look something like the screenshot above when you are done.
8. Click **OK**. The connection string you have built in the Data Link Properties dialog is entered in the Open Database dialog.
9. Click **Connect** to make the connection to the database. The connection is made and a dialog appears in which you select the required DB information. How to do this is described in the section **DB Data Selection**.

---

**See also**
- DBs and StyleVision
- DB Data Selection
- The DB Schema and DB XML files
- Connection Wizard
- ODBC Connections
- Datatype Conversions: DB to XSD
ODBC Connections

This section describes how to connect to a DB via ODBC. The steps listed below describe a connection to an IBM DB2 database, but they apply also to other types of databases that can be connected to via ODBC.

To connect using an ODBC connection, do the following:

1. In the Connect to a Data Source dialog, select **ODBC Connections**.

   ![ODBC Connections](image)

2. Select one of the options for specifying a Data Source Name (DSN). If you select System DSN or User DSN, the available DSNs are displayed in the Data Source pane. If you select File DSN, you are prompted to browse for the DSN file. Alternatively, you can build a connection string to the DB by selecting the Build a Connection String option.

3. If you wish to add a DSN to those in the Data Source pane, click the Create a New DSN icon.

4. In the Create an ODBC DSN dialog that appears, select the required driver, then click the **User DSN** or **System DSN** button.

5. In the dialog that appears (**screenshot below**), select the DB alias and give it a DSN.
6. Click OK to finish. The DB is added as a Data Source to the list in the Data Source pane.
7. After you have selected the DataSource (via the System DSN or User DSN option), or selected a DSN File (File DSN option), or built a connection string (Connection String option), click Connect.
8. When you are prompted for your user ID and password, enter these and then click OK. The connection is made and a dialog appears in which you select the required DB information. How to do this is described in the section DB Data Selection.

Note: The listing in the data source pane (when the System DSN or User DSN option is selected) can be edited using the Edit Selected DSN, Remove Selected DSN, and Refresh Listed DSNs icons at the bottom of the ODBC Connections screen.

---

**Building a connection string**

In the ODBC Connections screen, you can build a connection string by selecting the Build a Connection String (see screenshot above) radio button and then clicking the Build button that appears in the dialog. The Select Data Source dialog that then pops up is shown in the screenshot below.
In the File Data Source tab, you can browse for a File DSN that you can use to build the connection string. Alternatively, in the Machine Data Source tab, the data sources that are available on your machine are listed, and one of them can be selected to build the connection string. You can add a new data source by clicking the **New** button. When you click **OK** in this dialog, the connection string is entered in the ODBC Connections screen.

**Note:** When an ODBC connection is used to connect to an MS Access DB, data in the DB cannot be edited.
JDBC Connections

This section describes how to connect to a DB via JDBC. The steps listed below describe a connection to an IBM DB2 database, but they apply also to other types of databases that can be connected to via JDBC.

Pre-connection steps

Before connecting to the DB via JDBC, you must do the following:

1. Ensure that the Java Runtime Environment (JRE) is installed. If it isn't installed, then install it. Use a 32-bit JRE for the 32-bit version of StyleVision or a 64-bit JRE for the 64-bit version of StyleVision.
2. Install a JDBC driver. No special installation setup is required. All you need to do is copy the driver to a local directory, for example, `c:\jdbc`.
3. Set up the `CLASSPATH` to include the location where the JDBC driver is located. (The application reads the `CLASSPATH` environment variable to locate the JDBC driver.) To access and edit the `CLASSPATH`, do the following: Click Start | Control Panel | System | Advanced | Environment Variables. In the Environment Variables dialog that appears, select either the `CLASSPATH` user environment variable or the `CLASSPATH` system environment variable and click the Edit button. Add the path to the JDBC driver to the `CLASSPATH`. For example: `CLASSPATH=C:\jdbc\sqljdbc.jar; C:\jdbc\db2jcc.jar;`.
4. Log off and then log on again to make the `CLASSPATH` changes active.

Note: Installing the complete IBM DB2 or Oracle client will automatically populate the `CLASSPATH` with the JDBC drivers of the respective packages. For more details, see the summaries below.

Connecting via JDBC

To connect using a JDBC connection, do the following:

1. In the Connect to a Data Source dialog, select JDBC Connections. The JDBC Connections pane (screenshot below) appears.
2. Select a JDBC driver from the Driver dropdown list (all detected drivers, i.e. those in the CLASSPATH environment variable, are listed). Enter a connection string in the Database URL text box and a user name and password as required. Given below is the connection string syntax for commonly used databases, each with an example connection string.

**Oracle**

jdbc:oracle:thin:[user/password]@[host][:port]/SID

jdbc:oracle:thin:@//abcd234/ORA11

**IBM DB2**

jdbc:db2://host_name:port/dbname

jdbc:db2://MyDB2:50000/boz

**MySQL**

jdbc:mysql://host_name:port/dbname

jdbc:mysql://MyDB2:3306/moz

**MSSQL**

jdbc:sqlserver://
host:port;databaseName=name;user=name;password=Pwd

jdbc:sqlserver://abcd38:1433;databaseName=coz

jdbc:sqlserver://Q5;DatabaseName=coz;SelectMethod=Cursor

**PostgreSQL**

jdbc:postgresql://host:port/database
jdbc:postgresql://abc993:5432/qanoz

**Sybase**
jdbc:sybase:Tds:host:port/dbname
jdbc:sybase:Tds:abc12:2048/QUE

3. Click **Connect** to make the connection. The connection is made and a dialog appears in which you select the required DB information. How to do this is described in the section **DB Data Selection**.

---

**Step-by-step: MSSQL, MySQL, PostGreSQL, and other non-XML DBs**

Given below is a step-by-step guide for connecting to a non-XML DB via JDBC:

1. JDBC JAR files (driver files): Copy the files to any local location. Then add the full path and filename to the Windows **CLASSPATH** variable. For example:
   
   ```
   C:\jdbc\sqljdbc.jar;
   C:\jdbc\db2jcc.jar;
   
   
   ```

2. Log off and log on to make the **CLASSPATH** changes active.

3. Start StyleVision and access the **Connect to a Data Source** dialog.

4. The JDBC Connections pane lists, in the **Driver** dropdown box, the detected JDBC drivers. If the dropdown box is empty, make sure that the file `altovadb.jar` is present in the folder: `C:\Program Files\Altova\Common2015\jar`.

5. Connect to a database as described in the section **Connecting via JDBC** above.

---

**Step-by-step: Oracle**

Given below is a step-by-step guide for connecting to an Oracle DB via JDBC. If you don't need the XML and XDB features of the Oracle DB, follow the instructions in the step-by-step guide for non-XML DBs. The installation folder of the Oracle client is indicated in the steps below by the placeholder: `%ORACLE_HOME%`.

1. Install Oracle client software with OCI and ODBC features enabled. If an Oracle client is already installed check if the two jar files below are present:

   ```
   %ORACLE_HOME%\LIB\xmlparserv2.jar
   %ORACLE_HOME%\RDBMS\jlib\xdb.jar
   
   ```

2. Add the following files to the Windows **CLASSPATH** environment variable:

   ```
   %ORACLE_HOME%\jdbc\lib\ojdbc6.jar
   %ORACLE_HOME%\LIB\xmlparserv2.jar
   %ORACLE_HOME%\RDBMS\jlib\xdb.jar
   
   ```

3. Log off and log on to make the **CLASSPATH** changes active.

4. Start StyleVision and access the **Connect to a Data Source** dialog.

5. The JDBC Connections pane lists, in the **Driver** dropdown box, the detected JDBC drivers. If the dropdown box is empty, make sure that the file `altovadb.jar` is present in the folder: `C:\Program Files\Altova\Common2015\jar`.

6. Connect to a database as described in the section **Connecting via JDBC** above.
**Step-by-step: IBM DB2**

Given below is a step-by-step guide for connecting to an IBM DB2 database via JDBC.

1. If an IBM DB2 client has already been installed, nothing needs to be done since the CLASSPATH will have been set by the installation process.
2. If no IBM DB2 client has been installed, add the IBM DB2 JDBC driver jar files `db2jcc.jar` and `db2jcc_license_cu.jar` to the Windows CLASSPATH. Log off and log on to make the CLASSPATH changes active.
3. Start StyleVision and access the Connect to a Data Source dialog.
4. The JDBC Connections pane lists, in the Driver dropdown box, the detected JDBC drivers. If the dropdown box is empty, make sure that the file `altovadb.jar` is present in the folder: `C:\Program Files\Altova\Common2015\jar`.
5. Connect to a database as described in the section *Connecting via JDBC* above.

**Note:** When databases are connected to via JDBC, due to insufficient information returned by the drivers: (i) data editing is not possible for tables without a primary key; (ii) Execute for data-editing in SQL Editor will not work.

- **See also**
  - DBs and StyleVision
  - DB Data Selection
  - The DB Schema and DB XML files
  - Connection Wizard
  - Datatype Conversions: DB to XSD
Global Resources

The Global Resources option enables you to connect to a database that has been defined as a global resource. The advantage of using a global resource is that you can quickly and easily change the database source for your SPS by changing the active configuration. Before you attempt to connect to a DB via a global resource, ensure that a global resource has been created for the database to which you wish to connect.

When you click the Global Resources icon in the Open Database dialog, 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 (check Tools | Active Configuration or the Global Resources toolbar) is used, and the connection is made. You must now select the data structures and data to be used as described in DB Data Selection.

See also
- DB Data Selection
- Defining Global Resources
- Tools | Global Resources
- Tools | Active Configuration
- Connection Wizard
13.3 DB Data Selection

Selecting the schema and XML data that will be used in the SPS involves selecting one or more tables, or a cell, or a specific schema, depending on whether the database (DB) being used is a non-XML DB (such as MS Access) or an XML DB (IBM DB2 version 9.0, etc). We refer to the selection of the schema and XML data as DB data selection, and it is carried out immediately after connecting to the DB.

How the DB data is selected depends upon the type of DB to which the connection is being made:

- In the case of non-XML databases, you select the table/s for which the SPS is being created. StyleVision automatically generates: (i) a schema based on the structure of the table/s, and (ii) temporary XML files based on this schema and containing the data in the selected table/s. How to select the tables is described in the section DB Data Selection | Non-XML Databases.
- In the case of XML databases, you must do two things. First, select the XML cell of the DB in which the required XML data is stored. This XML data is loaded as the Working XML File of the SPS. Second, select the schema on which the SPS will be based. How to select the XML data and schema is explained in the section DB Data Selection | XML Databases.

See also

- DBs and StyleVision
- The DB Schema and DB XML files
- Connecting to a Database
- Datatype Conversions: DB to XSD
Non-XML Databases

After a connection has been made to a non-XML database, the Insert Database Objects dialog appears. This dialog consists of two parts. In the upper Source pane, which contains a graphical representation of the tables in the DB, you select the tables required for the SPS. An XML Schema and XML data files will be generated by StyleVision on the basis of the tables selected. In the lower Preview pane of the Insert Database Objects dialog, you can preview the contents of the selected table.

The Source pane

In the Source pane, the DB tables are displayed graphically (see screenshot below). Select the tables required for the SPS by checking the respective check boxes.

The toolbar of the Source pane (screenshot below) contains three icons, respectively, from left to right: Filter Folder Contents, Checked Objects Only, and Object Locator. The Checked Objects Only icon toggles the display between all tables and checked tables.

Filtering folder contents

To filter objects in the Source pane, do the following:

1. Click the Filter Folder Contents icon in the toolbar of the Source pane. The Filter icon appears next to the Tables folder.
2. Click the Filter icon next to the Tables folder, and select the filtering option from the popup menu (screenshot below), for example, Starts with.
3. In the entry field that appears, enter the filter string (in the screenshot below, the filter string on the Tables folder is A). The filter is applied as you type.

The object locator
To find a specific database item by its name, you can use the Source pane’s Object Locator. This works as follows:

1. In the toolbar of the Source pane, click the Object Locator icon. A combo box appears at the bottom of the Source pane.
2. Enter the search string in the entry field of this list, for example Altova (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 Source pane.

Adding and editing SELECT statements for local views
You can create `SELECT` statements in SQL to create local views. When the schema is generated from a DB connection which has local views (or `SELECT` statements) defined for it, the schema that is generated for the DB will contain a table for each `SELECT` statement.

To create a `SELECT` statement, do the following:

1. Click the **Add/Edit SELECT Statement** tab. This pops up the Enter a SQL Select Statement dialog *(screenshot below)*.

2. Enter the `SELECT` statement. If you connect to an Oracle or IBM DB2 database using JDBC, then the `SELECT` statement must have no final semicolon. If you wish to create a `SELECT` statement for an entire table, right-click the table in the Insert Database Objects dialog and select the context menu command **Generate and Add a SELECT Statement**.

3. Click **Add Select Statement**. The `SELECT` statement is added to the list of `SELECT` statements in the Insert Database Objects dialog *(screenshot below)*.
**Note:** If you connect to an Oracle or IBM DB2 database using JDBC and use a `SELECT` statement with the Add/Remove Table command to retrieve data, then the `SELECT` statement must have no final semicolon.

When you click **Finish** in the Insert Database Objects dialog, a table is created for each `SELECT` statement (*screenshot below*).

---

**Local relations between tables**
You can create local relations between two tables, similar to the primary-key/foreign-key kind of relationship. The relation is local, in StyleVision, which means that the database itself does not
have to be modified. The local relationship will be represented in the generated schema.

To create a local relation, do the following:

1. In the Insert Database Objects dialog, click the **Add/Edit Relations** tab. This pops up the Add/Edit Table Relations dialog *(see screenshot below)*.
2. Click the **Add Relation** button, and in the Primary/Unique Key column, click the dropdown button of the Select Table combo box *(screenshot below)*. Select a table for the Foreign Key column also. The relationship that will be generated eventually will select rows in which the selected Primary/Unique Key column matches the selected Foreign Key column.

3. Select the Primary/Unique Key table column that must match the Foreign Key table column, and then select the Foreign Key column. Once again, use the combo boxes in the respective columns *(see screenshot below)*. Notice that if there is a type mismatch, an error sign will be displayed.
4. Add more local relations, if required, by repeating steps 2 and 3 above.
5. Click **OK** to complete the relation. When the schema is generated, it will reflect the newly created relations.

Look at the two screenshots above and see how, in the generated schema, **Altova** and **Department** each contain **Address**. Those **Department** rows with **PrimaryKey** values equal to the **ForeignKey** value of the **Address** row will be output. And those **Altova** rows with **PrimaryKey** values equal to the **zip** value of the **Address** row will be output.

**Note:** Local relations between SQL **SELECT** statements are not supported.
**Previews and the Preview pane**

To preview the structure and contents of a table, select the table in the Source pane and then click the **Preview** button or **Reload** button—according to which of the two is displayed—in the Preview pane (screenshot below). The contents of the table are displayed in a table format in the Preview pane (screenshot below).

---

**Generating the XML Schema and Working XML File from the DB**

After you have selected the tables for which you wish to use in the SPS, click **Finish** to generate and load the XML Schema. An XML Schema with a structure corresponding to that of the DB with the selected tables is displayed in the Schema Tree Window. A Working XML File having a structure corresponding to that defined in the generated schema and containing data from the selected tables is also generated and is used for the output previews.
Note that all the selected DB tables are created in the XML Schema as children of the DB document element and as items in the Global Elements list. For a complete description of the structure of the generated XML schema, see The DB Schema and DB XML files. Note that the XML Schema generated from the DB will not be altered by any DB Filter that may be built subsequently.

After you have connected to the DB and generated the XML Schema, you can use the full range of StyleVision features to design an SPS for the DB.

**See also**
- DBs and StyleVision
- The DB Schema and DB XML files
- Connecting to a Database
- Datatype Conversions: DB to XSD
XML Databases

After having made the connection to the XML database (currently only IBM DB2 XML databases are supported) via the Open Database dialog (via the File | New | New from XML Column in DB Table command; see previous sections), you will need to do two things:

- Select the cell in the DB that contains the required XML document. The XML document will be loaded automatically as the Working XML File.
- Select the XML Schema for the SPS.

Selecting the XML Cell and Working XML File

After making the connection to the IBM DB2 database, the Select XML Table dialog (screenshot below) appears.

1. In the Select XML Table dialog, select the table that contains the XML data you wish to create as the Working XML File. In the screenshot below, the table NHE_TEST has been selected.

2. Click Next. This pops up the Choose XML Field dialog (screenshot below). If you wish to filter the selection displayed in the pane, enter an SQL WHERE clause and click Update. Note that the WHERE clause should be just the condition (without the WHERE keyword, for example: NHE_TEST_TEXT= 'Two' )
3. Select the cell containing the XML data you wish to create as the Working XML File. In the screenshot above, the selected cell is highlighted in blue.

4. Click Next. This pops up the Choose XML Schema dialog, in which you select the schema to be used for the SPS. See next section.

Note: If you connect to an Oracle or IBM DB2 database using JDBC and use a SELECT statement with the Add/Remove Table command to retrieve data, then the SELECT statement must have no final semicolon.

---

**Selecting the XML Schema for the SPS**

The schema that will be used for the SPS can be either an XML Schema contained in the DB or a schema at a file location that can be accessed by StyleVision. To select the schema, do the following:

1. In the Choose XML Schema dialog (screenshot below), select the appropriate radio button according to whether you wish to select the schema from among those stored in the DB or from a file location. Note that if a non-DB schema is selected—that is, a schema from an external file—then no DB validation will be carried out.
2. Select the schema. Schemas stored in the DB are listed in the dropdown list of the Schemas from Database combo box, and can be selected from there. An external schema can be selected by browsing for it.

3. Click Finish to complete.

The schema tree
After completing the process to select the XML data and the schema, the selected XML data is created as the Working XML File and the schema is loaded into the SPS. Both are displayed in the Schema Tree window (screenshot below).

The SPS can now be built using the usual StyleVision mechanisms. Note that the data in the Working XML File can be edited in Authentic View and saved to the DB.

Note: The Working XML File should be valid against the schema selected for the SPS. Also ensure that the schema’s root element (document element) corresponds to the root element of the XML document.

See also
- DBs and StyleVision
- The DB Schema and DB XML files
• Connecting to a Database
• Datatype Conversions: DB to XSD
13.4 The DB Schema and DB XML files

The DB XML Schema
When you load a non-XML database (non-XML DB) into StyleVision, an XML Schema with a structure based on that of the DB is generated by StyleVision and displayed in the Schema Tree window. (In the case of XML DBs, an existing schema (either stored in the DB or at a file location) is specified as the schema to be used in the SPS.) This section on schemas, therefore, refers only to non-XML DBs.

The XML Schema is created with a document element called DB. The DB element contains child elements which correspond to the top-level tables in the DB. These top-level table elements are also created as entries in the Global Elements list in the Schema Tree Window. The top-level elements in the screenshot below are: Address, Altova, Department, Office, and Person; they correspond to tables in the DB.

Each top-level table element may have an unlimited number of rows (see screenshot below). Each row corresponds to a record in the DB. In the schema tree the rows are represented by a single Row element. Each Row element has attributes which correspond to the fields of the table. One of these attributes is generated by StyleVision for every row of every table: AltovaRowStatus, which holds the current status of the row: added, updated, and/or deleted. The remaining attributes are the fields of the respective DB table.
Note: The structure of the generated XML Schema is as outlined above. Whatever tables are selected during the connection step are included in the structure. The construction of a DB Filter does not affect the structure of the XML Schema.

New DB Schema Structure
The structure of the XML Schema generated from DBs starting with the 2005 version of StyleVision is different than the structure generated in previous versions of StyleVision. The new structure enables the editing of databases in the Authentic View of Altova products—a feature which was not available with earlier versions. As a result, any SPS generated with earlier versions of StyleVision will generate an error when opened in versions of StyleVision starting from the 2005 version. To be able to use the DB editing and reporting features of StyleVision, you should recreate the SPS in the current version of StyleVision.

DB XML data files
After a connection to the XML DB has been made, the XML schema and column with XML data...
selected, the Schema Tree window *(screenshot below)* will list the selected schema and the
column that will be used for the Working XML File.

![Schema Tree](image)

Two temporary XML files are generated from the DB (see DBs and StyleVision for an illustration):

- A temporary editable XML file, which can be edited in Authentic View
- A temporary non-editable XML File, which is used as the Working XML File (for previews and output generation)

The temporary editable XML file is generated when the DB is loaded into StyleVision. It can be edited in Authentic View after the SPS has been created. The display in Authentic View can be filtered by using the Query mechanism available in Authentic View. Any modification made in Authentic View to the editable data is written to this temporary XML File. Clicking File | Save Authentic XML Data saves the information in the temporary editable XML file to the DB.

The temporary non-editable XML file is generated when the DB is loaded into StyleVision. It is used as the Working XML File and for generating HTML, RTF, PDF, and Word 2007+ output. The editable XML file must be saved before changes made in Authentic View can be viewed in a preview.

**Note:**

- In the Authentic View of other Authentic View products only one temporary (editable) XML file is created when a DB-based SPS is opened. Modifications made in Authentic View are written to this file. When the file is saved, the information in the XML file is written to the DB.
- You can filter the data that goes into the non-editable temporary XML File for report-generation. (See Edit DB Filter for details.)
- You do not have to specifically assign a Working XML File in order to see HTML, RTF, PDF, and Word 2007+ previews. The automatically generated (non-editable) temporary XML file is used for this purpose.

---

**See also**

- DBs and StyleVision
- Connecting to a Database
- DB Data Selection
- Datatype Conversions: DB to XSD
13.5 DB Filters: Filtering DB Data

The data that is imported into the temporary non-editable XML file from the database (DB) can be filtered. (Note that the non-editable XML file is used for report generation, and the effect of a DB filter will therefore be seen only in the HTML, RTF, PDF, and Word 2007+ preview; not in Authentic Preview, which displays the temporary editable XML file, and not in Authentic View.) The DB filter (DB Filter) can be created either within the DB itself (if this is supported in your DB application), or it can be created within the SPS (SPS file). In the SPS, one DB Filter can be created for each top-level data table in the XML Schema (i.e. for the data tables that are the children of the DB element). Each time a DB Filter is created or modified, the data from the DB is re-loaded into the temporary non-editable XML file that is generated for the DB. In this way, DB Filters help you to keep the XML file down to an optimal size and to thus make processing for report generation more efficient.

Note: Using a DB Filter modifies the data that is imported into the temporary non-editable XML File. If you save an SPS with a DB Filter and then generate an XML File from the SPS, the generated XML File will be filtered according to the criteria in the DB Filter.

Creating a DB Filter

1. In the Design Document or Schema Tree, select the data table element for which you wish to create a DB Filter (either by clicking the start or end tag of the element, or by selecting the element in the schema tree).

2. Select Database | Edit DB Filters or click the icon in the toolbar. The following dialog is displayed:

   ![Edit Database Filters Dialog]

3. Click the Append AND or Append OR button. This appends an empty criterion for the filter (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 below.

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 Filters.

Expressions in criteria
Expressions in DB Filter criteria consist of a field name, an operator, and a value. The available field names are the columns 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 Filters
You can also enter the name of a parameter as the value of an expression. This causes the parameter to be called and its value to be used as the value of that expression. The parameter you enter here can be a parameter that has already been declared for the stylesheet, or it can be a parameter that you declare subsequently. Note, however, that when a parameter that has not been declared is typed into the value field, the OK button is disabled.

Parameters are useful if you wish to use a single value in multiple expressions, or if you wish to pass a value to a parameter from the command line (see StyleVision Server for details).

To enter the name of a parameter as the value of an expression, type $ into the value input field followed (without any intervening space) by the name of the parameter. If the parameter has already been declared (see Parameters), then the entry will be colored green. If the parameter has not been declared, the entry will be red, and you must declare it.

Declaring parameters from the Edit DB Filter dialog
To access the Edit Parameters dialog (in order to declare parameters), do the following:

1. Click the Parameters... button in the Edit DB Filters dialog. This pops up the Edit Parameters dialog shown below.

![View Parameters](image)

2. Type in the name and value of the parameter in the appropriate fields.

Alternatively, you can access the Edit Parameters dialog and declare or edit a DB Parameter by selecting Edit | Edit Stylesheet Parameters.
Note: The Edit Parameters dialog contains all the parameters that have been defined for the stylesheet. While it is an error to use an undeclared parameter in the SPS, it is not an error to declare a parameter and not use it.

After a DB Filter is created for a data table element, that element in the Schema Tree is displayed with the filter symbol, as shown for the `Addresses` element in the screenshot below.

---

Re-ordering criteria in DB Filters

The logical structure of the DB Filter 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 Filter.

The DB Filter 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 Filter by moving a criterion or set of criteria up or down relative to the other criteria in the DB Filter. 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. criteria 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 Filter, select the criterion and click Delete.

---

**Modifying a DB Filter**

To modify a DB Filter, select Database | Edit DB Filters. This pops up the Edit DB Filters dialog box. You can now edit the expressions in any of the listed criteria, add new criteria, re-order criteria, or delete criteria in the DB Filter. After you have completed the modifications, click OK. The data from the DB is automatically re-loaded into StyleVision so as to reflect the modifications to the DB Filter.

---

**Clearing (deleting) a DB Filter**

To clear (or delete) a DB Filter, select the element for which the DB Filter has to be cleared either in the Design Window or the Schema Tree. (There is one DB Filter for each (top-level) data table element.) Then click Database | Clear DB Filter. The filter will be cleared, and the filter symbol will no longer appear alongside the name of the element in the Schema Tree.

---

### See also

- DBs and StyleVision
- Connecting to a DB
- DB Data Selection
- The DB Schema and DB XML files
- Parameters
- Datatype Conversions: DB to XSD
13.6 SPS Design Features for DB

You can design a DB-based SPS just as you would design any other schema-based SPS, that is by dragging-and-dropping schema nodes from the schema window into the design document; by inserting static content directly in the design document; and by applying suitable formatting to the various design components. The following points, however, are specific to DB-based SPSs.

Creating a dynamic table for a DB table
To create a dynamic table for a DB table, do the following:

1. In the schema tree, select the top-level DB table to be created as a dynamic table, drag it into the design.
2. When you drop it, create is as contents and delete the contents placeholder. If the Auto-Add DB Controls feature is on, your design will look something like this:

3. In the schema tree, select the Row element of the DB table you wish to create as a dynamic table.
4. Drag it to a location inside the Addresses element.
5. When you release the element, select Create Table from the menu that pops up, and select the DB fields you wish to create as columns of the dynamic table. The DB table is created as a dynamic table.

Note: You can also create a DB table in any other format, such as (contents).

Auto-add DB Controls

The Authentic Auto-add DB Controls menu command or the toolbar icon toggles the auto-insertion of navigation controls for DB tables on and off. When the toggle is switched on, this toolbar icon has a black border; when the toggle is off, the toolbar icon has no border. If the Auto-insert toggle is on, DB Controls (see screenshot below) are automatically inserted when a DB table is dropped into the Design document. It is dropped, by default, immediately after the start tag of the table.

These controls enable the Authentic View user to navigate the records of the DB table in Authentic View. The first (leftmost) button navigates to the first record; the second button navigates to the previous record; the third button is the Goto Record button, which pops up the Goto Record dialog (screenshot below); it pops up a dialog that prompts you for the number of the record to which you wish to go; the fourth button navigates to the next record; and the fifth button navigates to the last record.
To manually insert the navigation controls in the Design document—which is useful if you wish to insert the controls at some other location than the default location—then do the following:

1. Turn the Auto-insert DB Controls toggle off.
2. Place the cursor at the location where you wish the navigation controls to appear (but within the DB table element's start and end tags).
3. Right-click, and from the popup menu, select Insert DB Control | Navigation or Insert DB Control | Navigation+Goto. Alternatively, these commands can be accessed via the Insert menu.

---

**Inserting a DB Query button for Authentic View**

The DB Query button enables the Authentic View user to submit a DB query. This helps the user to build conditions for the records to be displayed in Authentic View. Query buttons can be inserted for individual DB tables anywhere between:

- The start tag of the DB table element and the start tag of the DB table element's (child) Row element.
- The end tag of the DB table element and the end tag of that table element's (child) Row element.

To insert a DB Query button in your Design document, do the following:

1. Place the cursor at the allowed location (see above) where you wish the Query button to appear.
2. Right-click, and from the context menu (or Insert menu), select Insert | DB Control | Query Button. The Query Button is inserted at that point in the Design document.

When it is clicked in Authentic View, the Query button will pop up the Edit Database Query dialog. This dialog is described in the Authentic View documentation.

---

**Records displayed and records fetched**

You can control the number of records displayed in Authentic View and the number of records fetched when the file is loaded. To make these settings, do the following:

1. In Design View, select the Row element that corresponds to the records to be displayed/fetched.
2. In the Properties sidebar, select template in the Properties For column, and the Authentic group of properties.
3. For the property **DB Records Displayed**, select **all** from the dropdown list or enter the number of records to be displayed.

4. For the property **DB Records Fetched**, select **all** or **all displayed** from the dropdown list, or enter the number of records to be fetched. The value **all displayed** is the default. This property determines how many records are fetched when the DB data is loaded. If the value is less than that of the **DB Records Displayed** property, then the value of **DB Records Displayed** is used as the value of **DB Records Fetched**. The **DB Records Fetched** property enables you to reduce the number of records initially loaded thus speeding up the loading and display time. Additional records are loaded and displayed when the row (record) navigation buttons in Authentic View are used.

**Note:**

- The number of records displayed is applied to Authentic View.
- If a large number of tables is open in Authentic View, then the user will get an error message saying that too many tables are open. On the design side, you can reduce the number of tables that are open by reducing the number of records displayed (because a record can contain tables). On the user side, the user can use queries to reduce the number of tables loaded into Authentic View.

---

**AltovaRowStatus**

In the XML Schema that is created from a DB, each table has an **AltovaRowStatus** attribute. The value of this attribute is automatically determined by StyleVision and consists of three characters, which are initialized to **---**. If a row is modified, or a new row is added, the value is changed using the following characters.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>The row has been added (but not yet saved to the DB).</td>
</tr>
<tr>
<td>U,u</td>
<td>The row has been updated (but not yet saved to the DB).</td>
</tr>
<tr>
<td>D,d,X</td>
<td>The row has been deleted (but not yet saved to the DB).</td>
</tr>
</tbody>
</table>
These values can be used to provide users with information about rows being edited. The status information exists up to the time when the file is saved. After the file is saved, the status information is initialized (indicated by ---).

Formatting design document components
When records are added, modified, or deleted, StyleVision formats the added/modified/deleted records in a certain way to enable users to distinguish them from other records. Datatype errors are also flagged by being displayed in red. If you wish to maintain this differentiation, make sure that the formatting you assign to rows in a table do not have the same properties as those assigned by StyleVision. The formatting assigned by StyleVision is as follows:

<table>
<thead>
<tr>
<th>Added</th>
<th>A</th>
<th>Bold, underlined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modified (Updated)</td>
<td>U, u</td>
<td>Underlined</td>
</tr>
<tr>
<td>Deleted</td>
<td>D, d, X</td>
<td>Strikethrough</td>
</tr>
<tr>
<td>Datatype error</td>
<td></td>
<td>Red text</td>
</tr>
</tbody>
</table>

See also
- DBs and StyleVision
- Connecting to a DB
- DB Data Selection
- The DB Schema and DB XML files
13.7 Generating Output Files

After you have created a DB-based SPS, you can generate files related to it and save them. The following files can be generated and saved:

- The XML Schema based on the DB structure
- The XML file having the structure of the generated XML Schema and content from the DB
- The XSLT file for HTML output
- The HTML output file
- The XSLT file for RTF output
- The RTF output file
- The XSLT file for FO output (XSL-FO)
- The FO output file
- The PDF output file
- The XSLT file for Word 2007+ output
- The Word 2007+ output file

The files can be generated and saved from within the GUI or from the command line.

---

From within the StyleVision GUI

1. In the **File** menu, select the **Save Generated Files** item. This pops up the following submenu.

| Save Generated XSLT-HTML File... |
| Save Generated HTML File(s)... |
| Save Generated XSLT-RTF File... |
| Save Generated RTF File(s)... |
| Save Generated XSLT-FO File... |
| Save Generated FO File(s)... |
| Save Generated PDF File(s)... |
| Save Generated XSLT-Word 2007+ File... |
| Save Generated Word 2007+ File(s)... |
| Save Generated DB Schema... |
| Save Generated DB XML Data... |
| Save Generated User-Defined Schema... |
| Save Generated User-Defined XML Data... |

2. Select the file you wish to generate. This pops up the Save As dialog.
3. Browse for the desired folder, enter the desired filename, and click **OK**.
From the command line
From the command line, you can call StyleVision so that it generates and saves files associated with a DB-based SPS. You can save not only the XML Schema and XSLT files, but also an XML file with data from the DB, and HTML, RTF, PDF, and Word 2007+ output files based on the design in the SPS.

See also
- DBs and StyleVision
- Connecting to a DB
- DB Data Selection
- The DB Schema and DB XML files
13.8 Query Database

The **Query Database** command in the **Database** menu 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 **Database | Query Database** command or the Query Database toolbar icon.

**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**.

![Database Query Window](image)

Toggles the Browser 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 | Connect to a Data Source* window.
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.

**See also**
- DBs and StyleVision
- Connecting to a Database
- Datatype Conversions: DB to XSD
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 in the **Database** menu for the first time in a session (and when no database connection exists), the Connect to a Data Source 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. If connections already exist, you can select the required connection from among these.

How to connect to a database via the Connect to a Data Source dialog is described in the section **Connecting to a Data Source**.
Supported databases
The following databases are supported. The available root object for each database is also listed. 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>Root Object</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle 9i, 10g, 11g</td>
<td>schema</td>
<td></td>
</tr>
<tr>
<td>MySQL 5.x, 5.5.28</td>
<td>database</td>
<td></td>
</tr>
<tr>
<td>PostgreSQL 8.0, 8.1, 8.2, 8.3, 9.0.10, 9.1.6, 9.2.1</td>
<td>database</td>
<td></td>
</tr>
<tr>
<td>Sybase ASE15</td>
<td>database</td>
<td></td>
</tr>
<tr>
<td>IBM DB2 8.x, 9.5, 9.7, 10.1</td>
<td>schema</td>
<td></td>
</tr>
<tr>
<td>IBM DB2 for i 6.1, 7.1</td>
<td>schema</td>
<td>Logical files are supported and shown as views.</td>
</tr>
<tr>
<td>IBM Informix 11.70</td>
<td>database</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). After selecting the required data source in the left-hand combo box, you can select the required root object from the right-hand combo box.
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.

See also
- DBs and StyleVision
- Connecting to a Database
- Browser Pane: Viewing the DB Objects
- Query Pane: Description and Features
- Query Pane: Working with Queries
- Results and Messages
- Datatype Conversions: DB to XSD
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 (see screenshot below).

This section describes the following:

- The layouts available in the Browser pane.
- How to filter database objects.
- How to find database objects.
- How to refresh the root object of the active data source.

---

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 (screenshot below) and select the layout from the drop-down list. 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 pane.
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 pane.

**Refreshing the root object**

The root object of the active data source can be refreshed by clicking the Refresh button of the Browser pane’s toolbar.

---

**See also**

- DBs and StyleVision
- Connecting to a Database
- Query Pane: Description and Features
- Query Pane: Working with Queries
- Results and Messages
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.
- **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.
- **Options**
  - Open the Options dialog of SQL Editor.
- **Open SQL Script in DatabaseSpy**
  - Opens the SQL script in Altova's DatabaseSpy product.

SQL Editor 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. The buffer for the entry helper information can be filled on connection to the data source or the first time it is needed.

- **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 list box 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.

- **SQL Editor | Auto-completion**: The Auto-Completion feature works by suggesting, while you type, relevant entries from various SQL syntax categories. It is available for the following databases: MS SQL Server 2000, 2005, and 2008, MS Access 2003 and 2007, and IBM DB2 v.9. When the Auto-Completion option is switched on, the Auto-Completion window (screenshot below) appears, containing suggestions for auto-completion. Select the required entry to insert it. You can define whether the autocompletion popup should be triggered automatically after a delay which you can set in the Triggering Auto-completion pane, or if the popup has to be invoked manually. You can also select the keys to be used to insert the selected completion. The SQL Editor can intelligently suggest autocompletion entries based on language statistics. If this feature is activated, items that are frequently used appear on top of the list of suggested entries. In the Auto-completion window (screenshot below) itself, note the buttons at the bottom of the window. The Context-Sensitive Suggestion button sets whether only entries that are relevant to the context are displayed or all possible entries with that spelling. The Single
Mode button enables you to click a category button to select only that category. The Set All Categories button selects all categories. You can then deselect a category by clicking its button. The Clear All Categories button de-selects all categories. The other buttons are the various category buttons.

SELECT "CID", "INFO", "HISTORY" FROM "ALTOVA_USER".

- **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**.

**See also**
- DBs and StyleVision
- Connecting to a Database
- Browser Pane: Viewing the DB Objects
- Query Pane: Working with Queries
- Results and Messages
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.

- **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 the Execute 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. Messages about the execution are displayed in the Messages tab.

See also

- DBs and StyleVision
- Connecting to a Database
- Browser Pane: Viewing the DB Objects
- Query Pane: Description and Features
- Results and Messages
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).

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. To copy the header together with the cells, use the command *Copy Selected Cells with Header*.

The Results tab has the following toolbar icons:

- **Go to Statement** Highlights the statement in the SQL Editor that produced the current result.
- **Find** Finds text in the Results pane. XML document content is also searched.
Messages tab
The Messages tab provides information on the previously executed SQL statement and reports errors or warning messages.

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 let you step through the list, downwards and upwards, 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.

Note: These toolbar icon commands are also available as context menu commands.

See also
- DBs and StyleVision
- Connecting to a Database
- Browser Pane: Viewing the DB Objects
- Query Pane: Description and Features
- Query Pane: Working with Queries
Chapter 14
SPS File and XBRL
14 SPS File and XBRL

In StyleVision, users can create an SPS based on an XBRL taxonomy. The taxonomy is displayed in a tree structure derived from the relationships in the presentation linkbase of the taxonomy. Users can then visually design the report to be created in the same way like they would design any other SPS: by dragging taxonomy components from the schema tree into the design. The XBRL Table Wizard enables the quick creation of a report in table format. The structure and contents of the table can be customized, and the table can subsequently be edited in the design to add new table elements and specify formatting properties. Furthermore, if a taxonomy is extended from the US-GAAP taxonomy, tables can be automatically generated in line with best practices recommended for US-GAAP taxonomies.

This section is organized into the following sub-sections:

- **Creating an XBRL SPS File** describes how to start creating an XBRL SPS using the XBRL taxonomy as the starting point.
- **Taxonomy Structure** explains the structure of the schema tree, which is the source of the taxonomy components that will be used for the SPS design, and **Taxonomy Source Properties** documents the various taxonomy property settings.
- **Taxonomy Source Properties** describes the settings to define properties of the source taxonomy when used in an SPS.
- **XBRL Templates** describes the various XBRL templates that can be inserted in the design.
- **Open, Closed, and Classic Modes** describes these "modes", which determine whether the data is constrained, and the templates that can use these modes.
- **The XBRL Table Wizard** shows how to use the XBRL Table Wizard.
- **The XBRL Chart Wizard** explains how chart representations of data in the XBRL file can be created.
- **Properties and Formatting** explains how CSS styles are used for formatting XBRL tables and how you can use these styles.
- **Inline XBRL** shows how content from the XBRL file can be output to an (X)HTML file as an Inline XBRL element.
- **Altova XBRL Extension Functions** lists the XPath extension functions developed by Altova for use with XBRL in an SPS.

### Useful inks

The [XBRL International website](https://www.xbrl.org) is a good reference point for XBRL. The [XBRL recommendations](https://www.xbrl.org) can also be accessed via the XBRL International website.
14.1 Creating an XBRL SPS File

To create a new SPS file based on an XBRL taxonomy, do the following:

1. Click the menu File, hover over the New command and, in the submenu that rolls out (screenshot below), click the command New from XBRL Taxonomy.

2. In the dialog that appears, browse for and select the required XBRL taxonomy, then click Open.

3. You will be prompted to assign a Working XBRL File. If you wish to assign one, click Browse and select the XBRL instance file you wish to use as the Working XBRL File. The data from the Working XBRL File will be used for the previews. Alternatively, you can choose to not assign a Working XBRL File by clicking the Skip button. It is useful to assign a Working XBRL File since you will then be able to preview different outputs with actual XBRL data and test your design more thoroughly.

A new SPS based on the XBRL taxonomy will be created.

Main points about the XBRL SPS file

Note the following points about the newly created SPS file:

- Design View displays a root XBRL template with a contents placeholder (screenshot below).

- The Design Overview sidebar (screenshot below) lists the taxonomy file (.xsd file) and Working XBRL File (.xbrl file). This window can be used to add other SPS elements as described in the Design Overview section. To set properties of the source taxonomy, right-click the XBRL Properties item or click its Context Menu icon , and then select the command Edit XBRL Source Taxonomy Properties. The dialog that appears is described in the section, Taxonomy Source Properties.
The Schema Tree sidebar displays the XBRL taxonomy in a tree form. This tree structure is derived from the taxonomy's presentation linkbase (the links specifying presentation relationships between concepts; typically, the presentation linkbase is in a separate XML file).

The structure of the schema tree is described in the next section, Taxonomy Structure.
14.2 Taxonomy Structure

The structure of the taxonomy in the schema tree (see screenshot below) is derived from the taxonomy's presentation linkbase (the links specifying presentation relationships between concepts; typically, the presentation linkbase is in a separate XML file).

---

First level: presentation links

The taxonomy's tree structure organizes the taxonomy's concepts according to information in the presentation linkbase. In a presentation linkbase, a set of related concepts are grouped within a presentationLink element. The first level of the tree structure (screenshot below), therefore, represents the presentation links in the presentation linkbase. The screenshot below shows a taxonomy with six items representing the six presentation links in the taxonomy. Additionally on this first level, an item named <all> displays all the taxonomy concepts in a flat list. The <all> item contains all taxonomy concepts, including those that are not included in the presentation linkbase.

---

Second level: aspects

The second level within a single presentation link shows the link's aspects.

Aspects are: concepts, a period, an identifier, a unit, explicit dimensions, typed dimensions, non-dimensional segments, and non-dimensional scenarios (see screenshot below).
At the second level of the XBRL taxonomy, therefore, the following items appear:

- **Concepts**: A tree structure is created based on the presentation arcs within the presentation link. Each presentation arc is evaluated and the resultant relationships produced by the network of presentation arcs is represented in a tree (see screenshot below). The white icons in the tree are abstract items; these cannot occur in the instance file but are used in the presentation linkbase to group XBRL items.

- **Periods**: Indicates the period of the presentation link. Lists the various period types that a fact in an instance document can have (`xbrli:startDate`, `xbrli:endDate`, `xbrli:instant`, and `xbrli:forever`). In the schema tree the `xbrli:period` element and its allowed children (the four period types listed above) are listed and can be inserted individually in the design.

- **Identifiers**: Shows the `scheme` attribute of the `xbrli:identifier` element. The `xbrli:identifier` element and its `scheme` attribute can be inserted in the design.
• **Units**: Specifies the unit of the presentation link.
• **Segments and scenarios**: These aspects contain explicit or typed dimensional information. They are indicated with yellow icons.
• **Non-dimensional (non-XDT) segments and scenarios**: These aspects constrain those child elements, respectively, of the segment and scenario elements that do not represent explicit or typed dimensional information. They are indicated with gray icons.
• **Dimensions**: If a concept in a presentation link has a dimension (dimensions are defined in the definitions linkbase), then that dimension is listed for that presentation link (screenshot below).
  o **Dimensions** are indicated in the tree with a three-dimensional axis (and in the case of US-GAAP taxonomies with an [Axis] label).
  o **Hypercubes** are indicated with a cube icon (and, for US-GAAP, with the [Table] label).
  o **Domains** are shown with an EX or TY icon indicating whether the dimension is explicit or typed. In US-GAAP taxonomies, the [Domain] label is also shown.
  o **Domain members** are indicated in US-GAAP taxonomies with the [Member] label.

The `<all>` item in the schema tree
The `<all>` item in the schema tree contains a flat list of all concepts in the taxonomy, irrespective of whether they are included in the presentation linkbase or not. This enables access to concepts that are not present in the presentation linkbase. Such scenarios are possible, for example, when a taxonomy is extended to include a small number of new concepts that are not added to the presentation linkbase, or when no presentation linkbase is associated with the taxonomy. The `<all>` item additionally lists the period and identifier items, and optionally any dimensions that have been defined.
14.3 Taxonomy Source Properties

The XBRL Taxonomy Source Properties dialog (screenshot below) enables XBRL Label options and Inline XBRL options to be set. The dialog has two panes: (i) XBRL Labels, and (ii) Inline XBRL.

**XBRL Labels**

Items in a taxonomy can have multiple labels, but only one label of each set can be used in the design. Which one is used is specified in the XBRL Taxonomy Source Properties dialog (accessed via the Properties item in the Design Overview window). Using a combination of the options in this dialog (screenshot below), the desired label properties can be selected for the document as a whole.

The XBRL Taxonomy Source Properties dialog enables you to select the required label properties using the following options:
Each of the three combo boxes (Language; Label Link Role; Label Role) lists all options in that category that are used in the taxonomy. For example, the combo box for the Language option lists every language that has been specified in the taxonomy. Select the language in which you require labels to appear. In the same way, select the desired label link role and label role.

On presentation arcs, there is an optional attribute that specifies the preferred label role. If this is available on a presentation arc, it will be used if the Preferred Label Role check box is checked (the default setting).

The standard label role is that named http://www.xbrl.org/2003/role/label and will be used if: (i) the label role specified in the dialog cannot be found, and (ii) the Standard Label Role option is checked. The Standard Label Role option is checked by default.

Overriding the document's label properties
The label properties for the document (made in the XBRL Taxonomy Source Properties dialog as described above) can be overridden for specific taxonomy items by: (i) selecting the relevant item in the design, and (ii) setting properties for that item's label in the Properties sidebar (screenshot below).

Inline XBRL
XBRL items (facts) from the source XBRL instance document may or may not be displayed in the SPS design. The Inline XBRL options in the XBRL Taxonomy Source Properties dialog enables you to select whether:

1. Even source XBRL items that are not included in the SPS design are generated in the (X)HTML output, or
2. Only the XBRL items that are included in the SPS design are generated in the (X)HTML output. Items that are not displayed in the design will be output as hidden data inside the <HEAD> element of the (X)HTML document. (Note that fractions in hidden data do not include numerators and denominators.)
To specify what source XBRL items should be included in the output, select the appropriate radio button (All XBRL items in the XBRL instance or Only XBRL items in the design document).
14.4 XBRL Templates

The XBRL SPS design consists of one $XBRL root template, which contains a hierarchy of templates. This is similar to SPS designs based on XML Schemas. One major difference, however, is that XBRL SPS designs can use XBRL-specific templates in addition to the schema node templates that are used in schema-based designs. These "XBRL templates" provide an abstraction layer that enables the SPS designer to think in XBRL terms as opposed to the XML structures used to express the XBRL data model.

XBRL templates for the different kinds of XBRL components can be placed in the design either by dragging a component from the Schema Tree or via the Insert XBRL Element command (of the Insert menu or Design View context menu). XBRL templates are useful because they provide a quick way to create the main XBRL components at desired locations in the design. XBRL templates can be edited just as any other template. For example, conditions or Design Fragments can be inserted within the template.

This section is organized as follows:

- **Types of Templates**: XBRL templates are grouped into different types, each of which has unique properties. Knowing these properties is important for working with the templates.
- A listing and description of the templates in the submenu of the Insert XBRL Element command.

---

**Template location**

When inserting an XBRL template, make sure that you understand the hierarchical context within which the node is being inserted.
Types of Template

XBRL templates are of the following types:

<table>
<thead>
<tr>
<th>Template Type</th>
<th>Color</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presentation link template</td>
<td>Blue, 'P'</td>
<td>![Presentation link template]</td>
</tr>
<tr>
<td>Concept template</td>
<td>Green, 'C'</td>
<td>![Concept template]</td>
</tr>
<tr>
<td>XBRL object template</td>
<td>Green</td>
<td>![XBRL object template]</td>
</tr>
<tr>
<td>Item template</td>
<td>Brown, 'i'</td>
<td>![Item template]</td>
</tr>
<tr>
<td>Aspect value template</td>
<td>Brown, italic</td>
<td>![Aspect value template]</td>
</tr>
<tr>
<td>XML node template</td>
<td>Brown, 'XML'</td>
<td>![XML node template]</td>
</tr>
</tbody>
</table>

Schema node templates are brown and have a normal (non-italic) typeface.

XBRL SPS designs are created using a combination of XBRL templates (described on this page and the rest of this section, XBRL tables, XBRL charts, and other SPS features (such as Auto-Calculations and Conditions)).

Presentation link templates

Presentation link templates are created when you drag a presentation link from the Schema Tree into the design and create it as an XBRL template. The name of the presentation link appears in the template tags. The screenshot below shows a presentation link template that has been created for the Minitax Statement presentation link.

![Presentation link template screenshot]

The presentation link template selects the presentation link as its context. Template contents can then be defined within this context. It is mainly used for schema tree synchronization when selecting a concept template.

Concept templates

Concept templates are created when you drag a concept from the Schema Tree into the design and create the concept as an XBRL template. The concept is created as a closed mode Concept template, and the concept's name appears in the template tags. You can switch to open mode by selecting Template Behavior | Open Mode in the template's context menu.
A Concept template is created empty. You need to place an Item template within it to return facts related to the selected concept. The aspect value template, **Concept QName**, returns the concept's QName.

---

**XBRL object templates**

These are templates of certain XBRL elements (Periods, Identifiers, Units, and non-dimensional segments and scenarios) that are created as **XBRL objects**. If the template is specified to be in **open mode**, the XBRL object returns all the elements of the selected type (for example, all Periods; *the screenshot below shows an open-mode XBRL object template*). If the template is specified to be in **closed mode**, the XBRL object returns a specific element of the selected type (for example, one Period from among all Periods). To switch the behavior between open and closed modes, select the template, and, in the template's context menu (accessed by right-clicking the template), hover over the **Template Behavior** command and select the mode you want. See the section **Open, Closed, and Classic Modes** for details.

---

To insert an XBRL object template, right-click in the location where you wish to insert the template. Then hover over the **Insert XBRL Element** menu item. From the submenu that appears, select the XBRL element you wish to add as the XBRL object template.

The important point to note is that these templates each reference a specific XBRL object—and not an XML node. This means that, being an object rather than an XML node, there will be no context node for XPath expressions created within the object. Mechanisms that require a context node, such as the contents placeholder, therefore will not be resolved. To access the XML node corresponding to the XBRL object, insert the XML Node template (see below) within the XBRL object. Alternatively, **Altova XBRL extension functions** can be used in XPath expressions to access the required nodes.

To display information within an XBRL object template, use one of the following design mechanisms:

- An aspect value template (see below) that accesses a specific aspect of the XBRL component that was created as an XBRL object.
- An XML Node template (see below) that references the containing XBRL object as an XML node and allows XPath expressions to be used in this node's context.
**Item templates**
An Item template iterates over all XBRL facts that satisfy the constraints defined by the enclosing XBRL templates. See the *Item template* section for details.

---

**Aspect value templates**
An aspect value template references a specification-defined value of the XBRL aspect represented by the containing XBRL object. To add an aspect value template, right-click within an XBRL object template (see above), hover over the **Insert XBRL Element** menu item, and, from the submenu that appears, select an aspect of the current XBRL component.

The screenshot above shows an aspect value template for the Instant value of the Period aspect. It is created within the XBRL Period template.

---

**XML Node templates**
An XML Node template references its nearest ancestor XBRL object as an XML node. This XML node will be the context node for XPath expressions created within the XML Node template. To add an XML Node template, right-click in the location where you wish to insert the template. Then hover over the **Insert XBRL Element** menu item. From the submenu that appears, select XML Node.

The screenshot above shows an XML Node template that has been created within the XBRL object template of the Period component.

**Note:** Not all XBRL templates can contain an XML Node template. For example, a Concept template cannot, since there is no XML node for a concept in the instance.
Context

The Context template returns XBRL context elements in the instance file (see listing below).

```xml
<xbrli:context id="FY06">
  <xbrli:entity>
    <xbrli:identifier scheme="http://www.altova.com/XBRL/DocEx1/Ex1.htm">00058</xbrli:identifier>
  </xbrli:entity>
  <xbrli:period>
    <xbrli:startDate>2006-01-01</xbrli:startDate>
    <xbrli:endDate>2006-12-31</xbrli:endDate>
  </xbrli:period>
</xbrli:context>
```

When the Context template is inserted in the design, it is created as an empty template. You must therefore insert a contents placeholder in the design if you wish to output the contents of the element.

Note the following points:

- The location where the context template is inserted is significant. So you must ensure that you understand the hierarchical context within which the XBRL context node is inserted.
- The template returns all context elements at the hierarchical level at which the template is inserted.
- Within a Context template a particular context element in the instance file can be selected by creating a condition around the contents placeholder. This could be done, for instance by specifying the value of the id attribute of a context element (see screenshot above).
Concept

Concept templates are created when you drag a concept from the Schema Tree into the design and create the concept as an XBRL template. The concept is created as a closed mode Concept template, and the concept's name appears in the template tags. You can switch to open mode by selecting Template Behavior | Open Mode in the template's context menu.

A Concept template is created empty. You need to place an Item template within it to return facts related to the selected concept. The aspect value template, Concept QName, returns the concept's QName.
**Item**

The Item template iterates over all XBRL facts that satisfy the constraints defined by the enclosing XBRL templates. For example:

- If an Item template is placed within a Period, then it returns those facts that match the constraints which are in effect at that point.
- If an Item template is placed directly within the $XBRL template, then all facts in the instance document are returned (see screenshot below).
- If it is placed within an XBRL object template (open or closed mode) then the facts addressed by the XBRL object template are returned. In the case of an open-mode XBRL object, all facts defined within the restrictions of the XBRL object are returned. In the case of a closed-mode XBRL object, the fact/s defined by the properties of the XBRL object are returned.

When the Item template is inserted in the design, it is created as an empty template. You must therefore insert a contents placeholder in the design if you wish to output the contents of the fact. The Item template has a property to allow additional dimensions (in the Properties entry helper), which takes a value of yes or no.
Concept QName

The Concept QName template is an aspect value template. It can be added within a Concept template (see screenshot below), and returns the QName of the XBRL fact/s addressed by the Concept template. The command to insert the Concept QName template (Insert XBRL Element | Concept QName) is enabled only when the cursor is placed within a Concept template.

The concept’s QName is:

[Code snippet showing the usage of the Concept QName template]
Periods

Period aspect
The Period template iterates over:

- (in open mode) the periods present in the XBRL instance;
- (in closed mode) a single given period, regardless of whether that period occurs in the XBRL instance.

The Period template is indicated with the following tags:

The period element can occur within an xbrli:context element, in a structure as shown below:

```
<xbra:context id="FY06">
  <xbra:entity>
  </xbra:entity>
  <xbra:period>
    <xbra:startDate>2006-01-01</xbra:startDate>
    <xbra:endDate>2006-12-31</xbra:endDate>
  </xbra:period>
</xbra:context>
```

When you insert the Period template in the design, it is created as an empty template. You can then do the following:

- **Access values of the period (for example, instant, start-date, end-date):** Insert the Period aspect's aspect value templates (see below) to return the respective values of these aspects.
- **Select Facts:** An open or closed mode Period template can be used to select facts that match the properties of the Period.

```
Closed-mode Period template that returns the period's Instant:


Selecting facts that match the Period properties of the closed-mode Period template:


Period values
Templates for the following period values can be inserted via the corresponding **Insert XBRL Element** command:
<table>
<thead>
<tr>
<th>XBRL Element</th>
<th>Datatype</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>instant</td>
<td>xs:dateTime</td>
<td>Period Instant DateTime</td>
</tr>
<tr>
<td>startDate</td>
<td>xs:dateTime</td>
<td>Period Start DateTime</td>
</tr>
<tr>
<td>endDate</td>
<td>xs:dateTime</td>
<td>Period End DateTime</td>
</tr>
</tbody>
</table>

The period aspect elements occur within an `xbrli:period` element. They are typically entered within a Period template. Each aspect value template is inserted with a `contents` placeholder; it returns the text content of the corresponding period aspect element.
Identifiers

Identifier aspect
The Identifier template iterates over:

- (in open mode) the XBRL identifier elements present in the XBRL instance. The identifier element occurs within an xbrli:context element;
- (in closed mode) a single given identifier, regardless of whether that identifier occurs in the XBRL instance.

The Identifier template is indicated with the following tags:

When you insert an Identifier template in the design, it is created as an empty template. You can then do the following:

- **Iterate over the Identifier's aspects (value and scheme), and return their values**: An open or closed mode Identifier template will iterate over the Identifier aspects (value, scheme) that are specified as contents of the Identifier template, and return the values of these aspects.
- **Select Facts**: An open or closed mode Identifier template can be used to select facts that match the properties of the selected Identifier.

### Values of "Identifier 1"

<table>
<thead>
<tr>
<th>Identifier 1 (<a href="http://www.altova.com/minitax">http://www.altova.com/minitax</a>)</th>
<th>Identifier value</th>
<th>Identifier value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The identifier's text value is: Identifier value</td>
<td>(content) Identifier value</td>
<td></td>
</tr>
<tr>
<td>The identifier's scheme is: Identifier scheme</td>
<td>(content) Identifier scheme</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Identifier 1 (<a href="http://www.altova.com/minitax">http://www.altova.com/minitax</a>)</th>
</tr>
</thead>
</table>

### Facts reported for identifier "Minitax 1"

<table>
<thead>
<tr>
<th>Minitax 1</th>
<th>Minitax:Concept_06_1</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>(content)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Identifier values**

Aspect value templates for the following Identifier aspects can be inserted via the corresponding Insert XBRL Element commands:

<table>
<thead>
<tr>
<th>XBRL Node</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>@scheme</td>
<td>Identifier scheme</td>
</tr>
<tr>
<td>text()</td>
<td>Identifier text value</td>
</tr>
</tbody>
</table>
Units and Unitrefs

Unit aspect
The Unit template iterates over:

- (in open mode) the units present in the XBRL instance;
- (in closed mode) a single given unit.

The Unit template is indicated with the following tags. See the section Unit Properties for a detailed description of the different types of units.

The unit element can have structures as shown below:

```xml
<xbrli:unit id="IdUnitShares">
   <xbrli:measure>xbrli:shares</xbrli:measure>
</xbrli:unit>

<xbrli:unit id="IdUnitSquareMeters">
   <xbrli:measure>minitax:meter</xbrli:measure>
   <xbrli:measure>minitax:meter</xbrli:measure>
</xbrli:unit>

<xbrli:unit id="IdUnitDollarsPerShare">
   <xbrli:divide>
      <xbrli:unitNumerator>
      </xbrli:unitNumerator>
      <xbrli:unitDenominator>
         <xbrli:measure>xbrli:shares</xbrli:measure>
      </xbrli:unitDenominator>
   </xbrli:divide>
</xbrli:unit>
```

When you insert a Unit template in the design, it is created as an empty template. You can then do the following (see screenshot below):

- **Access values of the unit**: Insert the Unit aspect's aspect value templates (see below) to return the respective values of these aspects.
- **Select Facts**: Select facts in the XBRL instance that contain the defined units. The Item aspect value template within each Unit template, will then return the facts that each closed-mode Unit template selects.
Unitref templates
The Unitref template iterates over all units in the instance, regardless of whether these are referenced by a reported fact or not. Unitref templates are inserted empty and with their tags collapsed. You can expand a collapsed Unit tag (by double-clicking it), and you can insert content within the tags.

Unit values (aspect value templates)
Aspect value templates for the following Unit aspects can be inserted via the Insert XBRL Element commands, Unit Numerator and Unit Denominator. Units can be thought of as ratios consisting of a numerator and denominator. Use Unit Numerator to generate the numerator part of a unit, and Unit Denominator to generate the denominator part. See the screenshot below.
Dimensions

Dimension aspect
Templates of explicit/typed dimensions are created by dragging the element corresponding to the dimension from the schema tree (see screenshot below) into the design and creating the element as a Dimension template. The Dimension template iterates over:

- (in open mode) the XBRL dimension elements present in the XBRL instance;
- (in closed mode) a single given dimension.

When you insert an explicit or typed dimension template in the design, it is created as an empty template. You can then do the following:

- Iterate over the domain (of explicit dimensions) or typed element (of typed dimensions), and return their respective aspect values: An open or closed mode dimension template will iterate over the dimension. Aspect value templates within the dimension template (such as Dimension QName, Explicit Value QName, Typed Element) return the respective aspect values. The first template in the screenshot below shows an explicit dimension template containing aspect value templates.
- Select Facts: An open or closed mode dimension template can be used to select concepts that match the properties of the selected dimension. The second template in the screenshot below shows an explicit dimension template that selects concepts.

Dimension values
Aspect value templates for the following Dimension aspects can be inserted via the corresponding Insert XBRL Element commands:

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Aspect Value Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit</td>
<td>Dimension QName</td>
</tr>
<tr>
<td>Explicit</td>
<td>Explicit Value QName</td>
</tr>
<tr>
<td>Typed</td>
<td>Dimension QName</td>
</tr>
<tr>
<td>Typed</td>
<td>Typed Dimension Element</td>
</tr>
</tbody>
</table>
Non-XDT Segments and Scenarios

Templates of non-XDT segments and scenarios are created via the Insert XBRL Element command of the Design View context menu. The template is created as empty template in open mode. You can switch to closed mode via the Template Behavior | Closed Mode command of the template's context menu.

The non-XDT component template (segment or scenario) can then be used as follows:

- The contents of the non-XDT component template can be returned by using either the aspect value template Non-dimensional elements (first template in the screenshot below) or the XML Node template.
- The third template in the screenshot below selects all facts that contain the non-dimensional data defined in the Non-XDT template.

Using the AspectValue template:

```xml
<xs:element name="Non-dimensional elements" type="xs:string">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Non-dimensional elements" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Using the XMLNode template:

```xml
<xs:element name="XX" type="xs:string">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="YY" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Fact Selection

```xml
<xs:element name="F" type="xs:boolean">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="item" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```
Footnote

There are two parts to footnotes: (i) a footnote reference, and (ii) the actual footnote. The XBRL Footnote mechanism uses the TOC bookmarking and referencing system to create the footnotes. The Footnote template matches the XBRL footnote element and is created as an empty template. You can then design contents for the footnote and the footnote number. If the tag is collapsed, you can expand it by double-clicking the collapsed tag. The Footnote template is indicated with the following tags:

Note: The XBRL Table Wizard automatically creates footnotes if the Generate Footnotes option is checked.

Adding footnotes

A footnote can also be linked to a fact with a footnote arc. In such cases, the footnote can be inserted in the concept template that returns the relevant fact/s (see screenshot below). Use the Insert XBRL Element | Footnote command.

Sorting footnotes

If there are multiple footnotes that are linked to a single fact via footnote arcs, then the footnotes can be sorted. For example, sorting can be specified to be in ascending order of the character at any position in the footnote text. After adding the Footnote template (with the Insert XBRL Element | Footnote command), right-click the Footnote template, select the command Sort By and, in the Sort dialog that appears, enter the sorting criteria and click OK.

The order of the sorting (ascending/descending) is indicated by an icon in the template tag (see screenshot below).
Grouping footnotes

Multiple footnotes can be grouped according to a grouping key. For example, groups can be created on the basis of a grouping key that is the first three characters of the footnote text. To define the grouping key, right-click the Footnote template, select the command Group By and, in the Grouping dialog that appears, select the grouping criterion and key, and click OK.

That grouping has been specified is indicated by a G icon in the template tag (see screenshot above). To display the values of the footnotes by group, enter a user-defined template for the current group (by entering the XPath expression current-group() as the value of the user-defined template); see screenshot above.

Filtering footnotes

Multiple footnotes can be filtered by using an XPath expression. To do this, right-click the Footnote template, select the command Edit XPath Filter and, in the Edit XPath Expression dialog that appears, enter an XPath expression that evaluates to a boolean that will select the footnotes to display, and click OK. Note that the context node of the XPath expression will be the footnote element.

That filtering has been specified is indicated by an icon in the template tag (see screenshot above). (see screenshot below).
XML Node

The XML Node template is intended to be used within an XBRL object template. It accesses the XML node associated with the XBRL object template, and so enables regular XML access and operations within the XML Node template, especially using XPath. For example, Auto-Calculations can be used and will have the element represented by the XBRL object as its context node.

In the screenshot above, both XBRL object templates contain XML Node templates.

- The first XML Node template returns the contents of all the period elements that correspond to the Period XBRL objects that are generated.
- The second XML Node template returns the contents of all the identifier elements that correspond to the Identifier XBRL objects that are generated. In addition an Auto-Calculation is used. The context node for the Auto-Calculation will be the identifier element.
14.5 Open, Closed, and Classic Modes

The following XBRL templates can be used in open or closed mode:

- Concept
- Period
- Identifier
- Unit
- Non-XDT segment
- Non-XDT scenario
- Explicit dimension
- Typed dimension

When one of the above templates is given a setting of open mode, the template iterates over all aspects of the selected type in the instance document. In closed mode, the aspect selection is restricted by the definition of the closed mode. To switch the behavior of a template between open and closed modes, select the template, and, in the template's context menu (obtained by right-clicking the template; screenshot below), hover over the Template Behavior command and select the mode you want.

If you switch a template from open to closed mode, the template's closed-mode properties dialog appears, in which you can define the restrictions you want to apply in closed mode. If the template is already in closed mode, selecting the XBRL Closed Mode command displays the aspect's closed-mode properties dialog, in which you can modify the restrictions. Different types of aspects have different closed-mode dialogs. For example, the Concept aspect has a different closed-mode properties dialog than the Period aspect or Identifier aspect. Each of these dialogs is described in the subsections of this section.

Classic mode

Some templates can also be given a setting of classic mode. This mode is a legacy mode that is similar to open mode. The difference between the two modes is this: In classic mode, an XBRL aspect template (for example, Period) restricts contexts (and the Item template returns facts reported for those contexts); in open mode, however, an XBRL aspect template restricts facts directly (and the Item template returns these facts).
Concept Properties

The Concept Properties dialog (*screenshot below*) appears when you click the Template Behavior | Closed Mode command of a Concept template's context menu. The dialog displays all the available concepts for which facts can be reported. You can select the concept over which you wish the closed-mode Concept template to iterate.

The closed mode template can then be used to display aspects of the selected Concept, such as Concept QName (by using the property's aspect value templates), or to select instance-document facts that are identified by the defined concept. The screenshot below shows both types of use.

In the first type of use, shown in the *Iterate Values* section of the screenshot above, two closed-mode concepts are defined, and their Concept QName values are returned. In the second type of use, shown in the *Fact Selection* section of the screenshot above, facts are selected for the
concept defined in the closed-mode template.

**Note:** Open mode templates behave similarly. However, they do not restrict the definition of the concept to a single concept. They iterate all concepts that satisfy the constraints in effect at that point in the design hierarchy.
Period Properties

The Period Properties dialog (screenshot below) appears when you click the Template Behavior | Closed Mode command of the Period template's context menu.

The dialog enables you to specify an Instant, Duration, or Forever value of the Period. Select the Period property you want to define (Instant, Duration, or Forever) by clicking its radio button. To enter a time value for an Instant or Duration, check the Enter time check box. Then select or enter the date (and time). Enter timezone information if this is required. Click OK when done. The Period elements that match the property defined in this dialog will be returned.

You can use the closed-mode template to display aspects of the Period, such as the Period Start (by using the Period's aspect value templates), or to select instance-document facts that contain the defined period. The screenshot below shows both types of use.
### Iterate Values

**Instant period (no time supplied):**

```
<2010-01-01 00:00:00+01:00> Period instant <(content)> Period instant <2010-01-01 00:00:00+01:00>
```

**Instant period (time supplied):**

```
<2009-12-31> Period instant <(content)> Period instant <2009-12-31>
```

**Duration period (no time supplied):**

```
<2015-01-01 00:00:00 - 2015-12-31 00:00:00> Period start <(content)> Period start - Period end <(content)> Period end <2015-01-01 00:00:00 - 2015-12-31 00:00:00>
```

**Duration period (time supplied):**

```
<2015-01-01 - 2015-12-30> Period start <(content)> Period start - Period end <(content)> Period end <2015-01-01 - 2015-12-30>
```

### Fact Selection

**Instant Period (no time supplied):**

```
<2010-01-01 00:00:00> (content) <2010-01-01 00:00:00>
```

**Instant Period (time supplied):**

```
<2010-01-01 00:00:00> (content) <2010-01-01 00:00:00>
```

**Duration Period (no time supplied):**

```
<2015-01-01 00:00:00 - 2015-12-31 00:00:00> (content) <2015-01-01 00:00:00 - 2015-12-31 00:00:00>
```

**Duration Period (time supplied):**

```
<2015-01-01 - 2015-12-30> (content) <2015-01-01 - 2015-12-30>
```
Identifier Properties

The Identifier Properties dialog (screenshot below) appears when you click the Template Behavior | Closed Mode command of the Identifier template's context menu.

![Identifier Properties (Closed Mode)](image)

The dialog enables you to specify the Identifier over which the template will iterate. Enter the value of the instance document's Identifier element and @scheme attribute. To display the value of the @scheme attribute in the template tags, check the corresponding check box. For example, the screenshot below shows two different closed-mode Identifier templates.

- The definition of the first Identifier template (Identifier 1) is shown in the screenshot at left. This template outputs the Identifier's value and scheme using the Identifier's aspect value templates.
- The second Identifier template selects all Concept 06 concepts that have an Identifier value of Minitax 1. The Item template within it returns that concept's value.
Iterate Values

The identifier's text value is: Identifier value (content) Identifier scheme
The identifier's scheme is: Identifier scheme (content)

Fact Selection

Facts reported for identifier "Minitax 1"
Unit Properties

The Unit Properties dialog (screenshot below right) appears when you click the Template Behavior | Closed Mode command of a Unit template's context menu.

- **Fact Selection**

  - Items with unit type "pure"
    - ![Pure](content) ![Rem](content) ![Pure]

  - Items with unit type "shares"
    - ![Shares](content) ![Rem](content) ![Shares]

  - Items with unit type "monetary - USD"
    - ![iso4217:USD](content) ![Rem](content) ![iso4217:USD]

  - Items with unit type "SquareMeters"
    - ![meter * meter](content) ![Rem](content) ![meter * meter]

  - Items with unit type "kilometers per hour"
    - ![kilometer / hour](content) ![Rem](content) ![kilometer / hour]
The screenshot above left shows five different Unit XBRL object templates (Pure, Shares, Monetary, Complex, and Ratio). The templates are defined (in the Properties dialog of each) to iterate over a specific type of Unit.

To specify a particular type of unit, in the Properties dialog, select one of the five unit types:

- Pure and Shares are in the namespace `xmlns:xbrli="http://www.xbrl.org/2003/instance"
- The Monetary type is in the namespace `xmlns:iso4217="http://www.xbrl.org/2003/iso4217"`
The Complex and Ratio types can be in any namespace that you define. The namespaces of measures used for these unit types must be specified in the Namespaces pane of the dialog.

For example, if you wish to iterate over the kilometer/hour unit, define the closed mode template to be a ratio that has the appropriate numerator and denominator (see screenshot below). The namespace/s of the units must be defined in the Namespaces pane.

The Unit X BRL object template will automatically be named with the measures of the unit (see screenshot below).

The Unit templates can be used in the following way (see screenshot below):

- To iterate over the aspect value templates inserted within the Unit XBRL object template, and return the values of these aspect value templates
- To select facts in the XBRL instance that contain the defined units. The Item aspect value template within each Unit template, will then return the facts that each closed-mode Unit template selects
Iterate Values

- iso4217:USD
- meter * meter
- kilometer / hour

Fact Selection

- Shares
- iso4217:USD
Explicit Dimension Properties

The Explicit Dimension Properties dialog (screenshot below) appears when you click the Template Behavior | Closed Mode command of the explicit dimension template's context menu.

The dialog enables you to specify the domain member of an explicit dimension over which the template will iterate. Enter the QName of the explicit dimension and then select the domain member from the list of domains that have been defined for the explicit dimension in the taxonomy. The screenshot above right shows the explicit dimension in the taxonomy structure. To
display the dimension's QName in the template tags, check the Display QName option. The QName is displayed within parentheses (see first explicit dimension template in the screenshot below).

```
<mint:USA (mint:GeographyDimension)
  Dimension QName (content)
  Explicit value QName (content)
  mint:USA (mint:GeographyDimension)

All facts reported for USA
  mint:USA mint:ConceptForGeography Item (content)
  Item mint:ConceptForGeography mint:USA
```

The screenshot above shows two closed-mode explicit dimension templates.

- The first template contains two aspect value templates (inserted via the context menu commands Insert XBRL Element | Dimension QName and Insert XBRL Element | Explicit Dimension Value QName).
- The second template selects all ConceptForGeography concepts that are linked to the USA domain of the GeographyDimension explicit dimension. The Item template within the concept template returns that concept's value.
Typed Dimension Properties

The Typed Dimension Properties dialog (screenshot below left) appears when you click the Template Behavior | Closed Mode command of the typed dimension template's context menu.

In closed mode, a Typed Dimension template iterates over a single element, considered to be a typed dimension value. The element content is defined here and is independent of the content of the XBRL instance.

Enter the QName of the typed dimension:

minitax:Age

Display the dimension's QName in Design View markup

Enter the typed element as it would appear in the XBRL instance (including its start and end tags):

1  <m:AgeValue xmlns:m="http://www.altova.com/minitax">22</m:AgeValue>

Add known namespace bindings to this XML data when generating XSLT

Note: If this option is unchecked, the XML entered above must also define any namespace prefixes it contains, using the 'xmlns:' syntax.
The dialog enables you to specify the typed element of a typed dimension over which the template will iterate. First enter the QName of the typed dimension (see screenshot above left), and then enter the typed element as it would appear in the XBRL instance document, including the start and end tags. The screenshot above right shows the typed dimension in the taxonomy structure. To display the dimension's QName in the template tags, check the Display QName option. The QName is displayed within parentheses (see second typed dimension template in the screenshot below).
The screenshot above shows three closed-mode typed dimension templates.

- The first template contains two aspect value templates (inserted via the context menu commands Insert XBRL Element | Dimension QName and Insert XBRL Element | Typed Dimension Element).
- In the second template, the XML Node template converts the XBRL object to an XML element and returns the contents of the element.
- The third template selects all Concept1ForAge concepts that have a typed element as defined in the typed dimension template. The Item template within the concept template returns that concept’s value.
Non-XDT Segment/Scenario Properties

The Non-XDT Segment Properties dialog (screenshot below) and Non-XDT Scenario Properties dialog appear when you click the Template Behavior | Closed Mode command of the respective Non-XDT component template's context menu. The dialogs for Non-XDT segments and Non-XDT scenarios are similar.

The dialog enables you to specify additional, non-dimensional, XML data that is used to set a constraint on which reported facts are processed by the Item template. The Non-XDT component template (segment or scenario) can then be used as follows:

- The contents of the Non-XDT component template can be returned by using either the aspect value template Non-dimensional elements (first template in the screenshot below) or the XML Node template.
- The third template in the screenshot below selects all facts that contain the non-dimensional data defined in the Non-XDT template.
Iterate Values

Using the AspectValue template:

```
<x:Event>
  <y:Event>
    <z:Event>
      Non-dimensional elements (content) Non-dimensional elements
    </z:Event>
  </y:Event>
</x:Event>
```

Using the XMLNode template:

```
<x:Event>
  <y:Event>
    <z:Event>
      (content)
    </z:Event>
  </y:Event>
</x:Event>
```

Fact Selection

```
<minitax:F>F<minitax:F>

Item (content) Item

<minitax:F>F<minitax:F>
```
14.6 XBRL Tables

The XBRL Table Linkbase specification is designed to allow rendering of XBRL facts in a tabular form. A table linkbase provides the basic layout of a table (or several tables), leaving the table rendering details to the processing application. The layout as provided by the linkbase defines the appearance of the table’s X and Y axes by assigning to each axis an arbitrarily complex structure of header cells. Each structure level corresponds to an XBRL aspect (concept, period, identifier, unit, segment, or scenario). The layout also defines the order of aspect values, filtering of aspect values, and other information.

The Table Wizard
StyleVision integrates table linkbases by reading the table linkbase information from the DTS and allowing the user to select a table definition from the table linkbase. The XBRL Table Wizard automatically fills its row and column breakdowns according to the definition of the selected table. Once the breakdowns are filled and the user completes the wizard, StyleVision generates the XBRL table. StyleVision also supports generating the XBRL table directly after the user selects a table definition, that is, without even opening the wizard.

Adding XBRL tables to the design
You can add an XBRL table to a design in any of the following ways (which are ordered by increasing level of user-friendliness):

- Manually add a table and all required design elements (XBRL templates, etc.) to the design.
- Use the XBRL Table Wizard to create the design elements; manually add/arrange breakdown items in the wizard as required.
- In the XBRL Table Wizard, fill the breakdown items automatically by selecting a table definition from the table linkbase. (The taxonomy must contain a table linkbase.)
- Completely auto-generate the table (without opening the wizard) by selecting a table definition from the table linkbase. (The taxonomy must contain a table linkbase.)

In the third case, the table definition in the table linkbase serves as a starting point for filling the breakdown items in the Table Wizard. The fact that the breakdown items originated in a table linkbase does not in any way restrict subsequent editing actions in the Table Wizard or in design view. Once a table has been selected and the breakdown items have been filled, the presence of a table linkbase no longer influences either the Table Wizard, subsequent design editing, or the XSL transformation.

Table Parameters
A table definition in a table linkbase might depend on parameters. For example, the linkrole of a relationship node might depend on a parameter. In such cases, different parameter values could produce fundamentally different table structures. This means that when a table is selected for filling the breakdown items, values must also be provided for any parameters defined in the table linkbase.
When selecting a table for filling the breakdown items, each table parameter can be assigned a single value only. This means that the resolution process results in a single table, which the user can subsequently customize in the wizard. The effect achieved by assigning multiple items to a parameter's value is reproduced by repeatedly using the Table Wizard to create a table, one for each desired parameter value item.
The XBRL Table Wizard

The XBRL Table Wizard is accessed when a concept is dragged from the source tree and dropped into the design. From the context menu that pops up select Create XBRL Table. The wizard helps you to quickly and easily create an XBRL report in the form of a table. The window of the XBRL Table Wizard consists of three parts:

1. An Available pane containing schema tree items that can be created as columns or rows of the table (see screenshot below);
2. Panes for defining the table structure (screenshot below, see XBRL Table Structure for details);
3. A pane for defining table options (see XBRL Table Options for details).

This section is organized into two parts:

- **XBRL Table Structure** describes how you create the XBRL table structure with the XBRL Table Wizard.
- **XBRL Table Options** explains the various options that can be specified for the XBRL table.
- **Component Properties** describes the property-set of each components.

After specifying the design of the table, click the OK button to insert the table in the design. If you wish to change any of the table design settings after the table has been inserted in the design, delete the table and recreate it. When you drop the concept into the design and open the XBRL Table Wizard, the Wizard will have the settings that were made for the previous table. You can then modify these settings as required and press OK to insert the modified table in the design.
XBRL Table Structure

When a concept from the schema tree is created as an XBRL table, its descendant concepts, periods, identifiers, units, and non-XDT segments and scenarios become available (in the Available pane) for creation as rows and columns of the XBRL table. To create one of these items from the Available pane as columns or rows, drag it, respectively, into the Column Breakdown pane or Row Breakdown pane (screenshot below) and drop it into a placeholder field of the pane. In the screenshot below, the Period item has been created as columns while the Concepts item has been created as rows. To delete an item that has been dropped into a placeholder field of the table structure, select it and press the Delete key.

Concepts

When the Concepts item is dragged into, say the Row Breakdown pane, the descendant concepts (of the concept that is being created as the XBRL table) will be created as the rows of the table. By default all descendant concepts will be created as rows of the table. To deselect individual concepts or to modify the selection, click the Ellipsis icon of the Concepts item in the Row Breakdown pane of the XBRL Wizard. This displays the Concept Properties dialog (screenshot below), in which you can select the descendant concepts to be created as rows. Then click OK.
Notice that the display of the descendant concepts in the Properties dialog is hierarchically structured. In the design, rows will be indented so as to reflect this structure (screenshot below; compare with structure in Properties dialog).

Additional options for the presentation of concepts in the design are available in the Table Options pane and in the Table group of properties in the Properties sidebar.
Periods
There are three types of periods that can be defined for a concept: instants, durations, and forever. What periods are displayed in the table can be specified in the Period Type pane of the Periods Properties dialog (screenshot below). To access the Periods Properties dialog, click the Ellipsis icon in the Periods item.

Note: If a date range is specified in the XBRL Table Options dialog, this range will be applied to periods; so only periods in this range will be processed.

Identifiers
Inserting an Identifier item in a column or row causes each unique identifier value to receive a column or row, respectively. The contents of facts with a particular identifier value will be listed in the column/row for that identifier value. The screenshot below shows the HTML output of an XBRL instance file in which there is only one identifier value: Nanonull Inc.

In the screenshot above, notice that entries in the Nanonull Inc column are a concatenation of two values. This is because, in the instance file, each fact (such as Assets, Total) occurs twice, each occurrence specified for the same identifier (Nanonull, Inc) but for different periods. See the next section, Periods and Identifiers, for a description of how to create separate columns for each unique identifier-period value.

Periods and identifiers
To create separate columns for each identifier's unique periods, do the following. In the XBRL Table Wizard, first drag the Identifier item into the Column Breakdown pane. Then drag the
Periods item to the *Column Breakdown* pane, to the placeholder field below the Identifier item. The placeholder field becomes highlighted (see screenshot below).

![Screenshot of Column Breakdown pane](image)

When the placeholder field becomes highlighted, drop the Periods item on it. The *Column Breakdown* pane will look something like this:

![Updated Column Breakdown pane](image)

The effect of this is to create separate columns for each period of each identifier. The screenshot below shows a table with columns for two periods of the *Nanonull, Inc* identifier.

<table>
<thead>
<tr>
<th>Balance Sheet (in Millions)</th>
<th>Nanonull Inc 2004-09-30</th>
<th>Nanonull Inc 2003-12-31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assets, Total</td>
<td>€21.49</td>
<td>€24.02</td>
</tr>
<tr>
<td>Current Assets, Total</td>
<td>€10.65</td>
<td>€12.32</td>
</tr>
<tr>
<td>Cash and Cash Equivalents</td>
<td>€1.51</td>
<td>€2.72</td>
</tr>
<tr>
<td>Trade Receivables, Current</td>
<td>€4.62</td>
<td>€5.34</td>
</tr>
<tr>
<td>Tax Receivables, Current</td>
<td>€0.24</td>
<td>€0.25</td>
</tr>
<tr>
<td>Inventories</td>
<td>€3.48</td>
<td>€3.59</td>
</tr>
<tr>
<td>Other Assets, Current</td>
<td>€0.79</td>
<td>€0.42</td>
</tr>
<tr>
<td>Non Current Assets, Total</td>
<td>€10.85</td>
<td>€11.7</td>
</tr>
<tr>
<td>Property, Plant and Equipment</td>
<td>€1.21</td>
<td>€1.42</td>
</tr>
<tr>
<td>Intangible Assets</td>
<td>€0.82</td>
<td>€1.11</td>
</tr>
<tr>
<td>Goodwill</td>
<td>€5.27</td>
<td>€6.65</td>
</tr>
<tr>
<td>Other Financial Assets, Non Current</td>
<td>€0.34</td>
<td>€0.34</td>
</tr>
<tr>
<td>Deferred Tax Assets</td>
<td>€2.21</td>
<td>€2.18</td>
</tr>
</tbody>
</table>

In the *Column Breakdown* pane, available items can be dropped repeatedly in the placeholder fields. The screenshot below, for example, shows how the Identifier and Period items have been dropped into adjacent placeholder fields.
This results in output (screenshot below) where the Identifier-Period columns are displayed first (that is, on the left), and these columns are followed (on the right) by Period-Identifier columns.

### Balance Sheet (in Millions)

<table>
<thead>
<tr>
<th></th>
<th>NaN0null Inc 2004-09-30</th>
<th>NaN0null Inc 2003-12-31</th>
<th>2004-09-30 NaN0null Inc</th>
<th>2003-12-31 NaN0null Inc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assets, Total</td>
<td>€ 21.49</td>
<td>€ 24.02</td>
<td>€ 21.49</td>
<td>€ 24.02</td>
</tr>
<tr>
<td>Current Assets, Total</td>
<td>€ 10.65</td>
<td>€ 12.32</td>
<td>€ 10.65</td>
<td>€ 12.32</td>
</tr>
<tr>
<td>Cash and Cash Equivalents</td>
<td>€ 1.51</td>
<td>€ 2.72</td>
<td>€ 1.51</td>
<td>€ 2.72</td>
</tr>
<tr>
<td>Trade Receivables, Current</td>
<td>€ 4.62</td>
<td>€ 5.34</td>
<td>€ 4.62</td>
<td>€ 5.34</td>
</tr>
<tr>
<td>Tax Receivables, Current</td>
<td>€ 0.24</td>
<td>€ 0.25</td>
<td>€ 0.24</td>
<td>€ 0.25</td>
</tr>
<tr>
<td>Inventories</td>
<td>€ 3.48</td>
<td>€ 3.59</td>
<td>€ 3.48</td>
<td>€ 3.59</td>
</tr>
<tr>
<td>Other Assets, Current</td>
<td>€ 0.79</td>
<td>€ 0.42</td>
<td>€ 0.79</td>
<td>€ 0.42</td>
</tr>
</tbody>
</table>

---

**Repeating tables**

The XBRL Table Wizard enables you to create a set of tables that repeat on a specific criterion. In the screenshot below, for example, a table is created with a structure in which concepts are rows and identifiers are columns. This table structure is repeated for each period (the Period item has been dropped into the *Repeat Table For* pane).
## Repeat Table for:
- **Period**

Drag and drop available Periods, Identifiers and Dimensions into placeholder fields.

## Column Breakdown:
- **Identifier**

Drag and drop available Concepts, Periods, Identifiers and Dimensions into placeholder fields.

## Row Breakdown:
- **Concepts**

Drag and drop available Concepts, Periods, Identifiers and Dimensions into placeholder fields.
XBRL Table Options

At the bottom of the XBRL Table Wizard is a list of options for the table being created (screenshots below). Here, you can also switch to US-GAAP mode if your taxonomy is based on US-GAAP and draw on US-GAAP-specific information for use in creating the table. The range for which periods should be considered can also be set (see screenshot below).

Other XBRL table options that may be set are as follows:

- **Use dimensional breakdown in presentation linkbase**: The dimensional structure specified in the presentation linkbase will be used to locate concepts in the table design.
- **Show facts reported with additional dimensions**: When facts have additional dimensions, these are reported.
- **Allow word-wrap in cells**: Word-wrapping in table cells can be switched on.
- **Generate footnotes**: Footnote templates will automatically be generated. There are two parts to footnotes: (i) footnote references are inserted for elements, and (ii) the actual footnotes are created at the bottom of the table.
- **Generate units for monetary items**: A Unit template is automatically inserted for monetary items in the design. This template applies the global template `xbrli:measure`. If this option is unchecked, the Unit template will not be inserted. However, it can be inserted at individual locations in the design at any time via the context menu. See XBRL Templates for details.
- **Display monetary items in**: You can select number units between **Tens** and **Billions** for monetary values. The selected unit will apply to the entire table. Monetary values in the table will be reported in these units, for example, in millions.
- **Generate an Inline XBRL Document (HTML only)**: The HTML output document will contain the Inline XBRL namespace and all other requisites for containing Inline XBRL elements. Clicking the Inline XBRL Options button pops up the Inline XBRL Options dialog (screenshot below), which enables you to specify the default number-formatting of the output document. These options are explained in the Inline XBRL section, they are the same as the options for Non-fraction (regular numbers).
• **Auto-remove empty rows**: Empty rows in the output are removed.
• **Auto-remove empty columns**: Empty columns in the output are removed.
• **Generate design fragments for each XBRL item**: Default design fragments are generated for XBRL items (such as `xbrli:monetaryItemType1`). For the same XBRL item type used multiple times in the taxonomy, a new design fragment is generated for each instance of the XBRL item. Notice that the names of such generated design fragments have a counter appended to it. The counter is incremented by one each time a new design fragment is generated for the same XBRL item. Inline XBRL design fragments are given a prefix of `ix_`.
• **Enable tree view (auto-add indentation)**: In the output, concept rows will be set up in a hierarchy as specified in the presentation linkbase and indented accordingly.
• **Enable interactive expand/collapse buttons (HTML only)**: Since the concepts constitute a hierarchy, concepts with descendants can be expanded or collapsed in HTML output. Expand/collapse buttons are created automatically in the HTML output, and the viewer can use these to customize the HTML display.
• **Enable interactive removal of columns (HTML only)**: If checked, this option creates a remove button at the top of each column in the HTML output. By clicking this button, a viewer can remove a column in the HTML display.
• A time range can be specified for the report. Only facts within this time range will be displayed. Dates for the time range can be selected using the arrow keys. The **Left Arrow** and **Right Arrow** keys can be used to select a date component, the **Up Arrow** and **Down Arrow** keys can be used to change the value of the selected date component up and down, respectively.
Component Properties

Each component has its individual set of properties. To edit a component’s properties, in the XBRL Table Wizard or XBRL Chart Wizard, click the Edit Properties icon of the component (see screenshot below).

This displays the properties dialog of the respective component. The properties dialog of open mode and closed mode are different, since open mode multiple items while closed mode returns the single specified item. These dialogs are described below.

Concept properties

The Concept Properties dialogs are shown below: open mode concepts (left) and closed mode concepts (right). The open mode dialog provides options to filter, label, and sort the concepts being iterated over. The closed mode dialog enables you to select the concept to return. See the section Concept Properties for detailed information.
Period properties

The Period Properties dialogs are shown below: open mode periods (left) and closed mode periods (right). The open mode dialog provides options to filter, label, and sort the periods being returned. The closed mode dialog provides options for entering the instant or duration of the period, with or without a timezone. See the section Period Properties for detailed information.
The XBRL specification distinguishes between three different types of periods:

- **Instant periods** define a specific point in time and are commonly used for e.g. balance sheet reports.
- **Duration periods** define a time frame using a start and end date and are commonly used for e.g. income statement reports.
- **Forever periods** are used to report concepts that are not dependent on any particular time frame.

- Show instant periods
- Show duration periods
- Group periods with same duration
- Show forever periods

**Filter**

- Do not filter
- Filter using XPath (context node: `<xbrli:period>`)
Identifier properties

The Identifier Properties dialogs are shown below: open mode identifiers (left) and closed mode identifier (right). The open mode dialog provides options to filter, label, and sort the identifiers being iterated over. The closed mode dialog enables you to enter the value and scheme of the single specified identifier. See the section, Identifier Properties, for detailed information.
Unit properties

The Unit Properties dialogs are shown below: open mode units (left) and closed mode units (right). The open mode dialog provides options to filter, label, and sort the units being returned. The closed mode dialog provides options for entering the unit type of the specified unit. See the section Unit Properties for detailed information.
Non-dimensional data (segment/scenario) properties

The Non-dimensional Data (Segment) and Non-dimensional Data (Scenario) Properties dialogs are shown below: open mode non-dimensional data (left) and closed mode non-dimensional data (right). The open mode dialog provides options to filter, label, and sort the non-dimensional data being returned. The closed mode dialog enables you to enter the XBRL instance data you require. See the section Non-XDT Segment/Scenario Properties for detailed information.
Explicit dimension properties

The Explicit Dimension Properties dialogs are shown below: open mode explicit dimensions (left) and closed mode explicit dimensions (right). The open mode dialog provides options to filter, label, and sort the explicit dimensions being returned. The closed mode dialog enables you to select the dimension and domain you want. See the section Explicit Dimension Properties for detailed information.
The XBRTable Wizard

Explicit Dimension Properties (Open Mode)

Filter
- Do not filter
- Filter using XPath (context item is the explicit value’s QName)

Label
- No label
- Default label
- Label using XPath (context item is the explicit value’s QName)

Explicit Dimension Properties (Closed Mode)

In closed mode, an Explicit Dimension template iterates over a single domain value, regardless of the content of the XBRL instance. The value is defined by the dimension it is created for, and by its XML content.

Enter the QName of the explicit dimension:

minitax:GeographyDimension

Display the dimension’s QName in Design View markup

Select the domain value:

- [ ] minitax:Asia
- [ ] minitax:Europe
- [x] minitax:USA
- [ ] minitax:GeographyDomain

OK  Cancel
14.8 The XBRL Chart Wizard

The XBRL Chart Wizard is accessed when a concept is dragged from the source tree and dropped into the design. From the context menu that pops up select Create XBRL Chart. The wizard helps you to quickly and easily create chart representation of selected data. The window of the XBRL Chart Wizard consists of four panes:

1. A Chart Settings pane, in which the basic configuration of the chart is specified, for example, the chart's title, height, width, color scheme, label sizes, etc. Chart settings are described in detail in the Charts section of the documentation.
2. An Available pane (see screenshot below) containing schema tree items that can be created as the categories and series of the chart;
3. A Categories pane, in which the X-Axis items of the chart can be specified. For example, the X-Axis of an annual sales chart could have items corresponding to the years of the period under consideration. To remove empty categories from the output, check the Remove Empty Categories check box at the bottom of the dialog.
4. A Series pane for selecting what concepts are to be plotted. For example, in an annual sales chart, if the series members were the revenue and net income concepts, then the facts of these two series members could be plotted for multiple periods. As a result, for each year on the X-Axis, the revenue and net income values would be represented on the Y-Axis (say as bars or as points on line graphs). To remove empty series from the output, check the Remove Empty Series check box at the bottom of the dialog.

This section of the documentation is organized into two parts:

- **Categories** describes how data for the Categories (X-Axis) is specified.
- **Series** explains how members of the series are selected.
- **Component Properties** describes the property-set of each components.

After specifying the settings and data for the chart, click the OK button to insert the chart in the design. If you wish to change any of the chart settings or data selections after the chart has been inserted in the design, right-click the chart and select Edit Chart Settings.
Example files
Examples of XBRL charts can be found in the Charts and XBRL folders in the Example project in the Project Window.

See also
- Charts
- Chart Settings
- Taxonomy Structure
- The XBRL Table Wizard
**Categories**

It is in the Categories pane of the Chart Wizard that the items displayed on the Category Axis are selected. (By default, the Category Axis is the X-Axis; the Y-Axis can be specified in the Advanced Chart Settings if you like. In pie charts, the Category Axis specifies which items are to form the segments of the chart.)

To select items for the Category Axis, first drag one or more components from the Available pane into the Categories pane. (In the screenshot below, the Period component and the explicit dimension for Statement have been dragged to the Categories pane.) Then click the Ellipsis icon of each component, to set its component's properties.

**Properties of components**

A component's properties include settings to set a filter on what is selected, to specify how labels are displayed on the Category Axis, and how to sort the selected Axis items. The section Component Properties contains a complete list of these properties.

**User-defined groupings**

Among the components in the Available tab, are predefined groupings for years and quarters (see screenshot above). These enable you to set up, for example, a category for individual quarters, as shown in the screenshot below.
The output of the category selection shown above would be something like that shown in the screenshot below: the category selection (shown on the X-Axis) are the years and quarters found in the XBRL instance document.

![Dividends per Share](image)

- **See also**
  - Charts
  - Chart Settings
  - Component Properties
  - Taxonomy Structure
  - The XBRL Table Wizard
Series

The Series Axis represents the value of selected data points. For example, you could select the Total Assets concept as one series of data points. In that case, the value of Total Assets facts will be output for every corresponding Category Axis selection (such as accounting dates).

Charts with single series

This kind of chart (with a single series selected as described above) is shown in the screenshot of the bar chart below.

![Bar Chart](image)

This chart selects a single series, the Total Assets concept, and represents its value for the corresponding Category Axis dates. The bar chart can easily be changed into other chart types, such as a line chart or a pie chart (screenshots below), by changing the chart type in the Chart Settings dialog.

![Line Chart](image)

The points on the line chart are the values of the Total Assets concepts for the corresponding Category Axis dates. The pie chart shows the values of the Total Asset concepts for each Category Axis date as a segment of the pie.
Charts with multiple series
In the single series example given above, charts were created for one series: the Total Assets concept. But a chart could have more than one series. In the example below, two series are specified: for the Net Income and Revenues concepts. Both series are shown for four duration periods on the Category Axis.

The bar chart above can be converted to other kinds of chart that can represent multiple series, such as line graph charts (screenshot below). But multiple series cannot be represented in charts such as pie charts—which are limited to one series.
Selecting the series
To select items for the Series Axis, first drag a component from the Available pane into the Series pane. (In the screenshot below, the Concepts component has been dragged to the Series pane.) Then click the Ellipsis icon of each component to set the component's properties.

In the case of the Concepts component, the Concept Properties dialog enables you to select the concepts to display. Each selected concept in this dialog will become a series of the chart.

For details of the properties of the various components, see the section Component Properties.

See also
- Charts
- Chart Settings
- Component Properties
- Taxonomy Structure
- The XBRL Table Wizard
Component Properties

Each component has its individual set of properties. To edit a component's properties, in the XBRL Table Wizard or XBRL Chart Wizard, click the Edit Properties icon of the component (see screenshot below).

This displays the properties dialog of the respective component. The properties dialog of open mode and closed mode are different, since open mode multiple items while closed mode returns the single specified item. These dialogs are described below.

Concept properties

The Concept Properties dialogs are shown below: open mode concepts (left) and closed mode concepts (right). The open mode dialog provides options to filter, label, and sort the concepts being iterated over. The closed mode dialog enables you to select the concept to return. See the section Concept Properties for detailed information.
The Period Properties dialogs are shown below: open mode periods (left) and closed mode periods (right). The open mode dialog provides options to filter, label, and sort the periods being returned. The closed mode dialog provides options for entering the instant or duration of the period, with or without a timezone. See the section Period Properties for detailed information.
The XBRL specification distinguishes between three different types of periods:

- **Instant periods** define a specific point in time and are commonly used for e.g. balance sheet reports.
- **Duration periods** define a time frame using a start and end date and are commonly used for e.g. income statement reports.
- **Forever periods** are used to report concepts that are not dependent on any particular time frame.

**Filter**
- Do not filter
- Filter using XPath (context node: <brl:period>)

**Label**
- No label
- Default label
- Label using XPath (context node: <brl:period>)

Buttons:
- Sort...
- OK
- Cancel
**Identifier properties**

The Identifier Properties dialogs are shown below: open mode identifiers (left) and closed mode identifier (right). The open mode dialog provides options to filter, label, and sort the identifiers being iterated over. The closed mode dialog enables you to enter the value and scheme of the single specified identifier. See the section, Identifier Properties for detailed information.
The Unit Properties dialogs are shown below: open mode units (left) and closed mode units (right). The open mode dialog provides options to filter, label, and sort the units being returned. The closed mode dialog provides options for entering the unit type of the specified unit. See the section Unit Properties for detailed information.
Non-dimensional data (segment/scenario) properties

The Non-dimensional Data (Segment) and Non-dimensional Data (Scenario) Properties dialogs are shown below: open mode non-dimensional data (left) and closed mode non-dimensional data (right). The open mode dialog provides options to filter, label, and sort the non-dimensional data being returned. The closed mode dialog enables you to enter the XBRL instance data you require. See the section Non-XDT Segment/Scenario Properties for detailed information.
Explicit dimension properties

The Explicit Dimension Properties dialogs are shown below: open mode explicit dimensions (left) and closed mode explicit dimensions (right). The open mode dialog provides options to filter, label, and sort the explicit dimensions being returned. The closed mode dialog enables you to select the dimension and domain you want. See the section Explicit Dimension Properties for detailed information.
The X BRL Chart Wizard

**Explicit Dimension Properties (Open Mode)**

- **Filter**
  - Do not filter
  - Filter using XPath (context item is the explicit value's QName)

- **Label**
  - No label
  - Default label
  - Label using XPath (context item is the explicit value's QName)

** Explicit Dimension Properties (Closed Mode)**

In closed mode, an Explicit Dimension template iterates over a single domain value, regardless of the content of the XBRL instance. The value is defined by the dimension it is created for, and by its XML content.

Enter the QName of the explicit dimension:

```
minitax:GeographyDimension
```

- Display the dimension's QName in Design View markup

Select the domain value:

- minitax:Asia
- minitax:Europe
- minitax:USA
- minitax:GeographyDomain

**See also**

- Charts
- Chart Settings
- Taxonomy Structure
- The XBRL Table Wizard
14.9 Properties and Formatting

When an XBRL table is created with the XBRL Table Wizard, it is created so that table properties and formatting can be applied via CSS. The mechanism works as follows:

- Table elements (for example, column headers, row headers, and cell contents) are assigned to distinct classes.
- Class selectors are created for these classes with certain default properties assigned to these selectors. As a result, the table is displayed with the default CSS properties assigned to the various table element classes. The auto-generated class selectors and their properties can be viewed under the Global heading of the Style Repository sidebar (screenshot below).

You can customize the formatting of the table by (i) modifying the default CSS properties of the StyleVision-generated class selectors; (ii) assigning your own classes and defining properties for these classes; (iii) setting additional styles in the Styles sidebar and Properties sidebar. How to do this in each of the three cases is described below.

Modifying properties of generated CSS class selectors

To modify properties of a CSS class selector that has been automatically generated, do the following:
1. In the Style Repository sidebar, expand that selector for which properties are to be modified (see screenshot below).

![Style Repository](image)

2. Select the required property group and then modify the value of the property you wish to change.

For more information about modifying CSS styles via the Style Repository sidebar, see the section, Working with CSS Styles. If you are not sure to what table element a particular selector applies, modify a property such as background-color and then check what element in the table design is affected.

---

**Creating new classes and defining their properties**

To insert a new class selector in the style repository, do the following:

1. Click the Add or Insert icon in the Style Repository toolbar.
2. Type in a name for the newly created selector item and press Enter.
3. In the expanded item box, define the required properties.

For more information about using CSS styles, see the section, Working with CSS Styles.

---

**Properties via the Styles sidebar and Properties sidebar**

Additionally, styles can be set directly on table components via the Styles sidebar and Properties sidebar, that is without using class or id selectors. For a description of table-specific formatting see the sections, Formatting Static and Dynamic SPS Tables and Row and Column Display.

See also

- Charts
- Chart Settings
- Taxonomy Structure
- The XBRL Table Wizard
14.10 Inline XBRL

In an SPS design that uses XSLT 2.0 or XSLT 3.0 (not XSLT 1.0), you can specify that content from an XBRL instance file be output to an (X)HTML file as an Inline XBRL element.

Inline XBRL elements are defined in the Inline XBRL specification and enable XBRL fragments to be embedded in an (X)HTML file. Internet browser applications typically render such elements as inline elements, and styling can be applied with CSS styles as required. The main advantage of having Inline XBRL elements in an (X)HTML file is that when the (X)HTML document is processed by an XBRL-generating application, the Inline XBRL markup in the (X)HTML document indicates the semantics of the XBRL fact, thus enabling the XBRL-generating application to extract the XBRL fact correctly and generate valid XBRL.

Formatting content as Inline XBRL

In an SPS that is based on an XBRL taxonomy, create Inline XBRL elements as follows:

1. Right-click the design component (content placeholder or Auto-Calculation) and select Edit Value Formatting from the context menu. This pops up the Value Formatting dialog (see screenshot below).
2. Select the Format as Inline XBRL Value radio button.
3. In the Generate Inline XBRL Element combo box (see screenshot above), select the required Inline XBRL element. The options for each Inline XBRL element are described below.
Note: Inline XBRL is not supported for single-line or multi-line input field components in the SPS design. For multi-line input fields, although the Inline XBRL element is generated, Internet browsers will display the Inline XBRL element tags.

Specifying the XBRL facts to include
The XBRL instance document that will be used as the source document for the (X)HTML output document might contain more facts than you wish to present in the (X)HTML report. These additional facts can also be included as hidden data in the (X)HTML document. Such data can then be used by applications that generate valid XBRL from the (X)HTML document.

How additional document data from the XBRL instance is to be handled is specified in the Taxonomy Source Properties dialog (accessed via the Properties item in the Design Overview window). In this dialog, you can choose to include (i) only XBRL items that are displayed in the design, or (ii) all XBRL items in the source XBRL instance document. In the latter case, items that are not displayed in the design will be output as hidden data inside the <HEAD> element of the (X)HTML document.

Options for Inline XBRL values
The following Inline XBRL elements can be generated. The various options for each element and important points to note about each element are listed below. You should note that the Format in (X)HTML option applies formatting only to the (X)HTML output.

- **Non-fraction (regular number)**: The format in the (X)HTML output can be selected from the combo box of the Format item. The minimum number of digits before and after the decimal separator can be specified. The Scale option is the power of 10 by which the output value is multiplied. (10 to the power of 0 is 1; 10 to the power of 1 is 10; 10 to the power of 2 is 100, etc.) The Scale value is included in the Inline XBRL element. It will be used by the (X)HTML-to-XBRL generator application to multiply with the content of the Inline XBRL element and so obtain the XBRL item. You must note that, when generating the Inline XBRL element for (X)HTML from the source XBRL instance, the XBRL fact is not automatically modified by the value of the Scale element. You will need to modify the value of the XBRL fact according to the Scale value in an Auto-Calculation.
To display XBRL facts that have a value of zero as a dash, check the Display Zero Value as Dash check box.

- **Fraction (numerator, denominator):** The numerator and denominator of a fraction are formatted separately as regular numbers, and the formatting options for each are the same as for Non-fraction (regular number) described above.

- **Non-numeric (date or arbitrary text):** This option enables date values and text values to be formatted. In the Format in (X)HTML combo box (screenshot below), you can select the Arbitrary Text option or any other from a variety of date formats.
When the *Arbitrary Text* option is selected, you can additionally specify whether HTML elements inside the arbitrary text should be escaped in the generated XBRL instance. Check the *Escape Child Elements* check box to do this.

- **Footnote**: No options are required. A standard Inline XBRL element for footnotes is created.

**See also**

- Taxonomy Source Properties
- Creating an XBRL SPS File
- The XBRL Table Wizard
- XBRL Templates
- Properties and Formatting
14.11 Altova XBRL Extension Functions

**Footnotes and Labels**

altovaext:footnotes(node()) returns schema-element(link:footnote)*

altovaext:labels(xs:QName, xs:string) returns schema-element(link:label)*

---

**Parse XML**

axfi:parse-xml(xs:string) returns document-node(element(*))

axfi:parse-xml(xs:string, element(xbrli:xbrl)) returns document-node(element(*))

---

**Aspects**

axfi:aspect-type(item()) returns xs:string

axfi:aspect-to-base64Binary(item()) returns xs:base64Binary

axfi:aspect-from-base64Binary(xs:base64Binary) returns item()

axfi:aspect-from-base64Binary(xs:base64Binary, element(xbrli:xbrl)) returns item()

---

**Concepts**

axfi:create-concept-aspect(xs:QName) returns axfi:concept-aspect.type

axfi:create-concept-aspect(xs:QName, element(xbrli:xbrl)) returns axfi:concept-aspect.type

axfi:concept-aspect-concept(axfi:concept-aspect.type) returns xs:QName

axfi:concept-aspects-in-instance(element(xbrli:xbrl)) returns axfi:concept-aspect.type*

axfi:concept-aspects-in-facts(schema-element(xbrli:item)*) returns axfi:concept-aspect.type*
**Location aspects**

```
axfi:create-location-aspect(element()) returns axfi:location-aspect.type

axfi:location-aspect-element(axfi:location-aspect.type) returns element()
```

---

**Identifiers**

```
axfi:create-identifier-aspect(xs:string, xs:string) returns axfi:identifier-aspect.type

axfi:create-identifier-aspect(xs:string, xs:string, element(xbrli:xbrl)) returns axfi:identifier-aspect.type

axfi:identifier-aspect-value(axfi:identifier-aspect.type) returns xs:string

axfi:identifier-aspect-scheme(axfi:identifier-aspect.type) returns xs:string

axfi:identifier-aspect-element(axfi:identifier-aspect.type, element(xbrli:xbrl)) returns element(xbrli:identifier)?

axfi:identifier-aspects-in-instance(element(xbrli:xbrl)) returns axfi:identifier-aspect.type*

axfi:identifier-aspects-in-facts(schema-element(xbrli:item)*) returns axfi:identifier-aspect.type*

axfi:identifier-aspects-in-contexts(schema-element(xbrli:context)*) returns axfi:identifier-aspect.type*
```

---

**Periods**

```
axfi:create-period-aspect-instant(xbrli:dateUnion) returns axfi:period-aspect.type

axfi:create-period-aspect-instant(xbrli:dateUnion, element(xbrli:xbrl)) returns axfi:period-aspect.type

axfi:create-period-aspect-start-end(xbrli:dateUnion, xbrli:dateUnion) returns axfi:period-aspect.type

axfi:create-period-aspect-start-end(xbrli:dateUnion, xbrli:dateUnion, element(xbrli:xbrl)) returns axfi:period-aspect.type

axfi:create-period-aspect-forever() returns axfi:period-aspect.type

axfi:create-period-aspect-forever(element(xbrli:xbrl)) returns axfi:period-aspect.type

axfi:period-aspect-is-instant(axfi:period-aspect.type) returns xs:boolean
```
axfi:period-aspect-is-start-end(axfi:period-aspect.type) returns xs:boolean
axfi:period-aspect-is-duration(axfi:period-aspect.type) returns xs:boolean
axfi:period-aspect-is-forever(axfi:period-aspect.type) returns xs:boolean
axfi:period-aspect-instant(axfi:period-aspect.type) returns xs:dateTime
axfi:period-aspect-start(axfi:period-aspect.type) returns xs:dateTime
axfi:period-aspect-end(axfi:period-aspect.type) returns xs:dateTime
axfi:period-aspect-element(axfi:period-aspect.type, element(xbrli:xbrl)) returns element(xbrli:period)?
axfi:period-aspects-in-instance(element(xbrli:xbrl)) returns axfi:period-aspect.type*
axfi:period-aspects-in-facts(schema-element(xbrli:item)*) returns axfi:period-aspect.type*
axfi:period-aspects-in-contexts(schema-element(xbrli:context)*) returns axfi:period-aspect.type*

Non-XDT segments

axfi:create-non-xdt-segment-aspect(element()*) returns axfi:non-XDT-segment-aspect.type
axfi:create-non-xdt-segment-aspect(element()*, element(xbrli:xbrl)) returns axfi:non-XDT-segment-aspect.type
axfi:non-xdt-segment-aspect-elements(axfi:non-XDT-segment-aspect.type) returns element()*
axfi:non-xdt-segment-aspects-in-instance(element(xbrli:xbrl)) returns axfi:non-XDT-segment-aspect.type*
axfi:non-xdt-segment-aspects-in-facts(schema-element(xbrli:item)*) returns axfi:non-XDT-segment-aspect.type*
axfi:non-xdt-segment-aspects-in-contexts(schema-element(xbrli:context)*) returns axfi:non-XDT-segment-aspect.type*

Non-XDT scenarios

axfi:create-non-xdt-scenario-aspect(element()*) returns axfi:non-XDT-scenario-aspect.type
axfi:create-non-xdt-scenario-aspect(element()*, element(xbrli:xbrl)) returns axfi:non-XDT-scenario-aspect.type

axfi:non-xdt-scenario-aspect-elements(axfi:non-XDT-scenario-aspect.type) returns element()*

axfi:non-xdt-scenario-aspects-in-instance(element(xbrli:xbrl)) returns axfi:non-XDT-scenario-aspect.type*

axfi:non-xdt-scenario-aspects-in-facts(schema-element(xbrli:item)*) returns axfi:non-XDT-scenario-aspect.type*

axfi:non-xdt-scenario-aspects-in-contexts(schema-element(xbrli:context)*) returns axfi:non-XDT-scenario-aspect.type*

---

**Dimensions (typed and explicit)**

axfi:create-dimension-aspect-explicit(xs:QName, xs:QName?) returns axfi:dimension-aspect.type

axfi:create-dimension-aspect-explicit(xs:QName, xs:QName?, element(xbrli:xbrl)) returns axfi:dimension-aspect.type

axfi:create-dimension-aspect-typed(xs:QName, element()?) returns axfi:dimension-aspect.type

axfi:create-dimension-aspect-typed(xs:QName, element()?, element(xbrli:xbrl)) returns axfi:dimension-aspect.type

axfi:dimension-aspect-is-explicit(axfi:dimension-aspect.type) returns xs:boolean

axfi:dimension-aspect-is-typed(axfi:dimension-aspect.type) returns xs:boolean

axfi:dimension-aspect-dimension(axfi:dimension-aspect.type) returns xs:QName

axfi:dimension-aspect-explicit-value(axfi:dimension-aspect.type) returns xs:QName?

axfi:dimension-aspect-typed-value(axfi:dimension-aspect.type) returns element()? 

axfi:dimension-aspects-in-instance(element(xbrli:xbrl), xs:QName, xs:boolean) returns axfi:dimension-aspect.type*

axfi:dimension-aspects-in-facts(schema-element(xbrli:item)*, xs:QName, xs:boolean) returns axfi:dimension-aspect.type*

axfi:dimension-aspects-in-contexts(schema-element(xbrli:context)*, xs:QName, xs:boolean) returns axfi:dimension-aspect.type*

---

**Units**
axfi:create-unit-aspect(xs:QName+, xs:QName*) returns axfi:unit-aspect.type

axfi:create-unit-aspect(xs:QName+, xs:QName*, element(xbrli:xbrl)) returns
axfi:unit-aspect.type

axfi:unit-aspect-numerator(axfi:unit-aspect.type) returns xs:QName+

axfi:unit-aspect-denominator(axfi:unit-aspect.type) returns xs:QName*

axfi:unit-aspect-element(axfi:unit-aspect.type, element(xbrli:xbrl)) returns
element(xbrli:unit)?

axfi:unit-aspects-in-instance(element(xbrli:xbrl)) returns axfi:unit-
aspect.type*

axfi:unit-aspects-in-facts(schema-element(xbrli:item)*) returns axfi:unit-
aspect.type*

axfi:unit-aspects-in-units(schema-element(xbrli:unit)*) returns axfi:unit-
aspect.type*

axfi:units-in-instance(element(xbrli:xbrl)) returns element(xbrli:unit)*

axfi:units-in-instance(element(xbrli:xbrl), item()*) returns element(xbrli:unit) *

Facts and items

axfi:facts-in-instance(element(xbrli:xbrl), item()*, xs:boolean) returns
element()*

axfi:items-in-instance(element(xbrli:xbrl), item()*, xs:boolean) returns
element()*

axfi:tuples-in-instance(element(xbrli:xbrl), item()*) returns element()*

axfi:contexts-in-instance(element(xbrli:xbrl)) returns element(xbrli:context)*

axfi:contexts-in-instance(element(xbrli:xbrl), item()*, xs:boolean) returns
element(xbrli:context)*
Chapter 15

Authentic View
15 Authentic View

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.

<table>
<thead>
<tr>
<th>Nanonull, Inc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location: US</td>
</tr>
<tr>
<td>Street: 119 Oakstreet, Suite 4876</td>
</tr>
<tr>
<td>City: Vereno</td>
</tr>
<tr>
<td>State &amp; Zip: DC 29213</td>
</tr>
<tr>
<td>Phone: +1 (321) 555 5155 0</td>
</tr>
<tr>
<td>Fax: +1 (321) 555 5155 4</td>
</tr>
<tr>
<td>E-mail: <a href="mailto:office@nanonull.com">office@nanonull.com</a></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.

**Authentic Preview**

In StyleVision, while editing an SPS, you are able to preview the Authentic View of the assigned Working XML File. If you click the Authentic View tab when no Working XML File has been assigned to the SPS, you are prompted to assign a Working XML File. In Authentic Preview, you can edit the XML document, similarly to standard Authentic View, and the editing changes can be saved to the Working XML File. This section describes Authentic View and how to edit documents in Authentic View.
15.1 **Authentic View Interface**

Authentic Preview is enabled by clicking the Authentic tab of the active document. If no Working XML File has been assigned to the SPS, you are prompted to assign one.

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 Preview window
- 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.
Overview of the GUI

The Authentic Preview provides you with menu commands, toolbar icons, and context menus with which to edit the XML document that is displayed in the Main Window.

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.

Main window
This is the window in which the Working XML document is displayed and edited. It is described in the section, Authentic View main window.

Status Bar
The Status Bar displays the XPath to the currently selected node. In the Authentic Preview of StyleVision, the XPath to the currently selected node is indicated in the Schema Tree, where the currently selected node is highlighted in gray. The XPath in Authentic Preview is not displayed in a status bar.

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.
**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](image)
  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](image)
  Show small markup. XML element/attribute tags are shown without names.

- ![Show large markup](image)
  Show large markup. XML element/attribute tags are shown with names.

- ![Show mixed markup](image)
  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](image)
  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.

**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.

**XML File commands**
The following icons, from left to right, correspond to the commands listed below:

- **Save Authentic XML Data:** Saves the XML data file.
- **Save Authentic XML Data As..:** Saves the XML data file as another file.
- **Reload Authentic View:** Reloads the saved XML data file. Any unsaved changes will be lost.
- **Validate:** Validates the XML data file.
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:

```
<Department>Name
Enter the department
(Name Person)
```

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.

```
<Department Name Person>
```

In large markup, attributes are recognized by the equals-to symbol in the start and end tags of the attribute:

```
=country USA =country
```

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 an 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.
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 `//sect1/para` element, and you append a `sect1` element, then the new `sect1` element will be appended not as a following sibling of `//sect1/para` but as a following sibling of the `sect1` element that is the parent of that `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 `para`, an `italic` element can be replaced by the `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.

<table>
<thead>
<tr>
<th>Attributes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>para</td>
<td></td>
</tr>
<tr>
<td>para</td>
<td></td>
</tr>
<tr>
<td>Desc</td>
<td></td>
</tr>
<tr>
<td>Office</td>
<td></td>
</tr>
<tr>
<td>OrgChart</td>
<td></td>
</tr>
</tbody>
</table>

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.
**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.
15.2 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:

- 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.

To learn how to use all the features of Authentic View, please do the Authentic View Tutorial using either XMLSpy or Authentic Desktop. The Authentic View Tutorial is available with these products.
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).

An editable Authentic View document with no XML markup.

An editable Authentic View document with XML markup tags.

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:

```
<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.

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).

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.)
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.

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>Nanonull, Inc.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Street:</strong> 119 Oakstreet, Suite 4876</td>
</tr>
<tr>
<td><strong>City:</strong> Verano</td>
</tr>
<tr>
<td><strong>State &amp; Zip:</strong> DC 29213</td>
</tr>
<tr>
<td><strong>Phone:</strong> +1 (321) 555 5155</td>
</tr>
<tr>
<td><strong>Fax:</strong> +1 (321) 555 5155 - 9</td>
</tr>
<tr>
<td><strong>E-mail:</strong> <a href="mailto:office@nanonull.com">office@nanonull.com</a></td>
</tr>
</tbody>
</table>

Please note: The icons or commands for editing dynamic tables must not be used to edit static tables.

Administration

<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>1.500</td>
<td>25  4   21</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  2   20</td>
</tr>
<tr>
<td>Loby</td>
<td>Matise</td>
<td>Accounting Manager</td>
<td>963</td>
<td><a href="mailto:l.matise@nanonull.com">l.matise@nanonull.com</a></td>
<td>add Shares</td>
<td>25  7   18</td>
</tr>
</tbody>
</table>

Employees: 3 (20% of Office, 9% of Company)

Non-Shareholders: Frank Further, Loby Matise.

Shares: 1500 (13% of Office, 6% of Company)

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.

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

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](image)

2. Set the cellspacing and cellpadding properties to "0". Your table will now look like this:
3. Place the cursor in the first row to format it, and click the \(\text{Table Properties}\) icon. Click the \textbf{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:

\[
\begin{array}{|c|c|c|}
\hline
\text{Name} & \text{Telephone} & \text{Email} \\
\hline
\end{array}
\]

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 \(\text{Split vertically}\) icon. Your table will look like this:

\[
\begin{array}{|c|c|c|}
\hline
\text{Name} & \text{Telephone} & \text{Email} \\
\hline
\end{array}
\]

5. Now place the cursor in the cell below the cell containing "Telephone", and click the \(\text{Split horizontally}\) 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>Office</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Home</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.
**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**.
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.

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 styleshe 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:

\[ \text{State} = \text{CA} \text{ AND } (\text{City} = \text{Los Angeles} \text{ OR } \text{City} = \text{San Diego} \text{ OR } (\text{City} = \text{San Francisco} \text{ AND } \text{CustomerNr} = 25)) \]

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 \( \text{} \). 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 Authentic XML Data... 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 Authentic XML Data.... 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 required. If the number of displayed records is limited, you may need to navigate to the required record (see Navigating a DB Table).

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 Authentic XML Data.... 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 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 Authentic XML Data...** 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.
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 xs:date datatype requires a lexical format of YYYY-MM-DD. If the date in an 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).

To display the Date Picker (see screenshot), click the Date Picker icon.

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).
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).

   ![](Define_Entities.png)

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).
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 (screenshot 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.
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
- PNG
- BMP
- WMF (Microsoft Windows Metafile)
- EMF (Enhanced Metafile)
- SVG (for PDF output only)

Relative paths

Relative paths are resolved relative to the SPS file.
Keystrokes in Authentic View

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.
Chapter 16

Authentic Scripting
16 **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 Basic *</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>Authentic Desktop Community</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Authentic Desktop Enterprise</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Authentic Browser Community **</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Authentic Browser Ent Trusted ***</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Authentic Browser Ent Untrusted</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* No AuthenticView
** Both Trusted and Untrusted versions
*** 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 (in Altova products other than StyleVision), then a warning dialog (screenshot below) pops up.
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.

**Note:** When StyleVision 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. The warning dialog described above appears in the Authentic View of Altova products other than StyleVision. You, as an SPS designer, should be aware of this.

---

**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.

---

**In this section**

In this section we explain how Authentic Scripting works. The section is organized into the following sub-sections:

- **Scripting Editor**, which describes the interface in which a scripting project containing the scripts for the SPS are saved
- **Macros**, which shows how macros can be associated with design elements, Authentic toolbar buttons, and Authentic context menus
- **Event Handlers**, which shows how event handler scripts are created and used
- **Scripting Options**, which documents the options
Example files
Example SPS files with Authentic Scripting are in the Authentic\Scripting folder of the Examples project in the Project window of StyleVision.
16.1 **Scripting Editor**

StyleVision’s Scripting Editor (*screenshot below*) enables you to add scripts for forms, macros, and event handlers to the SPS.

The Scripting Editor can be started either with the menu command **Authentic | Edit Authentic Scripts** or by clicking the **Authentic Script** item in the **view switcher menu of the Design View tab**.

The scripting language can be changed by doing the following:

1. Right-click the Project item, which is at the top of the Project Tree (*see screenshot above*).
2. In the context menu that pops up, select **Project Settings**.
3. In the Project Settings dialog that appears, select either JScript or VBScript as the scripting language and specify the target .NET framework. The user can then access and use classes and extensions of the selected Microsoft .NET framework: for example, for creating forms.

The **Scripting Editor** and its features are described in detail in the **Programmers’ Reference**.

Every SPS has a single scripting project, which is saved in the SPS. Clicking the **Save** button in Scripting Editor does not save the script to the SPS; it merely saves changes to the scripting project in memory. Changes in the scripting project are saved to the SPS only when the SPS is saved.
16.2 Macros

Macros are single programs that are executed when a user action occurs. Macros do not take a parameter, do not return a value, and do not call other macros directly. However, code that needs to be reused by multiple macros or event handlers can be put in a function in the Global Declarations area.

A macro that is not assigned to a specific user action will not be executed. Valid user actions may be an interaction with a specific design element, a toolbar button click, or the selection of a user-defined context menu item.

The EventContext property

An SPS design element can appear in multiple places. To determine in which context a macro is executed, the AuthenticView interface has a property called EventContext to hold the context. Via the EventContext property, a macro can query the XPath location where the macro was started, evaluate XPath expressions in the current node, and access variables defined in the scope of the location. For more details, see the section AuthenticView API.

Checking macro references

The designer might accidentally rename a macro or delete it from the script project, though it is still referenced in the design. To help prevent or correct such errors, the command Authentic | Check Macro References checks for invalid references. A dialog will pop up to help the user replace or delete the reference.

In this section

In this section we look at how macros can be associated with valid user actions:

- Macros on Design Elements
- Macros on Toolbar Buttons
- Macros on Context Menu Items
Macros on Design Elements

Macros can be assigned to design elements via the Authentic Properties window. In the screenshot below the macro named `OnResetAll` has been selected from a list of available macros and assigned to a button’s `OnClick` event:

![Properties Window](image)

Different design elements support different user actions for which macros can be assigned. The table below gives a complete list.

<table>
<thead>
<tr>
<th>User action</th>
<th>Supported design element</th>
<th>Can cancel event</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OnClick</code></td>
<td>Button, Image</td>
<td>No</td>
</tr>
<tr>
<td><code>OnBeforeLinkClick</code></td>
<td>Link</td>
<td>Yes</td>
</tr>
<tr>
<td><code>OnBeforeChange</code></td>
<td>Content, Rest-of-Content, Input Field, Multiline, Input Field, Checkbox, Radiobutton, Combobox</td>
<td>Yes</td>
</tr>
<tr>
<td><code>OnAfterChange</code></td>
<td>Content, Rest-of-Content, Input Field, Multiline, Input Field, Checkbox, Radiobutton, Combobox</td>
<td>No</td>
</tr>
<tr>
<td><code>OnSetFocus</code></td>
<td>Content, Rest-of-Content, Input Field, Multiline, Input Field, Checkbox, Radiobutton, Combobox</td>
<td>No</td>
</tr>
<tr>
<td><code>OnKillFocus</code></td>
<td>Content, Rest-of-Content, Input Field, Multiline, Input Field, Checkbox, Radiobutton, Combobox</td>
<td>No</td>
</tr>
</tbody>
</table>

Macros assigned to these user actions will be called only on primary user input. They are not called on an undo action, or when the action was from outside Authentic View (for example, from a COM API call). If a macro is to be used to cancel further event processing, the user needs to place a COM API call `AuthenticView.DoNotPerformStandardAction()` in the macro where it is required. For example, it can be put in the macro `OnChangeToEU` (listing below) which has been assigned to a `OnBeforeChange` event for a radio button:

```java
if ( !confirm( "Are you sure to change address data to EU?\nAll existing address data will be deleted!"") )
    AuthenticView.DoNotPerformStandardAction();
```

This will prevent the changing of the radio button value if the user canceled the confirm dialog. This
example is in the file `OnChange.sps` is in the `Authentic\Scripting` folder of the `Examples` project in the Project window. You can open this file and see how the macro works.

**Useful features**

After a macro has been assigned in the Properties window (*screenshot below*), it can be quickly accessed for editing in the Scripting Editor by clicking the **Edit Macro** button to the right of its name (*see screenshot below*).
Macros on Context Menu Items

Macros can also be used to customize Authentic context menus. This enables the SPS designer to add custom menu items that each execute a macro.

There is no GUI to aid in this process. The SPS designer must add an event handler for the On_AuthenticContextMenuActivated event, and then manipulate the menu via the AuthenticView API. This can be seen in the example file OnContextMenu.sps which is in the Authentic\Scripting folder of the Examples project in the Project window. The screenshot below shows the On_AuthenticContextMenuActivated event handler script in the Scripting Editor.

The event handler for the On_AuthenticContextMenuActivated event creates two menu items and assigns macros to them. This example file also contains an event handler for the On_AuthenticLoad event, which disables entry helpers and markup buttons when AuthenticView is loaded.

```
function On_AuthenticLoad( )
{
    // We are disabling all entry helpers in order to prevent user from manipulating XML tree.
    AuthenticView.DisableElementEntryHelper( );
    AuthenticView.DisableAttributesEntryHelper( );

    // We are also disabling the markup buttons for the same purpose.
    AuthenticView.SetToolBarButtonState( 'AuthenticMarkupBlock', authenticToolBarButtonDisabled );
    AuthenticView.SetToolBarButtonState( 'AuthenticMarkupInline', authenticToolBarButtonDisabled );
    AuthenticView.SetToolBarButtonState( 'AuthenticMarkupMixed', authenticToolBarButtonDisabled );
}
```
Custom Buttons

In the SPS design you can create a custom button to add to the Authentic toolbar and specify what macro it will trigger. To add a custom button to the Authentic toolbar, first select the command **Authentic | Custom Toolbar Buttons**, then click the **Add** button at the top right of the Authentic Custom Buttons dialog (see screenshot below) and select the type of custom button you want.

There are two types of custom buttons:

- **Text State Icons** are buttons that change the style of text (say, to bold or italic) and are associated with a particular element. Such elements would typically occur within elements of mixed-content type (that is, in elements that can contain both text and child elements). For example, a **para** element would be of mixed context because it can contain text, and, say, **bold** child elements, and **italic** child elements. In such a case, you can associate a Text State Icon (which is a bitmap image) to a child element of the mixed-content element. So, the **bold** element, for example, could be assigned a specific button image (its Text State Icon). A global template for the **bold** element is then defined to provide the bold formatting of the **bold** element. In Authentic View, the Text State Icon for the **bold** element will appear as a toolbar button. When the Authentic View user selects text within a **para** element of the example cited above and clicks the **bold** element's toolbar button, the selected text will be enclosed with the **bold** element tags and the bold formatting of the **bold** element's global template will be applied to the text.

- **Custom toolbar buttons** are associated with a particular macro.

Custom buttons take the following parameters:
- The location of an image for the button (in the Bitmap field) or text for the button (in the Text field).
- In the Element Name field (available for Text State Icons), enter the name of the element with which you wish to associate the Text State Icon,
- In the Macro Name combo box (available for Custom Toolbar Buttons), select the macro from the dropdown list that you wish to associate with the Custom Toolbar Button. The macros listed here are those that have been saved with the SPS. When you click the Create button, the Scripting Editor of StyleVision opens in its own window, enabling you to quickly and easily create a macro. Clicking the Edit button opens the selected macro for editing in the Scripting Editor.
- You can optionally enter a tooltip (in the Tooltip field) as a guide for the Authentic View user when he or she mouses over the custom button.
- In the Identifier field enter a text string that will be used as the identifier of the custom button. This identifier can then be used in scripting code if the designer wishes to control the button state via the AuthenticView API.

The screenshot above shows the the custom button US has the ChangeAddressToUS macro assigned to it. This custom button uses an image named US.bmp. Text for the button can be entered as a fallback. A tooltip has been entered and the custom button has the identifier Address_US. This example is from the file ToolbarButtons.sps is in the Authentic\Scripting folder of the Examples project in the Project window.
16.3 Event Handlers

Authentic Scripting offers the possibility to define event handlers to react to general—not design-element-specific—user actions. They represent the same mechanism as adding COM listeners for AuthenticView events in external applications, but they can be more conveniently edited and stored within the design.

To add an event handler script, start the Scripting Editor (Authentic | Edit Authentic Scripts). The available event handlers are displayed in the Project pane under the AuthenticView Events node.

Double-click an event to access its script template in the scripting window. In the scripting window, edit the event handler script template as required. See the section Macros on Context Menu Items for an example of how event handlers can be used.
16.4 Scripting Options

To show interactive documents with Authentic Scripting, Authentic Scripting must be turned on in the SPS file properties. Do this by selecting the command **File | Properties**. In the Properties dialog that pops up (*screenshot below*) select the Authentic tab and check **Enable Authentic Scripts**.

Additionally, you can specify whether macros can be run in debug mode from outside StyleVision (that is, in the Authentic View of other Altova products) or not. This feature is useful if you wish to test and debug a macro outside the StyleVision environment.
Chapter 17

Automated Processing
Automated Processing

The functionality of StyleVision together with the various XSLT and output files generated by StyleVision provide powerful automation possibilities. This section describes these capabilities.

StyleVision's file-generation functionality
After you have created an SPS design with StyleVision, you can generate several kinds of XSLT and output files from within the GUI, depending on which edition of StyleVision you are using (Enterprise, Professional, or Basic). The following files can be generated with the File | Save Generated Files command:

- XSLT files for HTML, RTF, FO, and Word 2007+ output.
- FO files, which can be passed to an FO processor (such as Apache's FOP) for creation of PDF output.
- HTML, RTF, PDF, and Word 2007+ output.

As you will notice from the list above, the files that can be saved with StyleVision are of two types:

1. The XSLT files generated by the SPS design, and
2. The final output files (such as HTML).

Note: Additionally, if database sources are used, XML Schema and XML data files can be generated based on the database structure and content.

The processes to generate the final HTML, RTF, and Word 2007+ output files are all one-step processes in which the XML document is transformed by an XSLT stylesheet to the output format. The PDF-generation process, however, requires two steps:

1. Transformation of XML to FO by using an XSLT stylesheet. StyleVision can generate both the XSLT file and the FO file.
2. Processing of the FO file with an FO processor (such as Apache's FOP) to produce PDF output. If an FO processor is set up to be used with StyleVision, then StyleVision can generate PDF output by first transforming the XML to FO using the built-in Altova XSLT Engines, and then processing the FO to PDF with the FO processor you have set up.

FOP and XSLT
One FOP option enables you to specify an input XML file, an input XSLT file, and an output PDF file:

```
  fop -xml input.xml -xslt input.xslt -pdf output.pdf
```

In this situation, FOP uses its built-in XSLT engine to carry out the first-step XML-to-FO transformation. It then passes the result FO document to FOP for the second-step FO-to-PDF processing.
You should be aware, however, that FOP's built-in engine might not support all the XSLT features that StyleVision and RaptorXML support. Consequently, there could errors if an XSLT stylesheet generated by StyleVision is specified as an input for an XML transformation using FOP's built-in XSLT engine. In such cases, use the XSLT engine of RaptorXML (RaptorXML+XBRL) Server to transform to FO, and then supply the FO file to FOP for processing to PDF.

**StyleVision Server and RaptorXML: generating files from outside the GUI**

Additionally to generating XSLT stylesheets and the required output formats via the StyleVision GUI (File | Save Generated Files command), you can generate output files using two other methods:

1. With StyleVision Server, which calls StyleVision's file generation functionality without opening the GUI, you can produce various kinds of output.
2. With RaptorXML, a standalone Altova application that contains Altova's XML(+XBRL) Validator, and XSLT and XQuery Engines. The XSLT Engines in RaptorXML can be used for transformations of XML to an output format by processing XML documents with XSLT stylesheets. The XSLT file will have to be created in advance so that it can be used by RaptorXML. (RaptorXML does not take an SPS as an input parameter.) The advantages of using RaptorXML are: (i) speed, as a result enabling fast transformations of large files; and (ii) in addition to a command line interface, RaptorXML provides interfaces for COM, Java, and .NET, and can therefore be easily called from within these environments. How to use RaptorXML for transformations is explained in the sub-section RaptorXML.
3. Multiple transformations can be carried out according to pre-set triggers (such as a daily time) using Altova StyleVision Server within an Altova FlowForce Server workflow. This is described in the section Automation with FlowForce Server.
17.1 Command Line Interface

StyleVision functionality can be called from the command line in two ways:

- By calling the StyleVision executable. This provides access to StyleVision's XSLT-file-generation functionality. The XSLT files are generated from the SPS file.
- By using StyleVision Server to generate output files (HTML, etc). The output files are generated from a PXF file, which is a package of an SPS file with its related files (XML, XSD, image files, etc). The PXF file is generated from StyleVision.

How to use the command line

There are two ways you can use the command line:

- Commands can be entered singly on the command line and be executed immediately. For example, in a command prompt window, you can enter a command for StyleVision or StyleVision Server, and press Enter to execute the command.
- A series of commands can be entered in a batch file for batch processing. For example:

  ```
  @ECHO OFF
  CLS
  StyleVision TestEN.sps -outxslt=HTML-EN.xslt
  StyleVision TestDE.sps -outxslt=HTML-DE.xslt
  StyleVision TestES.sps -outxslt=HTML-ES.xslt
  ```

  When the batch file is processed, the commands are executed and the files are generated.

StyleVision functionality in scheduled tasks

Using the Scheduled Tasks tool of Windows, StyleVision commands can be set to execute according to a predefined schedule. Either a single command or a batch file can be specified as the task to be executed. How to create such commands is described in How to Automate Processing.
StyleVision

The syntax for command line use is:

    StyleVision [SPS File] [options]

where

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StyleVision</td>
<td>calls StyleVision, which is located in the StyleVision application folder</td>
</tr>
<tr>
<td>&lt;SPS File&gt;</td>
<td>specifies the SPS file</td>
</tr>
<tr>
<td>&lt;options&gt;</td>
<td>One or more of the options listed below.</td>
</tr>
</tbody>
</table>

When a command is executed StyleVision runs silently (i.e. without the GUI being opened), generates the required output files, and closes. If an error or warning is encountered, the GUI is opened and the corresponding message is displayed in a message box.

Note: For the SPS to load correctly in StyleVision, the XSD and Working XML files that the SPS uses must be at the locations specified for them in the SPS.

---

Options

Options may be entered in any order.

- **XSLT file output**
  - `-OutXSLT=<file>` Writes XSLT-for-HTML to the specified file
  - `-OutXSLRTF=<file>` Writes XSLT-for-RTF to the specified file
  - `-OutXSLFO=<file>` Writes XSLT-for-FO to the specified file
  - `-OutXSLWord2007=<file>` Writes XSLT-for-Word 2007+ to the specified file

- **DB data output**
  - `-OutDBXML=<file>` Writes XML generated from DB to the specified file.
  - `-OutDBSchema=<file>` Writes XML Schema generated from DB to the specified file.

---

Examples

```
StyleVision "QuickStart.sps" -outxslt="QuickStartHTML.xslt"
StyleVision "C:\Test\QuickStart.sps" -outxslt="C:\Test\QuickStartHTML.xslt"
```
Points to note
Note the following points:

- Paths may be absolute or relative and should use backslashes.
- If the filename or the path to it contains a space, then the entire path should be enclosed in quotes. For example: "c:\My Files\MyXML.xml" or "c:\MyFiles\My XML.xml".
- Commands, paths, and folder and file names are case-insensitive.
StyleVision Server

StyleVision Server can be used via its command line interface (CLI) on Windows, Linux, and Mac OS systems to transform XML files into output HTML, PDF, RTF, and DOCX documents. The StyleVision Server CLI's `generate` command takes an XML file and a PXF file as its two arguments, and the desired output formats as its parameters. The XSLT stylesheets for the transformation are obtained from the PXF file submitted as input.

The only advantage of using StyleVision Server's CLI over RaptorXML Server's CLI is that StyleVision Server can take PXF files as its input (RaptorXML takes an XSLT file as its input). StyleVision Server is however best used when used as part of an Altova FlowForce workflow. A FlowForce workflow can start transformation jobs according to preset triggers: Multiple files can be transformed automatically within a network when the FlowForce job is triggered. See the section Automation with FlowForce Server for more information.

For more information about the StyleVision Server CLI, see the StyleVision Server documentation.

Output files

StyleVision Server can generate one or more of the following files:

- XSLT-for-HTML (.xslt) file from the specified SPS
- HTML (.html) file using the XML and XSLT files in the specified SPS or using alternative XML and/or XSLT files
- XSLT-for-RTF (.xslt) file from the specified SPS
- RTF file using the XML and XSLT-for-RTF files specified in the SPS or using alternative XML and/or XSLT-for-RTF files
- XSLT-for-FO (.xslt) file from the specified SPS
- FO file using the XML and XSLT-for-FO files specified in the SPS or using alternative XML and/or XSLT-for-FO files
- PDF file using XML and XSLT-for-FO specified in the SPS or using alternative XML and/or XSLT-for-FO files
- XSLT-for-Word 2007+ (.xslt) file from the specified SPS
- Word 2007+ file using the XML and XSLT-for-Word 2007+ files specified in the SPS or using alternative XML and/or XSLT-for-Word 2007+ files
- XML Schema file of a database-based SPS
- XML data file of a database-based SPS
17.2 Using RaptorXML

Altova RaptorXML is Altova’s third-generation, hyper-fast XML and XBRL processor. XBRL processing is available only in RaptorXML+XBRL Server. It has been built to be optimized for the latest standards and parallel computing environments. Designed to be highly cross-platform capable, the engine takes advantage of today’s ubiquitous multi-core computers to deliver lightning fast processing of XML and XBRL data.

RaptorXML is available in two editions:

- RaptorXML Server edition, 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.

For more information about RaptorXML, see the Altova website.

---

Typical use-cases

The functionality of RaptorXML that would be most relevant to StyleVision users is the XSLT transformation functionality. Typically, this functionality would be used as follows:

1. An XSLT stylesheet is generated from an SPS with the File | Save Generated Files command. Note that RaptorXML cannot be used to generate XSLT stylesheets from an SPS file.
2. The generated XSLT stylesheet is used to transform XML documents with RaptorXML. With RaptorXML you can generate HTML, RTF, FO, and Word 2007+ output.

---

Advantages of RaptorXML

The advantages of using RaptorXML are as follows:

- RaptorXML provides very fast validation and XSLT transformation, and is therefore useful for dealing with large files.
- Easy use with command line, COM, Java, and .NET interfaces.
- Automation and scheduling with the use of batch files and the scheduling processes such as the Scheduled Tasks process of Windows.

For a description of how RaptorXML can be used to automate the production of output documents (such as HTML) from XML source documents, see the section How to Automate Processing.

For additional and more detailed information about using RaptorXML, including how to use RaptorXML’s COM, Java, and .NET interfaces, see the RaptorXML user documentation.
PDF Output

To generate PDF output from an XML document requires two steps:

1. The XML document is transformed by an XSLT stylesheet. An XSLT transformation engine (such as that of RaptorXML) is used for this transformation. The result is an FO document.
2. The FO document is processed by an FO processor (such as Apache's FOP) to generate the PDF output. StyleVision can be set up to pass the FO result of an XSLT transformation to an FO processor. In StyleVision, the result of PDF generation is displayed in the PDF Preview window or can be saved as a file (via the File | Save Generated Files command).

RaptorXML and PDF

Since RaptorXML does not provide parameters to direct the FO output of an XSLT transformation to an FO processor, you will be left with an FO document as the result of the XSLT transformation step (the first step of the two-step PDF-generation process).

The FO document must now be passed to an FO processor for second-step processing from FO to PDF. The instructions for carrying out this step vary according to the processor being used. For example, in the case of the Apache FOP processor, the following simple command can be used to identify the input FO document and specify the name and location of the output PDF document:

```
fop -fo input.fo -pdf output.pdf
```

FOP offers other parameters, and these are listed in the FOP user reference.

FOP and XSLT

One FOP option enables you to specify an input XML file, an input XSLT file, and an output PDF file:

```
fop -xml input.xml -xslt input.xslt -pdf output.pdf
```

In this situation, FOP uses its built-in XSLT engine to carry out the first-step XML-to-FO transformation. It then passes the result FO document to FOP for the second-step FO-to-PDF processing.

You should be aware, however, that FOP's built-in engine might not support all the XSLT features that StyleVision and RaptorXML support. Consequently, there could be errors if an XSLT stylesheet generated by StyleVision is specified as an input for an XML transformation using FOP's built-in XSLT engine. In such cases, use the XSLT engine of RaptorXML+XBR) Server to transform to FO, and then supply the FO file to FOP for processing to PDF.
Batch processing to PDF
A quick and simple way to generate PDF by using RaptorXML for the first-step XSLT transformation and FOP for the second-step FO processing would be to write a batch file that combines the two commands. For example:

```
raptorxmlserver xslt --input=Test.xml --output=Test.fo Test.xslt
fop -fo Test.fo -pdf Test.pdf
```

The first command calls RaptorXML and produces test.fo as output. The second command passes test.fo to the FOP processor, which generates the PDF file test.pdf. For more information about batch processing and how batch files can be used to automate processes, see the following section: How to Automate Processing.
17.3 Automation with FlowForce Server

Transformations can be automated over a network by using Altova's FlowForce Server, which is available on Windows, Linux, and Mac OS systems. The process works as follows:

1. From StyleVision, a PXF file is deployed to FlowForce Server (with the File | Deploy to FlowForce command) as a .transformation file. The .transformation file contains all the files and information required to carry out transformations as designed in the SPS. (In the diagram below, the deployment is represented by the connector line running along the top.)

2. After the .transformation file has been deployed to FlowForce Server, jobs can be created in FlowForce that use the .transformation file to generate transformations according to triggers specified in the job definition. (A trigger could be, for example, a specific time every day.) Flow Force jobs are created in the FlowForce Web Server interface (shown in the center of the diagram below), which can be accessed from StyleVision or via an HTTP address. For information about creating FlowForce jobs, see the FlowForce documentation.

3. At execution time, FlowForce Server passes the transformation instructions and relevant files to StyleVision Server, which then carries out the transformation (see diagram below).

The role of StyleVision Server in the FlowForce workflow is shown in the diagram below. (The role of MapForce Server in the workflow is also displayed since FlowForce jobs can be created that send Altova MapForce mappings to the Altova MapForce Server for execution.)
Not that additionally to being invoked by a FlowForce job, StyleVision Server can also be invoked via its command line. Usage is described in the StyleVision Server documentation.
17.4 How to Automate Processing

A batch file (a text file saved with the file extension .bat) contains a sequence of commands that will be executed from the command line. When the batch file is executed, each command in the batch file will be executed in turn, starting with the first and progressing through the sequence. A batch file is therefore useful in the following situations:

- Executing a series of commands automatically (see below).
- Creating a chain of processing commands, where a command requires input produced by a preceding command. (For example, an XML file produced as output of one transformation is used as the input of a subsequent transformation.)
- Scheduling a sequence of tasks to be executed at a particular time.

Batch file with sequence of commands

A sequence of commands to be executed is entered as follows:

```plaintext
@ECHO OFF
CLS
StyleVision TestEN.sps -outxslt=HTML-EN.xslt
StyleVision TestDE.sps -outxslt=HTML-DE.xslt
StyleVision TestES.sps -outxslt=HTML-ES.xslt
```

When the batch file is processed, the commands are executed and the files generated. The batch file above uses StyleVision to generate three XSLT files from an SPS file.

Batch processing to PDF

A quick and simple way to generate PDF by using RaptorXML for the first-step XSLT transformation and FOP for the second-step FO processing would be to write a batch file that combines the two commands. (For the sake of simplicity, assume that the calls to RaptorXML and FOP correctly locate the executables.)

```plaintext
raptorxml xslt --input=Test.xml --output=Test.fo Test.xslt
fop -fo test.fo -pdf test.pdf
```

The first command calls RaptorXML and produces test.fo as output. The second command passes test.fo to the FOP processor, which generates the PDF file test.pdf. For more information about RaptorXML, see Using RaptorXML.
Chapter 18

StyleVision in Visual Studio
18 StyleVision in Visual Studio

StyleVision can be integrated into the Microsoft Visual Studio 2005/2008/2010/2012/2013. This unifies the best of both worlds, integrating advanced SPS file creation capabilities with the advanced development environment of Visual Studio.

In this section, we describe:

- The broad installation process and the integration of the StyleVision plugin in Visual Studio.
- Differences between the Visual Studio version and the standalone version.

See also

- StyleVision Integration
18.1 Installing the StyleVision Plugin

To install the StyleVision Plugin for Visual Studio, you need to do the following:

2. Install StyleVision (Enterprise or Professional Edition).
3. Download and run the StyleVision integration package for Microsoft Visual Studio. This package is available on the StyleVision (Enterprise and Professional Editions) download page at www.altova.com. (Please note: You must use the integration package corresponding to your StyleVision version (current version is 2015).)

Once the integration package has been installed, you will be able to use StyleVision 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.

See also

- StyleVision Integration
18.2 Differences with StyleVision Standalone

This section lists the ways in which the Visual Studio versions differ from the standalone versions of StyleVision.

**Entry helpers (Tool windows in Visual Studio)**

The entry helpers of StyleVision are available as Tool windows in Visual Studio. The following points about them should be noted. (For a description of entry helpers and the StyleVision GUI, see the section, User Interface.)

- 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.

**StyleVision commands as Visual Studio commands**

Some StyleVision 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**: StyleVision projects are handled as Visual Studio projects.
- **Customize Toolbars, Customize Commands**: The Toolbars and Commands tabs in the Customize dialog (Tools | Customize) contain both Visual Studio commands as well as StyleVision commands.
- **Views**: In the View menu, the command **StyleVision** contains options to toggle on entry helper windows and other sidebars, and to switch between the editing views, and toggle certain editing guides on and off.
- **StyleVision Help**: This StyleVision menu appears as a submenu in Visual Studio's Help menu.

---

See also

- **StyleVision Integration**
Chapter 19

StyleVision in Eclipse
19 **StyleVision in Eclipse**

Eclipse is an open source framework that integrates different types of applications delivered in the form of plugins.

The StyleVision Plugin for Eclipse enables you to access the functionality of StyleVision from within the Eclipse 4.2 / 4.3 / 4.4 Platform. It is available on Windows platforms. In this section, we describe **how to install** the StyleVision Plugin for Eclipse and how to set up the **StyleVision perspective**. After you have done this, components of the StyleVision GUI and StyleVision menu commands will be available within the Eclipse GUI.

- **See also**
  - [StyleVision Integration](#)
19.1 Installing the StyleVision Plugin for Eclipse

Before installing the StyleVision Plugin for Eclipse, ensure that the following are already installed:

- StyleVision Enterprise or Professional Edition.
- Java SE Runtime Environment 5.0 (JRE 5.0) or higher, which is required for Eclipse. See the Eclipse website for more information. Install a 32-bit or 64-bit JRE to match your version of StyleVision (32-bit or 64-bit).
- Eclipse Platform 4.2 / 4.3 / 4.4. Install a 32-bit or 64-bit Eclipse to match your version of StyleVision (32-bit or 64-bit).

After these have been installed, you can install the StyleVision Plugin for Eclipse, which is contained in the StyleVision Integration Package (see below).

Note on JRE

If, on opening a document in Eclipse, you receive the following error message:

```
java.lang.UnsupportedClassVersionError: com/altova/....
(Unsupported major.minor version 49.0)
```

it indicates that Eclipse is using an older JRE that is on your machine. Since Eclipse uses the PATH environment variable to find a javaw.exe, the problem can be solved by fixing the PATH environment variable so that a newer version is found first. Alternatively, start Eclipse with the command line parameter `-vm`, supplying the path to a javaw.exe of version 5.0 or higher.

StyleVision Integration Package

The StyleVision Plugin for Eclipse is contained in the StyleVision Integration Package and is installed during the installation of the StyleVision Integration Package. Install as follows:

1. Ensure that StyleVision (Enterprise or Professional Edition), JRE, and Eclipse are already installed (see above).
2. From the Components Download page of the Altova website, download and install the StyleVision Integration Package. There are two important steps during the installation; these are described in Steps 3 and 4 below.
3. During installation of the StyleVision Integration Package, a dialog will appear asking whether you wish to install the StyleVision Plugin for Eclipse (see screenshot below). Check the option and then click Next.
4. In the next dialog ((Eclipse) Installation Location, *screenshot below*), you can choose whether the Install Wizard should integrate the StyleVision Plugin into Eclipse during the installation (the *Automatically* option) or whether you will integrate the StyleVision Plugin into Eclipse (via the Eclipse GUI) at a later time.
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 StyleVision 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 StyleVision Plugin for Eclipse will be integrated in Eclipse and will be available when you start Eclipse the next time.

Manually integrating the StyleVision plugin in Eclipse
To manually integrate the StyleVision 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\Common2015\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\StyleVision2015\eclipse\UpdateSite`, and providing a name such as 'Altova StyleVision'.
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 StyleVision Enterprise or Professional Edition license) is required for the StyleVision 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 StyleVision Plugin for Eclipse, select the Eclipse menu option Help | About Eclipse. Then select the StyleVision icon.

See also
- StyleVision Integration
19.2 Stylevision Entry Points in Eclipse

The following entry points in Eclipse can be used to access StyleVision functionality:

- **StyleVision Perspective**, which provides StyleVision's GUI features within the Eclipse GUI.
- **StyleVision toolbar buttons**, which provides access to StyleVision Help and the Create New Document functionality.

---

**StyleVision Perspective**

In Eclipse, a perspective is a configured GUI view with functionality from various applications. When the StyleVision Plugin for Eclipse is integrated in Eclipse, a default StyleVision perspective is automatically created. This perspective is a GUI that includes StyleVision's GUI elements: its editing views, menus, entry helpers, and other sidebars.

When a file having a filetype associated with StyleVision is opened (.sps, for example), this file can be edited in the StyleVision 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 advantages 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 StyleVision perspective involves the following:

- Switching to the StyleVision perspective.
- Setting preferences for the StyleVision perspective.
- Customizing the StyleVision perspective.

---

**Switching to the StyleVision perspective**

In Eclipse, select the command **Window | Open Perspective | Other**. In the dialog that pops up (screenshot below), select **StyleVision**, and click **OK**.
The empty window or the active document will now have the StyleVision 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 (screenshot below).

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 StyleVision 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 StyleVision toolbars. To access the Preferences dialog (screenshot below), select the command Window | Preferences.

In the list of perspectives in the left pane, select StyleVision, then select the required preferences. Finish by clicking OK.

Customizing the StyleVision 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 StyleVision perspective), make the perspective active (in this case the StyleVision 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.

**StyleVision toolbar buttons**

Two StyleVision-related buttons are created automatically in the toolbar (*screenshot below*).

These are for, from left: (i) opening the StyleVision Help, and (ii) creating new StyleVision documents (corresponding to commands in StyleVision's **File** menu).

**See also**

- **StyleVision Integration**
Chapter 20

User Reference
20 User Reference

This section contains a complete description of StyleVision toolbars, Design View symbols, and menu commands. It is divided into the following broad parts:

- An explanation of symbols used in Design View.
- A description of the Edit XPath Expression dialog.
- A description of all the toolbars with their icons, as well as a description of how to customize the views of the toolbars.
- All menu commands.

While the User Reference section contains a description of individual commands, the mechanisms behind various StyleVision features are explained in detail in the relevant sections. The mechanisms have been organized into the following groups:

- SPS File Content
- SPS File Structure
- SPS File Advanced Features
- SPS File Presentation
- SPS File Additional Functionality
- SPS File and Databases
- SPS File and XBRL

For command line usage, see Command Line Interface: StyleVisionBatch.

See also
- User Interface
- Quick Start Tutorial
20.1 Design View Symbols

An SPS design will typically contain several types of component. Each component is represented in the design by a specific symbol. These symbols are listed below and are organized into the following groups:

- **Nodes in the XML document**
- XML document content
- Data-entry devices
- Predefined formats
- XPath objects
- URI objects

Each of these component types can:

- be moved using drag and drop;
- be cut, copied, pasted, and deleted using (i) the commands in the Edit menu, or (ii) the standard Windows shortcuts for these commands;
- have formatting applied to it;
- have a context menu pop up when right-clicked.

---

**Nodes in the XML document**

Element and attribute nodes in the XML document are represented in the SPS design document by tags. Each node has a start tag and end tag. Double-clicking either the start or end tag collapses that node. When a node is collapsed all its contents are hidden. Double-clicking a collapsed node expands it and displays its content.

The following types of node are represented:

- **Document node**

  ![Document node example]

  The document node (indicated with $XML$) represents the XML document as a whole. It is indicated with a green $XML$ tag when the schema source is associated with an XML document, and with $DB$ when the schema source is associated with a DB. The document node in the screenshot at left is expanded and contains the OrgChart element, which is collapsed. The document node in the screenshot at right is collapsed; its contents are hidden.

- **Element node**

  ![Element node example]

  An element node is inserted together with all its ancestor elements if the ancestors are not present at the insertion point. In the screenshot above, the Name element node is shown expanded (left) and collapsed (right).
• **Attribute node**

An **attribute node** is inserted together with all its ancestor elements if the ancestors are not present at the insertion point. Attribute names contain the prefix `=`. In the screenshot above, the `href` attribute node is shown expanded *(left)* and collapsed *(right)*.

Nodes are included in the design as node templates. For information on the various kinds of templates that can be included in the design, see the section, *Templates and Design Fragments*.

---

**XML document content**

XML document content is represented by two placeholders:

- *(contents)*
- *(rest-of-contents)*

The **contents** placeholder represents the contents of a single node. All the text content of the node is output. If the node is an attribute node or a text-only element node, the value of the node is output. If the node is an element node that contains mixed content or element-only content, the text content of all descendants is output. In XSLT terms, the **contents** placeholder is equivalent to the `xsl:apply-templates` element with its `select` attribute set for that node.

**Note:** When applied to an element node, the **contents** placeholder does not output the values of attributes of that element. To output attribute nodes, you must explicitly include the attribute in the template (main or global).

The **rest-of-contents** placeholder applies templates to the rest of the child elements of the current node. The template that is applied for each child element in this case will be either a global template (if one is defined for that element) or the default template for elements (which simply outputs text of text-only elements, and applies templates to child elements). For example, consider an element `book`, which contains the child elements: `title`, `author`, `isbn`, and `pubdate`. If the definition of `book` specifies that only the `title` child element be output, then none of the other child elements (`author`, `isbn`, and `pubdate`) will be output when this definition is processed. If, however, the definition of `book` includes the **rest-of-contents** placeholder after the definition for the `title` element, then for each of the other child elements (`author`, `isbn`, and `pubdate`), a global template (if one exists for that element), or the default template for elements, will be applied.

---

**Data-entry devices**

In order to aid the Authentic View user edit the XML document correctly and enter valid data, data-entry devices can be used in the design. You can assign any of the following data-entry devices to a node:

- **Input fields (single line or multi-line)**
• **Combo boxes**

--

• **Check boxes**

--

• **Radio buttons**

These tags can be collapsed and expanded by double-clicking an expanded and the collapsed tag, respectively. For a detailed description of how each of these data-entry devices is used, see [Data-Entry Devices](#).

---

**Predefined formats**

Predefined formats are shown in mauve tags, which can be expanded/collapsed by double-clicking.

The screenshot above shows tags for the predefined format `<p> (para), expanded (at left) and collapsed (at right). To apply a predefined format, highlight the items around which the predefined format is to appear (by clicking a component and/or marking text), and insert the predefined format.

---

**XPath objects**

StyleVision features two mechanisms that use XPath expressions:

• **Conditional templates**

  Condition tags are blue. The start tag contains cells. The leftmost cell contains a
question mark. Other cells each contain either (i) a number, starting with one, for each
when condition; and/or (ii) an asterisk for the optional otherwise condition. A condition
branch can be selected by clicking it. The number of the selected condition branch is
highlighted in the start tag, and the template for that branch is displayed (within the start
and end tags of the condition). The XPath expression for the selected condition branch is
also highlighted in the Design Tree. Note that tags for conditions cannot be expanded/
collapsed.

- **Auto-Calculations**

  ![Office Summary: AutoCalc](image)

  Auto-Calculations are represented in Design View by the =\{AutoCalc\} object (see
  screenshot above). The XPath expression for the selected Auto-Calculation is highlighted
  in the Design Tree. The dialog to edit the Auto-Calculation is accessed via the Properties
  sidebar.

**URI objects**

There are three URI-based objects that can be inserted in a design:

- **Images**

  If an image is inserted in the SPS design and can be accessed by StyleVision, it
  becomes visible in Design View. If it cannot be accessed, its place in the SPS is marked
  by an image placeholder.

- **Bookmarks (Anchors)**

  ![n1:OrgChart A A n1:CompanyLogo](image)

  ![n1:OrgChart A n1:CompanyLogo](image)

  Bookmark tags are yellow and indicated with the character λ (screenshots above). A
  bookmark is created with the command Insert | Insert Bookmark, and can be empty or
  contain content. Content must always be inserted after the anchor is created. Anchor
tags can be expanded (screenshot above left) or collapsed (screenshot above right).

- **Links**

  ![Go to top A](image)

  ![A](image)

  Link tags are yellow and indicated with the character λ (screenshots above). A link is
  created with the command Insert | Hyperlink. The object around which the link is
  created can be inserted in the design before or after the link is created. If an item is to be
  created as a link, it should be selected and the link created around it. Link tags can be
  expanded (screenshot above left) or collapsed (screenshot above right).

- **See also**
• Toolbars
• Design sidebars
• Content Editing Procedures
20.2 Edit XPath Expression Dialog

The Edit XPath Expression dialog (screenshot below) is used to create and edit XPath expressions for a range of StyleVision features. The dialog automatically supports the XPath version according to which XSLT version has been selected for the SPS being currently edited (XPath 1.0 for XSLT 1.0, XPath 2.0 for XSLT 2.0, and XPath 3.1 for XSLT 3.0).

The Edit XPath Expression dialog has two modes: (i) Builder mode, for creating XPath expressions, and (ii) Evaluator mode for checking the result of the XPath expression being currently edited. You can switch between the two modes by clicking the respective buttons (Builder and Evaluator).

Click OK when you have completed editing the XPath expression.

Builder mode
In Builder mode, you can build XPath expressions quickly and correctly by either (i) entering the XPath expression in the Expression text box via the keyboard, or (ii) using the entry helpers of Builder mode to insert nodes, operators, and functions by double-clicking them from their respective lists (see screenshot below). When an expression that has been entered in the Expression text box contains errors, the expression is underlined in red, thus alerting you to the problem. Builder mode is described in detail in the section, XPath Expression Builder.
**Evaluator mode**

In Evaluator mode, you can see, in the *Results* pane on the right-hand side of the dialog (see screenshot below), the results of evaluating the currently entered XPath expression. The *Evaluation Context* pane shows the structure and contents of the currently assigned Working XML document. The Edit XPath Expression dialog's Evaluator mode is described in detail in the section, *XPath Expression Evaluator*.

![Screenshot of Edit XPath Expression dialog in Evaluator mode](image)

---

**See also**

- XPath Expression Builder
- XPath Expression Evaluator
- Conditional Templates
- Auto-Calculations
- XSLT Engine Information
XPath Expression Builder

When the Builder button in the Edit XPath Expression dialog is clicked (see screenshot below), entry helper panes to help you build an XPath expression become visible. Double-click an entry in any of these entry helpers to enter it at the current cursor point in the XPath expression. The \textit{X} in the text above the text box—\textit{XPath Expression}—indicates the XPath version. The XPath version depends upon the XSLT version that has been selected for the SPS being currently edited (XPath 1.0 for XSLT 1.0, XPath 2.0 for XSLT 2.0, and XPath 3.0 for XSLT 3.0). To switch the XPath version, switch the \textit{XSLT version of the SPS}.

There are three entry helper panes:

- A schema tree for entering element and attribute nodes in the XPath expression. If the \textit{Absolute XPath} check box is unchecked, then the location path to the selected node is entered relative to the context node (the node in the design within which the XPath expression is being built). An absolute XPath expression starts at the document root, and is used for the selected node if the \textit{Absolute XPath} check box is checked.
- An entry helper pane for: (i) axes (ancestor::, parent::, etc), (ii) operators (for example eq and div), and (iii) expressions (for \# in \# return \#, etc). This pane displays the axes, operators, and expressions either listed alphabetically or grouped by functional category. Select the option you want by clicking the appropriate icon above the pane.
- An entry helper with the functions of the active XPath version either listed alphabetically or grouped by functional category. Select the option you want by clicking the appropriate icon above the pane. The \textit{n} and \textit{t} buttons above the pane display the arguments of functions, respectively as names and datatypes.

Features of the Builder:

- To view a text description of an item in either 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).

Signatures are listed using either the names or datatypes of the function's arguments and output. Select the \text{n} or \text{t} button above the pane to toggle between the two display options.

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 XPath expression's edit field, and (ii) an 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.

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 different signatures exist for a function having the same name, 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.

---

**Building XPath expressions**

The Edit XPath Expression dialog helps you to build XPath expressions in the following ways.

- **Context node and schema tree**
  The Selection text box in the Select Schema Attribute or Element pane immediately shows you the context node. Place the cursor over the text box to see the full path of the context node. In the schema tree below the Selection text box, you can see where the context node occurs and quickly build the XPath expression by referring to the schema tree. The Condition, Auto-Calculation, etc, for which the expression is being created, will be inserted at a location within this context node, and the XPath expression will be evaluated with this node as its context.

- **Inserting a node from the schema tree**
  In the Select Schema Attribute or Element pane, the entire schema is displayed. Double-click a node in the schema tree to insert that node in the XPath expression. If the Absolute XPath check box is not checked, the selected node will be inserted with a location path expression that is relative to the context node. For example, in the screenshot above, the Newsitem element, which is a child of the Newsitems element (the context node), has been inserted with a location path that is relative to the context node (that is, as Newsitem). If the Absolute XPath check box were checked, the Newsitem node would have been inserted as $XML/presswatch/newsitems/newsitem.

- **Namespace information**
  The schema tree in the Select Schema Attribute or Element pane contains a Namespace item. Expanding this item displays all the namespaces declared in the stylesheet. This information can be useful for checking the prefixes of a namespace you might want to use in an XPath expression.

- **Inserting XPath axes, operators and expressions**
  The Select Operator/Expression pane lists the XPath axes (ancestor::, parent::, etc), operators (for example, eq and div), and expressions (for # in # return #, etc) for
the XPath version selected as the XSLT version for the SPS. The display can be toggled between an alphabetical listing and a hierarchical listing (which groups the items according to functionality). To insert an axis, operator, or axis in the XPath expression, double-click the required item.

- **Inserting XPath functions**
  The Select Function pane lists XPath functions alphabetically or grouped according to functionality (click the respective icon at the top of the pane to switch between the two arrangements). Each function is listed with its signature. If a function has more than one signature, that function is listed as many times as the number of signatures. Arguments in a signature are separated by commas, and each argument can have an occurrence indicator (? indicates a sequence of zero or one items of the specified type; * indicates a sequence of zero or more items of the specified type). The arguments can be displayed as names or as datatypes; select the \[n\] or \[t\] button above the pane to toggle between the two display options. Each function also specifies the return type of that function. For example: 

\[=> date ?\] indicates that the expected return datatype is a sequence of none or one date item. Placing the mouse over a function pops up a brief description of the function.

To insert a function in the XPath expression, double-click the required function.

- Java and .NET extension functions can be used in XPath expressions, enabling you to access the functions of these programming languages. The Java and .NET buttons at the bottom of the dialog, pop up info boxes with explanations about how to use Java and .NET extension functions in XPath expressions. For more information about this, see the Extension Functions section of this documentation.

**Note:** Java and .NET extension functions are not supported in the Community Edition of Altova's Authentic View products. They are supported in the Enterprise Editions of these products.
Intelligent editing during direct text entry
If you type an expression directly in the Expression text box, options that are available at that point are displayed in a popup (see screenshot below).

These include elements (such as presswatch in the screenshot above), descendant nodes (presswatch/selection in the screenshot above), XPath functions (fn:upper-case above) and XPath axes (ancestor-or-self above). The list of available options becomes more restricted as the expression is entered in the Expression text box. Go up and down the list of options using the Up and Down keys, and press Enter if you wish to select an option and enter it in the expression.

The Otherwise check box
The Otherwise check box (see screenshot below) adds an Otherwise branch to a conditional template as its last branch. Only one Otherwise branch may be present in a conditional template. When a conditional template is evaluated, the first branch to evaluate to true is executed. If no branch evaluates to true, then, the Otherwise branch is executed if present, otherwise the conditional template is exited without any of its branches being executed. Since the Otherwise branch is triggered only in the event that no preceding branch evaluates to true, it does not need to have a condition defined for it. As a result, when the Otherwise check box is selected, the entry field of the XPath expression is disabled.
For details of how to use the Otherwise condition, see Conditional Templates.

See also
- XPath Expression Evaluator
- Conditional Templates
- Auto-Calculations
- XSLT Engine Information
**XPath Expression Evaluator**

Clicking the **Evaluator** button in the Edit XPath Expression dialog switches the dialog to Evaluator mode (see screenshot below). The dialog in this mode has two panes: the **Evaluation Context** pane and the **Results** pane.

![XPath Expression Evaluator](image)

---

**The XPath expression and its evaluation**

The XPath expression in the **XPath Expression** text box can be edited, and the expression can be evaluated. The results of the evaluation are displayed in the **Results** pane. In the screenshot above, for example, the result of evaluating the XPath expression `count(newsitem)` is displayed as the integer 4.

You can use the functions of the Java and .NET programming languages in the XPath expression. The **Using Java** and **Using .NET** buttons at the bottom of the dialog, pop up info boxes with explanations about how to use Java and .NET extension functions in XPath expressions. For more information about this, see the Extension Functions section of this documentation.

---

**Using Builder mode and switching to Evaluator mode for the results**

If you wish to use entry helpers to build the XPath expression, you can switch to Builder mode (by clicking the **Builder** button), build the expression in Builder mode, then switch to Evaluator mode to see the results of the evaluation.
When is the XPath expression evaluated?
Evaluation is carried out in two mutually exclusive situations:

- *Evaluate on typing:* If this check box is selected, the XPath expression is evaluated: (i) with every keystroke used to edit the expression, and (ii) when the mode is switched from Builder mode to Evaluator mode.
- *Evaluate now:* This button is enabled when the *Evaluate on Typing* option is not checked. Click it to evaluate the expression.

The Evaluation Context pane
The Evaluation Context pane shows the structure and contents of the currently assigned Working XML document. Nodes in the document tree can be expanded or collapsed by clicking the respective icons of individual nodes.

The icons above the pane display or hide the following XML syntactic constructs: (i) processing instructions, (ii) comments, (iii) attributes, (iv) text nodes. You can therefore see the entire XML document structure, together with the text contents of nodes, but you can also hide certain constructs if you wish to reduce clutter in the pane.

Changing the context node for evaluation purposes
You can change the context node of the XPath expression by clicking the node in the document tree that you want as the new context node. If the Evaluate on Typing option is checked, then the result will appear immediately in the Results pane.

This feature is useful for checking results with different context nodes. Note, however, that the actual context node for the expression will be the context node within which the current design component is being created. At runtime, the actual context node will be used, not the context node used in the Evaluator.

Remembering the selected context
If you check this option (located below the Evaluation Context pane) and close the dialog by clicking OK, the last selected context node will be remembered when the dialog is re-opened. If this option is not selected, then the context node in Evaluator mode will be the actual context node of the design component in the document.

This option is useful if you are in the process of testing an XPath expression and wish to save it with a specific context node till you complete your testing. Note, however, that at runtime, the actual context node will be used, not the context node saved in the Evaluator.
See also

- XPath Expression Builder
- Conditional Templates
- Auto-Calculations
- XSLT Engine Information
20.3 Toolbars

A number of StyleVision commands are available as toolbar shortcuts, organized in the following toolbars:

- Formatting
- Table
- Authentic
- RichEdit
- Insert Design Elements
- Design Filter
- Global Resources
- Standard

The icons in each toolbar are listed in the sub-sections of this section, each with a brief description of the corresponding command.

---

Positioning the toolbars

A toolbar can float freely on the screen or can be placed in a toolbar area along any edge of the GUI. Toolbars are most commonly placed along the top edge of the GUI, just below the Menu bar. However, they can also be placed along the side or bottom edges of the GUI.

To position a toolbar in a toolbar area, do the following:

1. Grab the toolbar by its handle (if the toolbar is already in a toolbar area) or by its title bar (if the toolbar is floating).
2. Drag the toolbar to the desired toolbar area, if it exists, and drop it at the desired location in that toolbar area. If no toolbar area exists at the edge along which you wish to place the toolbar, dragging the toolbar to that edge will automatically create a toolbar area there when the toolbar is dropped.

To make a toolbar float freely grab it by its handle, drag it away from the toolbar area, and drop it anywhere on the screen except at an edge or in an existing toolbar area.

---

Switching the display of toolbars on and off

The display of individual toolbars can be switched on and off using any of the following three methods:

- In the View | Toolbars menu (screenshot below), select or deselect a toolbar to, respectively, show or hide that toolbar.
Right-click any toolbar area to display a context menu (screenshot below) that allows you to toggle the display of individual toolbars on and off.

In the Toolbars tab of the Customize dialog (Tools | Customize), toggle the display of individual toolbars on or off by clicking a toolbar’s check-box. When done, click the Close button to close the dialog.

Adding and removing toolbar buttons

Individual toolbar buttons can be added to or removed from a toolbar, that is, they can be made visible or be hidden. To add or remove a button from a toolbar, do the following:

1. In the toolbar where the button to be added or removed is, click the More Buttons button (if the toolbar is in a toolbar area) or the Toolbar Options button (if the toolbar is a floating toolbar). The More Buttons button is an arrowhead located at the right-hand side of the toolbar (in horizontal toolbar areas) or at the bottom of the toolbar (in vertical toolbar areas). The Toolbar Options button is an arrowhead located at the right-hand side of the floating toolbar.

2. In the Add or Remove Buttons menu that pops up, place the cursor over the Add or Remove Buttons menu item (screenshot below). This rolls out a menu which contains the names of the toolbars in that toolbar area plus the Customize menu item (screenshot below).
3. Place the cursor over the toolbar that contains the toolbar button to be added or removed (screenshot above).

4. In the menu that rolls out (screenshot above), click on the name of the toolbar button to add or remove that button from the toolbar.

5. Clicking the Customize item pops up the Customize dialog.

The **Reset Toolbar** item below the list of buttons in each toolbar menu resets the toolbar to the state it was in when you downloaded StyleVision. In this state, all buttons for that toolbar are displayed.

**Note:** The buttons that a toolbar contains are preset and cannot be disassociated from that toolbar. The process described above displays or hides the button in the toolbar that is displayed in the GUI.

---

**See also**

- User Interface
Format

The Format toolbar (screenshot below) is enabled in Design View and contains commands that assign commonly used inline and block formatting properties to the item/s selected in the SPS design.

Predefined HTML formats

The HTML format selected from the dropdown list is applied to the selection in Design View. For example, a selection of div applies HTML's Block (div) element around the current selection in Design View. The HTML format is converted to the corresponding RTF, PDF and Word 2007+ properties for the RTF, PDF and Word 2007+ outputs, respectively.

Text properties

The bold, italic, underline, and strikethrough inline text properties can be directly applied to the current selection in Design View by clicking on the appropriate button. Font style, font size, foreground and background color can also be applied via toolbar buttons.

Alignment

Alignment properties (left-aligned, centered, right-aligned, and justified) can be directly applied to the selection in Design View.

Lists

Lists can be inserted at the cursor insertion point, or the selection in the SPS can be converted to a list.

Hyperlinks

Inserts a hyperlink at the cursor insertion point. See Hyperlink for a description of how to use this command.

See also

- Toolbars
- Formatting and Styles
- Input Formatting
Table

The **Table toolbar** contains commands to structure and format static and dynamic tables in Design View. These commands are shown in the screenshot below (which is that of the Table toolbar customization menu, available when you click the **Customize** button at the right of the toolbar).

<table>
<thead>
<tr>
<th><strong>Add Table Header Column</strong></th>
<th><strong>Add Table Footer Column</strong></th>
<th><strong>Add Table Header Row</strong></th>
<th><strong>Add Table Footer Row</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Insert Row</strong></td>
<td><strong>Append Row</strong></td>
<td><strong>Delete Row</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Insert Column</strong></td>
<td><strong>Append Column</strong></td>
<td><strong>Delete Column</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Join Cell Right</strong></td>
<td><strong>Join Cell Left</strong></td>
<td><strong>Join Cell Above</strong></td>
<td><strong>Join Cell Below</strong></td>
</tr>
<tr>
<td><strong>Split Cell Horizontally</strong></td>
<td><strong>Split Cell Vertically</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Align Top</strong></td>
<td><strong>Center Vertically</strong></td>
<td><strong>Align Bottom</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Delete Table</strong></td>
<td></td>
<td><strong>View Table Markup</strong></td>
<td><strong>View Cell Bounds</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Table Properties...</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Reset Toolbar</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Row and Column operations**

Rows and columns in any SPS table (static or dynamic) can be inserted, appended, or deleted with reference to the cursor location. Rows and columns are inserted before the current cursor location or appended after all rows/columns. The row/column in which the cursor is can also be deleted. These operations are achieved with the **Insert Row/Column**, **Append Row/Column**, or **Delete Row/Column** buttons. You can also add table headers and footers as either columns or rows **Add Table Header/Footer Column/Row**.
Cell operations
An SPS table cell in which the cursor is located can be joined to any one of the four cells around it. The joining operation is similar to that of spanning table cells in HTML. The buttons to be used for these operations are **Join Cell Right/Left/Above/Below**. Also, an SPS table cell in which the cursor is located can be split, either horizontally or vertically, using the **Split Cell Horizontally** and **Split Cell Vertically** buttons, respectively. SPS table cell content can be aligned vertically at the top, in the middle, and at the bottom. The display of cell borders can be switched on and off with the **View Cell Bounds** toggle.

Table operations, properties, display
Placing the cursor in a static or dynamic table and clicking **Delete Table** deletes that table. Table markup can be toggled on and off with the **View Table Markup** command. The **Table Properties** command pops up the Table Properties dialog, in which properties of the table can be defined.

**See also**
- Toolbars
- Tables
- Table Menu
Authentic

The **Authentic toolbar** contains commands for customizing Authentic View and editing XML documents in Authentic View. These commands are shown in the screenshot below (the customization menu for adding and removing Authentic toolbar buttons, available when you click the *Customize* button at the right of the toolbar).

All these features are available to the Authentic View user. They enable you, as the SPS designer, to test the SPS using features at the Authentic View user's disposal.

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save Authentic XML Data</td>
</tr>
<tr>
<td>Save Authentic XML Data as ...</td>
</tr>
<tr>
<td>Reload Authentic View</td>
</tr>
<tr>
<td>Select New Row with XML Data for Editing...</td>
</tr>
<tr>
<td>Validate XML</td>
</tr>
<tr>
<td>F8</td>
</tr>
<tr>
<td>Hide Markup</td>
</tr>
<tr>
<td>Show Small Markup</td>
</tr>
<tr>
<td>Show Large Markup</td>
</tr>
<tr>
<td>Show Mixed Markup</td>
</tr>
<tr>
<td>Append Row</td>
</tr>
<tr>
<td>Insert Row</td>
</tr>
<tr>
<td>Duplicate Row</td>
</tr>
<tr>
<td>Move Row Up</td>
</tr>
<tr>
<td>Move Row Down</td>
</tr>
<tr>
<td>Delete Row</td>
</tr>
<tr>
<td>XML Signature...</td>
</tr>
<tr>
<td>Define XML Entities...</td>
</tr>
<tr>
<td>Define XML Entities...</td>
</tr>
<tr>
<td>Reset Toolbar</td>
</tr>
</tbody>
</table>

**Validating, saving, and reloading XML documents**

While editing an XML document in Authentic View, you can check the validity of the Working XML File by using the *Validate XML* button. Editing changes can be saved to the Working XML File with the *Save Authentic XML Data* button. The XML document can also be reloaded at any time from the last saved version.

**Select new row with XML data for editing**

This command is enabled only in SPSs that are based on an XML DB. The command enables a new row from the XML column to be loaded into Authentic View for editing. See the description of the command for details.
**Markup tags in Authentic View**
In Authentic View, the display of markup tags can be customized. Markup tags can be hidden (Hide Markup), can be shown with node names (Show Large Markup), without node names (Show Small Markup), or with any of these three options for individual nodes (Show Mixed Markup).

---

**Editing of dynamic tables in Authentic View**
In Authentic View, row operations can be performed on dynamic tables. Rows can be inserted, appended, and duplicated, as well as be moved up and down, using the appropriate toolbar buttons (Insert Row, Append Row, Duplicate Row, Move Row Up, Move Row Down, and Delete Row).

---

**Defining DTD Entities**
Entities can be defined at any time while editing the Working XML File in Authentic View. Clicking the Define DTD Entities button pops up the Define DTD Entities dialog. (See Authentic | Define Entities for details of usage.)

---

- See also
  - Toolbars
  - Authentic Menu
RichEdit

The RichEdit toolbar contains commands for marking up text in Authentic View with text-styling properties.

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 enables the Authentic View user to specify the font, font-weight, font-style, font-decoration, font-size, color, background color and alignment of text. Select the text you wish to style in Authentic View and specify the styling you wish to apply via the buttons and controls of the RichEdit toolbar. The text that has been styled will be enclosed in the tags of the styling element.

See also

- Toolbars
- RichEdit
- Authentic Menu
Insert Design Elements

The **Insert Design Elements toolbar** contains icons for commands to insert design elements in the SPS design, and for related commands. The various design elements that can be inserted via these toolbar icons are shown in the screenshot below. There are three types of items in the toolbar:

1. **Design elements**, which are context-node-sensitive (the majority of elements in the toolbar),
2. **Layout elements**, which are independent of node context, and
3. **Grid-related toggles** to aid design.

Design elements

The design elements are the context-node-sensitive elements that are available in the **Insert** menu. To insert a design element using its toolbar icon, do the following:
1. Select the toolbar icon for the element you wish to insert.
2. Click the location in the design where the element is to be inserted. A Insert Design Element for the selected design element (screenshot below) pops up. This displays the schema tree with the context node highlighted. The context node is the node within which the cursor has been placed for the insertion of the design element.

3. If you wish to insert the design element within the currently selected context node, click OK. If you wish to select another context node, do so in the schema tree and then click OK.
4. In the case of some design elements, such as Auto-Calculations, a further step is required, such as the definition of an Auto-Calculation. In other cases, such as the insertion of a user-defined template, the Insert Design Element dialog is skipped. In such cases, another dialog, such as the Edit XPath Expression dialog will pop up. Carry out the required step and press the dialog's OK button.

The design element will be inserted at the end of Step 3 or Step 4, depending on the kind of design element being inserted.

**Layout elements**

There are three layout element commands in the Insert Design Elements toolbar: to insert (i) a layout container; (ii) a layout box; and (iii) a line. Note that layout boxes and lines can only be
inserted within a layout container.

To insert a layout container, select the **Insert Layout Container** icon and then click at the location in the design where you wish to insert the layout container. You will be prompted about the size of the layout container, on selecting which the layout container will be inserted. To insert a layout box, click the **Insert Layout Box** icon, then move the cursor to the location within the layout container at which you wish to insert the layout box and click. The layout box is inserted. Click inside the layout box to start typing. To insert a line, click the **Insert Line** icon, then move the cursor to the location within the layout container at which you wish to start drawing the line. Click to define the start point of the line and then drag the cursor to the desired endpoint. Release the cursor at the end point. The line is inserted and extends from the indicated start point to the indicated end point.

To re-size layout containers and layout boxes, place the cursor over the right or bottom border of the layout container or layout box and drag the border so as to obtain the desired size. To move a layout box, place the cursor over the top or left border of the layout box and, when the cursor turns to a cross, drag the layout box to the new location.

**Grid-related toggles**
The **Show Grid** command toggles the display of the drawing grid on and off. When the **Snap to Grid** command is toggled on, elements created within the layout container, such as layout boxes and lines, snap to grid lines and grid line intersections. The properties of the grid can be set in the Design tab of the Options dialog (**Tools | Options**).

- **See also**
  - Toolbars
  - Insert Menu
  - Layout Containers
**Design Filter**

The **Design Filter toolbar** (*screenshot below*) contains commands that enable you to filter which templates are displayed in the design. Each icon in the toolbar is explained below.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Show only one template</td>
<td>Shows the selected template only. Place the cursor in a template and click to show that template only.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Show all template types</td>
<td>Shows all templates in the SPS (main, global, named, and layout).</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Show imported templates</td>
<td>Toggles the display of imported templates on and off.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Show/Hide main template</td>
<td>Toggles the display of the main template on and off.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Show/Hide global templates</td>
<td>Toggles the display of global templates on and off.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Show/Hide Design Fragments</td>
<td>Toggles the display of Design Fragments on and off.</td>
</tr>
</tbody>
</table>

The Design Filter combo box (*screenshot below*) displays a list of all the templates in the SPS. Selecting a template in the combo box causes the template to be selected in the design. The combo box, therefore, enables you to quickly navigate to the desired template in the design, which is useful if the design has several templates, some of which might be currently hidden.

- **See also**
  - Toolbars
  - Design Tree
Global Resources

The **Global Resources toolbar** *(screenshot below)* enables you: (i) to select the active configuration for the application, and (ii) to access the [Altova Global Resources dialog](#).

Select the active configuration from among the options in the dropdown list of the combo box. Click the Manage Global Resources icon to access the Altova Global Resources dialog.

- **See also**
  - [Altova Global Resources](#)
# Standard

The **Standard toolbar** contains buttons for commands that provide important file-related and editing functionality. These icons are listed below with a brief description. For a fuller description of a command, click the command to go to its description in the Reference section.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Command</th>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="New from XML Schema / DTD" /></td>
<td>New from XML Schema / DTD</td>
<td>Ctrl+N</td>
<td>Creates a new SPS document based on a schema. Clicking the dropdown arrow enables you to create the SPS from a DB or an HTML document, or an empty SPS.</td>
</tr>
<tr>
<td><img src="image" alt="Open" /></td>
<td>Open</td>
<td>Ctrl+O</td>
<td>Opens an existing SPS document.</td>
</tr>
<tr>
<td><img src="image" alt="Reload" /></td>
<td>Reload</td>
<td></td>
<td>Reloads the SPS from the last saved version.</td>
</tr>
<tr>
<td><img src="image" alt="Save Design" /></td>
<td>Save Design</td>
<td>Ctrl+S</td>
<td>Saves the active SPS document.</td>
</tr>
<tr>
<td><img src="image" alt="Save All" /></td>
<td>Save All</td>
<td>Ctrl+Shift+S</td>
<td>Saves all open SPS documents.</td>
</tr>
<tr>
<td><img src="image" alt="Print" /></td>
<td>Print</td>
<td>Ctrl+P</td>
<td>Prints the Authentic View of the Working XML file.</td>
</tr>
<tr>
<td><img src="image" alt="Print Preview" /></td>
<td>Print Preview</td>
<td></td>
<td>Displays a print preview of the Authentic View of the Working XML File.</td>
</tr>
<tr>
<td><img src="image" alt="Cut" /></td>
<td>Cut</td>
<td>Shift+Del</td>
<td>Cuts the selection and places it in the clipboard.</td>
</tr>
<tr>
<td><img src="image" alt="Copy" /></td>
<td>Copy</td>
<td>Ctrl+C</td>
<td>Copies the selection to the clipboard.</td>
</tr>
<tr>
<td><img src="image" alt="Paste" /></td>
<td>Paste</td>
<td>Ctrl+P</td>
<td>Pastes the clipboard item to the cursor location.</td>
</tr>
<tr>
<td><img src="image" alt="Delete" /></td>
<td>Delete</td>
<td>Del</td>
<td>Deletes the selection.</td>
</tr>
<tr>
<td><img src="image" alt="Undo" /></td>
<td>Undo</td>
<td>Alt+Backspace</td>
<td>Undoes an editing change. An unlimited number of Undo actions can be performed at a time.</td>
</tr>
<tr>
<td><img src="image" alt="Redo" /></td>
<td>Redo</td>
<td>Ctrl+Y</td>
<td>Redoes an undo.</td>
</tr>
<tr>
<td><img src="image" alt="Find" /></td>
<td>Find</td>
<td>Ctrl+F</td>
<td>Finds text in Authentic View and Output Views.</td>
</tr>
<tr>
<td><img src="image" alt="Find Next" /></td>
<td>Find Next</td>
<td>F3</td>
<td>Finds the next occurrence of the searched text.</td>
</tr>
<tr>
<td><img src="image" alt="Zoom" /></td>
<td>Zoom</td>
<td></td>
<td>Sets the Zoom Factor of Design View.</td>
</tr>
<tr>
<td><img src="image" alt="Show Small Design Markup" /></td>
<td>Show Small Design Markup</td>
<td></td>
<td>Switches markup tags to small markup format.</td>
</tr>
<tr>
<td><img src="image" alt="Show Large Design Markup" /></td>
<td>Show Large Design Markup</td>
<td></td>
<td>Switches markup tags to large markup format.</td>
</tr>
<tr>
<td>Icon</td>
<td>Command</td>
<td>Shortcut</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><img src="image" alt="XSLT 1.0" /></td>
<td>XSLT 1.0</td>
<td></td>
<td>Sets XSLT 1.0 as the stylesheet language.</td>
</tr>
<tr>
<td><img src="image" alt="XSLT 2.0" /></td>
<td>XSLT 2.0</td>
<td></td>
<td>Sets XSLT 2.0 as the stylesheet language.</td>
</tr>
<tr>
<td><img src="image" alt="XSLT 3.0" /></td>
<td>XSLT 3.0</td>
<td></td>
<td>Sets XSLT 3.0 as the stylesheet language.</td>
</tr>
<tr>
<td><img src="image" alt="Spelling" /></td>
<td>Spelling</td>
<td></td>
<td>Runs a spelling check on the SPS document.</td>
</tr>
</tbody>
</table>

- See also
  - Toolbars
20.4 File Menu

The File menu contains commands for working with SPSs and related files. The following commands are available:

- **New**, to create a new SPS from a variety of sources.
- **Open, Reload, Close, Close All**, to open and close the active file, and to reload the active file.
- **Save Design, Design As, All**, which are commands to save the active SPS and all open SPS files.
- **Save Authentic XML Data, Save As**, enabled in Authentic View, it saves changes to the Working XML File.
- **Save Generated Files**, to save output files that can be generated using the SPS.
- **Web Design**, generates all the files required to run an ASPX application, in the folder location you specify.
- **Properties**, to set the encoding of the output documents, the CSS compatibility mode of the browser, how relative image paths in Authentic View should be resolved, and whether images should be embedded or linked in the RTF (Enterprise and Professional editions) and Word 2007+ (Enterprise edition only) outputs.
- **Print Preview, Print**, enabled in Authentic View and output views, these commands print what is displayed in the previews.
- **Most Recently Used Files, Exit**, respectively, to select a recently used file to open, and to exit the program.
New

Placing the cursor over the New command pops out a submenu (screenshot below) that enables you to create a new SPS document of one of five types:

- A new SPS file based on an XML Schema or DTD or XML Schema generated from an XML file (New from XML Schema / DTD / XML). The selected schema is added to the Design Overview sidebar and a graphical tree representation is added to the schema tree (in the Schema Tree sidebar). In Design View, the SPS is created with an empty main template. A new SPS can also be created from a file (schema or XML) via a URL or global resource (see below).
- A new SPS file based on an XML Schema generated from a DB you select (New from DB or New from XML Column in IBM DB2). The connection process is described in the section Connecting to a DB and Setting up the SPS. The SPS is created in Design View with an empty main template.
- A new SPS file based on an XBRL taxonomy.
- A new SPS based on a user-defined schema you create node-by-node from an HTML file (New from HTML File). The user-defined schema is added to the Design Overview sidebar and Schema Tree sidebar. In the schema tree, it will have a single document element (root element), and the HTML file is loaded in Design View.
- An SPS can be created from an XSLT-for-HTML or an XSLT-for-FO or an FO file. Template structure and styling in the XSLT will be created in the SPS. You can then modify the SPS components and add content and formatting to the SPS. See New from XSLT for details.
- A new empty SPS (New (empty)). No schema is added to either the Design Overview sidebar or the schema tree. An empty main template will be created in Design View.

Note: A global resource can be used to locate a file or DB resource.

Selecting the type of design

After you have selected (XSD and XML) sources files, if required, the Create New Design dialog appears.

The Create New Design dialog (screenshot below) prompts you to select either: (i) a free-flowing document design, or (ii) a form-based document design (in which components are positioned absolutely, as in a layout program).
In a free-flowing document design, document content is laid out to fit the output media object or viewer (paper or screen). Items in the document content can only be placed relative to each other, and not absolutely. This kind of design is suited for documents such as reports, articles, and books.

In a form-based document, a single Layout Container is created, in which design components can be positioned absolutely. The dimensions of the Layout Container are user-defined, and Layout Boxes can be positioned absolutely within the Layout Container and document content can be placed within individual Layout Boxes. If you wish the design of your SPS to replicate a specific form-based design, you can use an image of the original form as a blueprint image. The blueprint image can then be included as the background image of the Layout Container. The blueprint image is used to help you design your form; it will not be included in the output.

Selecting 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). Select the 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, do the following:

1. Click the **Switch to URL** command. This switches to the URL mode of the Open dialog (screenshot below).
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 (screenshot above). The Open button only becomes active at this point.

6. Click the Open button to load the file. The file you open appears in the main window.

Note: The Browse function is only available on servers which support WebDAV and on Microsoft SharePoint Servers. The supported protocols are FTP, HTTP, and HTTPS.

Note: To give you more control over the loading process, 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="Icon" /></td>
<td>Checked in. Available for check-out.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Checked out by another user. Not available for check-out.</td>
</tr>
<tr>
<td><img src="image" alt="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 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.

- See also
  - SPS and Sources
  - Schema Sources
  - Schema Tree sidebar
Open, Reload, Close, Close All

The Open (Ctrl+O) command allows you to open an existing SPS or PXF file. The familiar Open dialog of Windows systems is opened and allows you to select a file with an extension of .sps.

The Reload command reloads the SPS file from the file saved to disk. Any changes made since the file was last saved will be lost. The Working XML file will also be reloaded, enabling you to update the Working XML File if it has been changed externally.

The Close command closes the currently active SPS document. Note that while several files can be open, only one is active. The active document can also be closed by clicking the Close button at the top right of the Main Window. If you have unsaved changes in the document, you will be prompted to save these changes.

The Close All command closes all the open SPS documents. If you have unsaved changes in an open document, you will be prompted to save these changes.

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:
  - Checked in. Available for check-out.
  - Checked out by another user. Not available for check-out.
  - Checked out locally. Can be edited and checked-in.

- 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.

- See also
  - File | New
  - Main Window
Save Design, Save All

The **Save Design (Ctrl+S)** command saves the currently open document as an SPS file (with the file extension `.sps`).

The **Save All (Ctrl+Shift+S)** command saves all the open SPS documents.

---

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.

![Screenshot of File Open dialog]

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:

  - Checked in. Available for check-out.
  - Checked out by another user. Not available for check-out.
  - Checked out locally. Can be edited and checked-in.

- 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.

See also
- File | Close
Save As

The Save As command enables the design to be saved: (i) as an SPS file or (ii) as a PXF file (Portable XML Form file). Clicking the command pops up the Save Design dialog (screenshot below). Select the required format and click OK.

The SPS format is the standard Altova format for StyleVision designs. The PXF format is an Altova format that allows all files related to the design (schema files, XML files, images files, generated XSLT stylesheets, etc) to be embedded with the design. This format is very useful for transporting all the files required to open the design in Authentic View and/or to generate HTML, RTF, PDF and/or Word 2007 output based on the design.

Save as SPS
Selecting the SPS option causes the familiar Save As dialog of Windows systems to pop up. Saving works exactly as described for the Save Design command. The advantage of using the Save As command is that files that have already been saved with a filename can be saved with another filename.

Save as PXF
Selecting the PXF option causes the familiar Save As dialog of Windows systems to pop up. Saving works exactly as described for the Save Design command—with the additional step of selecting the files you wish to include in the PXF file. After you specify the PXF filename, the Configure PXF dialog (screenshot below) will appear, in which you can select/deselect the files you wish to embed.
In the Global Configuration pane of the Design-time Files tab, you can select/deselect the design-related source files to be embedded/omitted. You can additionally choose to embed XSLT files generated from the design. In the XSLT files pane, select the output formats for which you wish to generate and embed XSLT files. If an XSLT file is included in the PXF file and the PXF file is opened in the Authentic View of an Altova product, then the toolbar button to generate and view that output format is enabled in Authentic View (screenshot below).

In the Additional Files tab (screenshot below), you can specify any additional files you wish to include that are not design-time files. These could be, for example, image files referenced in the design by a URL generated with an XPath expression. In the screenshot below, the image file NewsItems.bmp located in the Images folder is selected for inclusion in the PXF file.
To include an additional file in the PXF file, click the **Add Document** button and then browse for the file you want. The Open dialog (in which you browse for the required file) opens the folder in which the SPS is located. Files from this folder or any descendant folder may be selected. After an additional file has been added to the PXF file, it and the folder structure leading to it are displayed. The screenshot above indicates that the additional file `NewsItems.bmp` is in a folder named `Images`, which is itself contained in the folder in which the SPS file is located.

If a file is selected from a folder located in any level above the folder containing the SPS file, an error is reported.

In the SPS design, any reference to an additional file must be made with a relative path and must use the folder structure shown in the **Additional Files** pane. For example, `NewsItems.bmp` in the screenshot above must be referenced with the relative path: `Images/NewsItems.bmp`.

**Note:** In order to save PXF files, the option `Embed Images for RTF and Word 2007+ (File | Properties | Images)` must be selected.
See also

- Save Design, Compatible To, All
- PXF File: Container for SPS and Related Files
Save Authentic XML Data, Save As

In Authentic View, you can edit the Working XML File or DB related to the SPS. The **Save Authentic XML Data** command saves these modifications to the Working XML File or DB. Alternatively to editing the XML file in StyleVision, you can edit an XML document or DB in the Authentic View of Altova XMLSpy or Altova Authentic Desktop.

The **Save Authentic XML Data As** command enables you to save the Authentic XML document as another file.

- **See also**
  - Authentic View
### Save Generated Files

The **Save Generated Files** command pops up a submenu which contains options for saving the following files (*screenshot below*). For perspective on how the generated files fit into the general usage procedure, see [Usage Procedure | Generated Files](#).

<table>
<thead>
<tr>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save Generated XSLT-HTML File...</td>
</tr>
<tr>
<td>Save Generated HTML File(s)...</td>
</tr>
<tr>
<td>Save Generated XSLT-RTF File...</td>
</tr>
<tr>
<td>Save Generated RTF File(s)...</td>
</tr>
<tr>
<td>Save Generated XSLT-FO File...</td>
</tr>
<tr>
<td>Save Generated FO File(s)...</td>
</tr>
<tr>
<td>Save Generated PDF File(s)...</td>
</tr>
<tr>
<td>Save Generated XSLT-Word 2007+ File...</td>
</tr>
<tr>
<td>Save Generated Word 2007+ File(s)...</td>
</tr>
<tr>
<td>Save Generated DB Schema...</td>
</tr>
<tr>
<td>Save Generated DB XML Data...</td>
</tr>
<tr>
<td>Save Generated User-Defined Schema...</td>
</tr>
<tr>
<td>Save Generated User-Defined XML Data...</td>
</tr>
</tbody>
</table>

---

### Save Generated XSLT-HTML File

The **Save Generated XSLT-HTML File** command generates an XSLT file for HTML output from your SPS. You can use this XSLT file subsequently to transform an XML document to HTML.

---

### Save Generated HTML File(s)

The **Save Generated HTML File(s)** command generates an HTML file or files. Multiple HTML files will be generated if *multiple document output* has been specified in the design. This operation requires two input files:

- The Working XML File assigned to the currently active SPS file. If no Working XML File has been assigned, the **Save Generated HTML File** command is disabled. For DB-based SPSs, the automatically generated non-editable XML file is the Working XML File (see [The DB Schema and DB XML files](#)); you do not assign a Working XML File.
- An XSLT file, which is automatically generated from the currently active SPS file.
Save Generated XSLT-RTF File
The Save Generated XSLT-RTF File command generates an XSLT file for RTF output from your SPS. You can use this XSLT file subsequently to transform an XML document to RTF.

Save Generated RTF File(s)
The Save Generated RTF File(s) command generates an RTF file. Multiple RTF files will be generated if multiple document output has been specified in the design. This operation requires two input files:
- The Working XML File assigned to the currently active SPS file. If no Working XML File has been assigned, the command is disabled. For DB-based SPSs, the automatically generated non-editable XML file is the Working XML File (see The DB Schema and DB XML files); you do not assign a Working XML File.
- An XSLT-for-RTF file, which is automatically generated from the currently active SPS file.

Save Generated XSLT-FO File
The XSL-FO File command generates an XSLT-for-FO (XSL-FO) file from your SPS. You can use this XSL-FO file subsequently to transform an XML document to an FO document.

Save Generated FO File
The Save Generated FO File command generates an FO file. This operation requires two input files:
- The Working XML File assigned to the currently active SPS file. If no Working XML File has been assigned, an error message saying the XML file could not be loaded is displayed. For DB-based SPSs, the automatically generated non-editable XML file is the Working XML File (see The DB Schema and DB XML files); you do not assign a Working XML File.
- An XSLT-for-FO file, which is generated from the currently active SPS file.

Save Generated PDF File(s)
The Save Generated PDF File(s) command generates a PDF file. Multiple PDF files will be generated if multiple document output has been specified in the design. It requires the following inputs and settings:
- A Working XML File. If no Working XML File has been assigned, an error message saying the XML file could not be loaded is displayed. For DB-based SPSs, the automatically generated non-editable XML file is the Working XML File (see The DB Schema and DB XML files); you do not have to assign a Working XML File.
- An XSLT-for-FO file, which is generated from the currently open SPS file.
• The path to the FO processor must be set in the XSL-FO tab of the Options dialog (Tools | Options).

---

**Save Generated XSLT-Word2007+ File**
The Save Generated XSLT-Word 2007+ File command generates an XSLT file for Word 2007+ output from your SPS. You can use this XSLT file subsequently to transform an XML document to a Word 2007+ document.

---

**Save Generated Word 2007+ File(s)**
The Save Generated Word 2007+ File(s) command generates a Word 2007+ file. Multiple Word 2007+ files will be generated if multiple document output has been specified in the design. This operation requires two input files:

- The Working XML File assigned to the currently active SPS file. If no Working XML File has been assigned, the command is disabled. For DB-based SPSs, the automatically generated non-editable XML file is the Working XML File (see The DB Schema and DB XML files); you do not assign a Working XML File.
- An XSLT-for-Word 2007+ file, which is automatically generated from the currently active SPS file.

---

**Save Generated DB Schema**
When you connect to a DB in order to create a DB-based SPS, StyleVision generates and loads a temporary XML Schema based on the DB structure. The Save Generated DB Schema command enables you to save this generated XML Schema. Note that for XML DBs, StyleVision does not generate a schema file; it uses a schema file from the DB or some other external file location. Consequently, this command is not enabled for XML DBs.

---

**Save Generated DB XML Data**
The Save Generated DB XML Data command generates and saves an XML file that contains data from the DB in an XML structure conformant with the structure of the XML Schema generated from the DB. If DB Filters have been defined in the StyleVision Power Stylesheet, these are applied to the data import. Note that for XML DBs, StyleVision does not generate an XML file, but uses XML data in the XML columns of the XML DB. Consequently, this command is not enabled for XML DBs.
**Save Generated User-Defined Schema**
This command is activated when the SPS involves a user-defined schema. The schema you create in the Schema Tree sidebar is saved as an XML Schema with the `.xsd` extension.

---

**Save Generated User-Defined XML Data**
The data in the imported HTML file that corresponds to the user-defined schema is saved as an XML file. The corresponding data are the nodes in the HTML document (in Design View) that have been created as XML Schema nodes.
Deploy to FlowForce

The **Deploy to FlowForce** command enables you to deploy a .transformation file to your Altova FlowForce Server. The .transformation file contains all the files and information required to carry out transformations as designed in the SPS. After the .transformation file has been deployed to the FlowForce Server, you can create jobs in Altova FlowForce that use the .transformation file to generate transformations according to triggers specified in the job definition. For information about creating FlowForce jobs, see the FlowForce documentation.

A .transformation file is generated from a Portable XML Format (PXF) file. So, the **Deploy to FlowForce** command can be used when a PXF file is active. (If an SPS file is active, the **Deploy to FlowForce** command will be active, but clicking it will prompt you to save the SPS file as a PXF file. To create a PXF file from an SPS file, use the **File | Save As** command and select PXF as the format to save as.)

Note the following points:

- When a PXF file is saved, an option is provided for including external files (such as image files) in it. If an external file is not included in the PXF file but is required for the transformation, then the external file must be saved on the FlowForce Server. Since the external files will be accessed from the working directory (specified in the FlowForce job definition), they should be placed relative to the working directory, in such a way that links originating in the working directory will correctly access them.
- Designs containing a database schema source are currently not supported, and cannot be deployed to FlowForce Server.
- When a FlowForce job requiring a StyleVision transformation is executed, the job is passed to StyleVision Server, and StyleVision Server will extract the contents of the PXF file to the working directory that was specified in the job's parameters. To ensure that there is no filename collision when this extraction occurs, there should be no file in the working directory that has the same name as a file contained in the PXF file.

Before running the **Deploy to FlowForce** command, make sure that Altova FlowForce Server and Altova StyleVision Server are correctly licensed and running. See the Altova FlowForce documentation for more information about setting up FlowForce Server. (StyleVision Server is packaged with FlowForce Server.)

---

The **Deploy command**

The **Deploy to FlowForce** command pops up the Deploy Transformation dialog (screenshot below).
In this dialog, you specify the following:

- The address and port number of the FlowForce Web Server (not the FlowForce Server), together with access details (user and password) for the FlowForce Server.
- The filename of the transformation file and the location on the FlowForce Server where it is to be saved. The filepath must start with a slash, which represents the root directory of the FlowForce Server.
- If changes have been made to the design since the file was last saved, the Save design changes before deploying check box will be enabled. Check the box if you wish to save these changes; otherwise uncheck the box.

On clicking OK, the .transformation file is deployed to the FlowForce Server at the location specified. If you have check the Open web browser to create new job check box (see screenshot above), the FlowForce Web Server interface is opened in a web browser, and the job created during the deployment step can be edited directly in the browser.

Note: For information about how to work with FlowForce Server, see the FlowForce documentation.
Web Design

The **Web Design** command rolls out a submenu containing the **Generate ASPX Web Application** command. This latter command generates all the files required to run an ASPX application, in the folder location you specify. A web browser will read the ASPX file that is the output document. C# code in this file will start a process whereby data in the source database or XML file will be transformed dynamically using an XSLT file in the ASPX package. The ASPX file (the output document of the transform process) will be updated with the latest data in the source database or XML file.

For more information, see [ASPX Interface for Web Applications](#).
Properties

The Properties command pops up the Properties dialog (screenshot below), in which you can set various properties for the active SPS.

Output

The following properties can be set in the Output tab:

- **Output Encoding**: In the Output Encoding pane you can select the encoding of your output documents. Changing the encoding in this dialog changes the encoding for the currently active SPS. You can also specify the default encoding for all subsequently created SPS documents; this is done in the Encoding tab of the Options dialog.

- **HTML output mode**: You can select whether an entire HTML document or only the child elements of the HTML body element are output. The child elements are output parallel to each other—that is, on the same level—and will contain all descendants recursively. As a result, the output documents can be fragments of HTML code. If multiple document
output is enabled, HTML fragments can be created in separate documents.

- **HTML output mode (DOCTYPE):** You can select whether an HTML5, HTML 4.01 Transitional document, or XHTML 1.0 Transitional document is generated for the HTML output. This setting can be changed at any time while creating or editing the SPS document.

- **Internet Explorer Compatibility and CSS support:** CSS support in versions of Internet Explorer (IE) prior to IE 6 was incomplete and in some respects incorrectly interpreted. CSS support was enhanced and corrected in IE 6, and further improved and extended in IE 7, IE 9, and higher.

  In an SPS, you can select the desired compatibility mode in the Properties dialog (screenshot above). You can select either IE 5, IE 7, or IE 9. (Note that for IE 9 compatibility to apply, IE 9 or higher must be installed.) The specified level of IE support is immediately available in Authentic View and HTML Preview. Note that new SPS documents are created with IE7 compatibility selected. SPS documents created in earlier versions of Altova StyleVision can be re-saved in the required Compatibility Mode (selected in the Properties dialog).

---

**XSD/XSLT**

In this tab, you can specify which XSD validator to use for XML validation and which XSLT version to use in the SPS.

StyleVision has both an XSD 1.0 validator and an XSD 1.1 validator. You can choose from among the following options:
• Use the XSD 1.1 validator if the XSD document's `@vc:minVersion` attribute is set to 1.1; otherwise use the XSD 1.0 validator.
• Always use the XSD 1.1 validator.
• Always use the XSD 1.0 validator.

Define the XSLT version for the active document in this tab. Checking the `use xsl:import-schema` declaration option causes the `xsl:import-schema` element of the XSLT 2.0 and 3.0 specifications to be included in the XSLT document generated by StyleVision. It is recommended that you select this option in order for datatypes to be read from the schema in the event that there is no `xsi:schemaLocation` attribute in the XML document.

Images
This tab provides settings for selecting whether images are linked to or embedded in the RTF and Word 2007+ output. Embedding is possible when XSLT 2.0 is used. This setting is made for each SPS individually. To embed images, do the following:

1. With the required SPS active, open the Properties dialog (File | Properties).
2. Check the Embed Images for RTF and Word 2007+ check box (default setting is checked). Note that images will only be embedded if XSLT 2.0 is set as the XSLT version of the active SPS.
3. Click OK and save the SPS. The setting is saved for the active SPS.

If the Embed Images check box is not checked, images will be linked according to the image file path specified in the images properties (select the image and select URL in the Properties entry helper). For information about how paths are resolved, see the section Image URLs.

This setting must be selected in order to save PXF files.

Note: The RTF format supports embedded images only for EMF, JPG, and PNG files.

Paths
Default paths for various files created by the SPS file and for paths saved in the SPS file are specified in settings in this tab.
The following defaults are set in the Paths tab:

- Whether preview files are created in the directory of the SPS file or the Working XML File of the main schema source.
- Where additionally generated files (image files, barcode image files, chart image files, etc.) are created.
- Whether file paths in the SPS are relative only when the target directory is in the directory tree of the SPS file, or relative even when the target directory is outside the directory tree of the SPS file.

---

**Resolution**

The resolution factor resolves how lengths that are specified as pixels in the design are to be converted to non-screen (print) units of length. You can select 72dpi, 96dpi, or 120dpi from the dropdown list of the combo box. The higher the dpi, the smaller will each pixel and the corresponding absolute length be. The conversion factors (for obtaining the number of points from the number of pixels) for each dpi are, respectively, 1.0, 0.75, and 0.6. For a complete explanation, see [Pixel Resolution](#).
**Authentic**
The following Authentic properties can be set:

- **Default paste mode for Authentic:** Specifies whether an Authentic View selection that has been saved to the clipboard will be pasted, by default, as XML or text in Authentic View. If it is pasted as XML, it will be pasted with markup (XML tags), if the copied selection contains markup. Otherwise the default paste mode copies the text content of nodes without markup. The default paste mode can be overridden in Authentic View by right-clicking at the insertion point and, in the context menu that appears, selecting the command *Paste As | XML* or *Paste As | Text*, as required. The default paste mode can be changed at any time while editing the SPS document.

You can also specify whether, when text is dragged and dropped in Authentic View, the user is given the option of selecting how to paste the text or whether the default paste mode is used. To give the user the choice of deciding the past mode, select the radio button *Ask on every drag-and-drop*; to use the default paste mode without consulting the user, select the radio button *Use default mode for paste*.

- **Authentic scripting:** These are options for scripts that are saved with an SPS: (i) Scripts can be enabled or not; (ii) Events in the scripts can be processed or can be disabled; (iii) Macros in the scripts can be run in debug mode from outside StyleVision (that is in the Authentic View of other Altova products) or can be disallowed. This feature is useful if you wish to test and debug a macro outside the StyleVision environment.

- **Maximum template call depth:** Specifies the maximum number of template calls that can be made recursively in the Authentic View output. If the number of template calls exceeds the number specified here, an error is returned. This setting can be used to prevent Authentic View from entering an unending loop.

---

**See also**
- **Tools | Options**
- **Pixel Resolution**
Print Preview, Print

The Print Preview command is enabled in Design View and Authentic View (Authentic View is supported in the Enterprise and Professional editions only). The Print Preview command opens a window containing a preview of the SPS design (when Design View is active) or of the Authentic View of the Working XML File when Authentic View is active). The preview will show the design with or without tags according to what is on screen.

You can do the following in the Print Preview window, via the toolbar commands at the top of the page (screenshot above) and the page navigation icons at the bottom of the page. The commands in the Print Preview toolbar are as follows, starting from the left.

- Print the page using the Print button.
- Set paper orientation to portrait or landscape.
- Set page properties by clicking the Page Setup button to get the Page Setup dialog.
- Toggle on/off the display and printout of headers and footers.
- Set the view so that either the page width or page height occupies, respectively, the full screen width or full screen height.
- Set how many pages are to fit within the screen.
- Change the zoom factor of the preview pages using the Zoom In and Zoom Out buttons or the combo box to select a zoom factor.

To navigate the pages of the preview, use the page navigation buttons at the bottom of the preview or by entering the page number in the Page text-box.

The Print command is enabled in the Authentic View and output preview tabs. It prints out the selected view of the Working XML File according to the page setup for that view. Note that the page setup for Authentic View can be edited in the Page Setup dialog, which you access via the Print 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.

See also
- File | Properties
Most Recently Used Files, Exit

The list of most recently used files, shows the file name and path information for the nine most recently used files. Clicking one of these entries, causes that file to be opened in a new tab in the Main Window.

To access these files using the keyboard, press ALT+F to open the File menu, and then the number of the file you wish to open; for example, pressing 1 will open the first file in the list, 2 the second file, and so on.

The Exit command is used to quit StyleVision. If you have an open file with unsaved changes, you will be prompted to save these changes.

See also
- Main Window
20.5 Edit Menu

The Edit menu contains commands that aid the editing of SPS and Authentic View documents. Besides the standard editing commands, such as Cut (Shift+Del or Ctrl+X), Copy (Ctrl+C), Paste (Ctrl+V), and Delete (Del), which are not described in this section, the following commands are available:

- **Undo, Redo, Select All**, to undo or restore your previous actions, and to select all content of the SPS.
- **Find, Find Next, Replace**, to find text in the SPS, Authentic View, and XSLT stylesheet previews, and to replace text in Authentic View.
- **Stylesheet Parameters**, to edit parameters declared globally for the SPS.
- **Collapse/Expand Markup**, to collapse and expand SPS design component tags.

Commands are also available via the context menu which appears when you right-click a component or right-click at a cursor insertion point. Additionally, some commands are available as keyboard shortcuts and/or toolbar icons. Note, however, that commands which are not applicable in a particular document view or at a given location are grayed out in the menu.

See also
- Toolbars
Undo, Redo, Select All

The **Undo (Ctrl+Z)** command enables you to undo an editing change. An unlimited number of Undo actions is supported. Every action can be undone and it is possible to undo one command after another till the first action that was made since the document was opened.

The **Redo (Ctrl+Y)** command allows you to redo any number of previously undone commands. By using the Undo and Redo commands, you can step backward and forward through the history of commands.

The **Select All** command selects the entire contents of the Design Document window.

- **See also**
  - Toolbars
Find, Find Next, Replace

The Find (Ctrl+F) command allows you to find words or fragments of words in the Design View, JavaScript Editor, Authentic View, and XSLT-for-HTML, XSLT-for-RTF, XSLT-for-FO, and XSLT-for-Word 2007+ stylesheets.

Design View, HTML Preview, and Authentic View
Clicking the Find command in Design View, HTML Preview, or Authentic View pops up the following dialog:

Note the following:

- In Design View, the static data is searched, but not node names.
- In Authentic View, the dynamic data (XML data) is searched, not text from the static (XSLT) input.
- To match the entry with whole words, check "Match whole word only". For example, an entry of soft will find only the whole word soft; it will not find, for example, the soft in software.
- To match the entry with fragments of words, leave the "Match whole word only" check box unchecked. Doing this would enable you, for example, to enter soft and software.
- To make the search case-insensitive, leave the "Match case" checkbox unchecked. This would enable you to find, say, Soft with an entry of soft.

XSLT-for-HTML, XSLT-for-RTF, XSLT-for-FO, XSLT-for-Word 2007+, and JavaScript Editor
Clicking the Find command in the XSLT-for-HTML, XSLT-for-RTF, XSLT-for-PDF, XSLT-for-Word 2007+, or JavaScript Editor tab pops up the following dialog:
The following points should be noted:

- To enter a regular expression as the search term, check the Regular expression check box. You can create a regular expression with the help of a menu that pops out when you click the right-pointing arrowhead near the search term entry field.

- To set restrictions on what part of the document to search, click the Advanced button. This makes more search options available (screenshot below):
Select the types of document content you wish to search by checking the appropriate check box.

---

**Find Next command**

The **Find Next (F3)** command repeats the last Find command to search for the next occurrence of the requested text. See **Find** for a description of how to use the search function.

---

**Replace (Ctrl+H)**

The **Replace** command is enabled in Design View, JavaScript Editor, and Authentic View (not supported in Basic edition) and enables you to search for a text string and replace it with another text string.

---

- **See also**
  - Toolbars
Stylesheet Parameters

The Stylesheet Parameters command enables you to declare and edit parameters and their default values. The command is available in both the Design Document view and the Authentic Editor View. When you click this command, the Edit Parameters dialog (shown below) pops up.

The following points should be noted:

- You can insert, append, edit and delete parameters for the entire stylesheet and for the DB Filters.
- Parameter names must begin with a letter, and can contain the characters A to Z, a to z, 0 to 9, and the underscore.
- The Edit Parameters dialog contains all the user-defined parameters in an SPS.
- Parameters can also be declared in the Design Overview sidebar.

See also

- Parameters
- DB Filters: Filtering DB Data
Collapse/Expand Markup

The **Collapse/Expand Markup** command is a toggle command, which collapses and expands the selected tag. It can be applied to any kind of tag: node, predefined format, SPS mechanism, etc. To collapse/expand a tag, double-click the tag; the end tag of an expanded tag may also be double-clicked to collapse that tag.

The screenshots below show how a series of tags are collapsed. Double-clicking a collapsed tag expands it.

Collapsing a tag can be useful for optimizing the display according to your editing needs.

- **See also**
  - Symbols
  - Template Filter
20.6 Project Menu

The **Project** menu (*screenshot below*) enables you to create, structure, and modify projects. You can quickly set up a project, specify files in the project, and organize files by file type into separate folders. A project is displayed graphically in the Project sidebar, from where files can be accessed for use in the SPS design.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Project</td>
<td>for creating a new project</td>
</tr>
<tr>
<td>Open Project</td>
<td>for opening an existing project</td>
</tr>
<tr>
<td>Reload Project</td>
<td>for refreshing a project in the Projects sidebar</td>
</tr>
<tr>
<td>Close Project</td>
<td>for closing a project in the Project sidebar</td>
</tr>
<tr>
<td>Save Project</td>
<td>for saving and naming a project</td>
</tr>
<tr>
<td>Add Files to Project</td>
<td>for adding files to a project in the Project sidebar</td>
</tr>
<tr>
<td>Add Global Resource to Project</td>
<td>for adding Altova Global Resources to a project in the Project sidebar</td>
</tr>
<tr>
<td>Add URL to Project</td>
<td>for adding a file via a URL to a project in the Project sidebar</td>
</tr>
<tr>
<td>Add Active File to Project</td>
<td>for adding the file currently active SPS file to a project</td>
</tr>
<tr>
<td>Add Active and Related Files to Project</td>
<td>for adding the currently active SPS file and its related files</td>
</tr>
<tr>
<td>Add Project Folder to Project</td>
<td>for adding folders to a project in the Project sidebar</td>
</tr>
<tr>
<td>Add External Folder to Project</td>
<td>for adding local folders to a project in the Project sidebar</td>
</tr>
<tr>
<td>Add External Web Folder to Project</td>
<td>for adding folders via URL to a project in the Project sidebar</td>
</tr>
</tbody>
</table>

The **Project** menu contains the following commands, which are explained in detail in the subsections of this section:
Recent projects
At the bottom of the Project menu, the file names of the nine most recently used projects are listed, thus allowing quick access to these files.

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 sidebar.

See also
- Projects in StyleVision
- Project sidebar
New Project, Open Project, Reload Project

The **New Project** command creates a new project. The new project replaces the previous project (if any) in the Projects sidebar. If the project you have been working on has unsaved changes, a prompt appears asking whether you wish to save changes to the project. Note that the **New Project** command only creates a new project without saving it; you have to explicitly save the project using the **Save Project** command.

The **Open Project** command opens an existing project and displays it in the Projects sidebar. If a project was previously open in the Projects sidebar, it is replaced by the opened project. If the previous project has unsaved changes, a prompt appears asking whether you wish to save changes to that project before it is replaced in the Projects sidebar.

The **Reload Project** command reloads the current project. This command is especially useful if you are working in a multi-user environment, where other users might make changes to the project.

---

**See also**

- [Projects in StyleVision](#)
- [Project sidebar](#)
Close Project, Save Project

The Close Project command closes the active project. If the project contains unsaved changes, a prompt appears asking whether you wish to save the project before closing it. A project with unsaved changes is indicated with an asterisk after the project name in the Project sidebar (screenshot below).

![Project Sidebar Screenshot](image)

The Save Project command saves the current project. Note that it is when a project is saved for the first time that it is named. A project can only be renamed outside StyleVision; for example, by using Windows File Explorer to locate and rename the file.

- See also
  - Projects in StyleVision
  - Project sidebar
Add Files / Global Resource / URL to Project

Add Files to Project
The Add Files to Project command adds files to the current project. The command pops up an Open dialog box, in which you select a single file or a group of files to add to the project. The file/s will be added to sub-folders within the project folder according to the file type extensions defined for each sub-folder. If the same file type extension has been defined for more than one folder, then a file with that file type extension will be added to the first folder (in the Projects sidebar) having that file type extension.

To add a file to a folder or sub-folder within the main project folder, right-click that folder, select the command Add Files, and then browse for the required file/s.

Add Global Resource to Project
The Add Global Resource to Project command pops up a dialog that lists global resources in the currently active Global Resources XML File and enables you to select one of these resources to add to the active project. Select the required global resource and click OK.

Add URL to Project
The 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. The command pops up the Add URL to Project dialog (screenshot below).

You can enter either a file URL (with or without the file:// protocol) or a server URL. For the server URL, enter your user name and password, then enter the server URL. Click Browse to connect to the server, then, from the list that appears in the Available Files display, click the file you wish to add to the project folder.
Note that URLs can also be added to folders and sub-folders of the main project folder. To do this, right-click the project folder and select the command **Add URL**. This pops up the Add URL dialog. Proceed as described above.

---

**Drag-and-drop**
A file can be dragged to another folder, but cannot be moved within the same folder (within which files are arranged alphabetically). Additionally, files can be dragged from Windows File Explorer to the Project window.

---

**Deleting Files, Resources, and URL**
To delete a file, Altova Resource, or URL, select that object in the Project sidebar, right-click, and, from the context menu, select **Delete**.

- **See also**
  - Projects in StyleVision
  - Project sidebar
Add Active (and Related) Files to Project

Add Active File to Project
The Add Active File to Project command adds the active SPS file to the current project. This file is added to the first folder defined for the .sps file type extension. If you wish to add not just the SPS but the related schema, Working XML, Template XML, CSS and image files, use the Add Active and Related Files to Project command (see below). To add the active file to a folder or sub-folder within the main project folder, right-click that folder, select the command Add Active File.

Add Active and Related Files to Project
The Add Active and Related Files to Project command adds the currently active SPS file as well as the related schema files, and, if any, the Working XML, Template XML, CSS and image files. Each file is added to the first folder defined for that particular file type extension. To add the active file and related files to a folder or sub-folder within the main project folder, right-click that folder, select the command Add Active and Related Files.

Deleting Files
To delete a file, select the file in the Project sidebar, right-click, and, from the context menu, select Delete.

See also
- Projects in StyleVision
- Project sidebar
Add Project and External Folders to Project

Add Project Folder to Project
The Add Project Folder to Project command adds a new folder to the current project. When you click the command, the Properties dialog (screenshot below) pops up, in which you enter the name and file type extensions for the folder (file type extensions are separated by a semi-colon). When a file having the file type extension defined for the folder is added to the project, the file will automatically be added to this folder. The newly added project folder is appended to the list of project folders in the Project sidebar.

![Properties dialog](screenshot)

To create a sub-folder of any given project folder, right-click the folder for which the sub-folder is required. In the context-menu that pops up, select Add Project Folder. In the Properties dialog, enter the folder name and the file type extensions for the folder.

Add External Folder to Project
The Add External Folder to Project command adds a new external folder to the current project. The command adds a local or network folder, with all its contents, to the current project. The added external folder can be expanded and collapsed. To add an external folder to a project folder as a sub-folder, right-click the project folder and, from the context menu, select the command Add External Folder.

Add External Web Folder to Project
The Add External Web Folder to Project command adds a new external folder via a URL to the current project. The added external folder can be expanded and collapsed. To add an external web folder to a project folder as a sub-folder, right-click the project folder and, from the context menu, select the command Add External Web Folder.

On clicking the command, the Add Web Folder dialog pops up (screenshot below). Do the following:

1. Click in the Server URL field to enter the server URL, and enter the login ID in the User and Password fields.
2. Click **Browse** to connect to the server and view the folders available there.

3. Click the folder you want to add to the project view. The **OK** button only becomes active once you do this. The folder name and server URL now appear in the File URL field.

4. Click **OK** to add the folder to the project.

5. Click the plus icon to view the folder contents.

6. To define the file types to display for the web folder, right-click, select **Properties** from the context menu, and enter the required file type extensions.
Drag-and-drop
In the Project window, a folder can be dragged to another folder or to another location within the same folder. Additionally, folders can be dragged from Windows File Explorer to the Project window.

Deleting Folders
To delete a folder or multiple folders, select the file in the Project sidebar, right-click, and, from the context menu, select Delete.

See also
- Projects in StyleVision
- Project sidebar
20.7 View Menu

The View menu (screenshot below) enables you to change the look of the GUI and to toggle on and off the display of GUI components. You can switch the display of individual toolbars, individual design sidebars, design filters, and the status bar on and off.

- See also
  - User Interface
  - Toolbars
Toolbars and Status Bar

Placing the cursor over the Toolbars item pops out a submenu (screenshot below), which enables you to turn on and off the display of the different toolbars.

- Formatting
- Table
- Authentic
- RichEdit
- Insert Design Element
- Design Filter
- Global Resources
- Standard Toolbar

When a toolbar is checked, it is displayed. In the screenshot above all the toolbars are displayed. To toggle on or off the display of a toolbar, click the appropriate toolbar. For a complete description of toolbars, see the section Reference | Toolbars.

Status Bar

The display of the Status Bar, which is located at the bottom of the application window, can be switched on or off by clicking the Status Bar toggle command.

- See also
  - Toolbars
  - Tools | Customize
Design Sidebars

The **View** menu contains toggle commands to switch the display of each sidebar on and off (screenshot below).

When a sidebar is toggled on (the command’s icon is framed) it is displayed in the GUI. Click a sidebar to set its display on or off, as required. This command is also used to make a hidden sidebar visible again. The display setting specified for a sidebar is View-specific: a setting made in a particular View (Design View, Authentic View, Output View, no document open) is retained for that particular View till changed.

- **See also**
  - User Interface | Design sidebars
Design Filter, Zoom

Design Filter
The Design Filter menu item rolls out a sub-menu containing commands that enable you to filter the templates that are displayed in Design View. This is useful if your design is very long or contains several templates. Using the Design Filter mechanism, you can specify what kinds of template to display. The following filter options are available:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![icon]</td>
<td>Show only one template</td>
<td>Shows the selected template only. Place the cursor in a template and click to show that template only.</td>
</tr>
<tr>
<td>![icon]</td>
<td>Show all template types</td>
<td>Shows all templates in the SPS (main, global, named, and layout).</td>
</tr>
<tr>
<td>![icon]</td>
<td>Show imported templates</td>
<td>Toggles the display of imported templates on and off.</td>
</tr>
<tr>
<td>![icon]</td>
<td>Show/Hide main template</td>
<td>Toggles the display of the main template on and off.</td>
</tr>
<tr>
<td>![icon]</td>
<td>Show/Hide global templates</td>
<td>Toggles the display of global templates on and off.</td>
</tr>
<tr>
<td>![icon]</td>
<td>Show/Hide Design Fragments</td>
<td>Toggles the display of Design Fragments on and off.</td>
</tr>
</tbody>
</table>

Note that these commands are also available as toolbar icons in the Design Filters toolbar.

Zoom
The Zoom command enables you to select a Zoom factor from the submenu that rolls out. You can also zoom in or out by changing the Zoom factor in the Zoom combo box (in the Standard toolbar), or by pressing the Ctrl key and scrolling with the mouse.

- See also
  - Toolbars | Design Filter
20.8 Insert Menu

The Insert menu provides commands enabling you to insert a variety of design components into the SPS. Some of these commands are available as toolbar icons. Additionally, Insert menu commands are also available via context menus which appear when, in the SPS design, you right-click a cursor insertion point. In the context menus, commands that are not available at that location in the SPS are disabled.

**Note:** Since the Insert commands are used for constructing the SPS, they are available in Design View only.

---

**See also**

- **User Interface**
- **Toolbars**

---

**See also:**

*Content Editing Procedures*, for a detailed description of the usage of various components that can be inserted using the Insert menu.

*Toolbars*, for a detailed description of individual toolbars.

*Enclose With Menu*, for equivalent commands that enclose a selection with a design component.
Contents

The Contents command inserts a (content) placeholder at the cursor location point. There (content) placeholder can be inserted within two types of node, element and attribute, and it indicates that all children of the current node will be processed.

- If the current node is an element node, the node's children element nodes and text nodes will be processed. For the processing of children element nodes, global templates will be used if these exist. Otherwise the built-in template rule for elements will be used. For the processing of text nodes, the built-in template rule for text nodes will be used, the effect of which is to output the text. Effectively, the built-in template rule for elements, outputs the text of all descendant text nodes. It is important to note that the values of attributes will not be output when the (content) placeholder is used—unless a global template is defined for the attribute's parent element or one of its ancestors and the attribute is explicitly output, using either the (content) placeholder or any other content-rendering component.

- If the current node is an attribute node, the built-in template rule for the attribute's child text node will be used. This template copies the text of the text node to the output, effectively outputting the attribute's value.

The (content) placeholder can also be inserted for a node by placing the cursor inside the node tags, right-clicking, and selecting Insert | Contents or by clicking the Insert Contents icon in the Insert Design Elements toolbar, and then clicking the location in the design where the element is to be inserted.

Styling the contents

The (content) placeholder can be formatted by selecting it and using a predefined format and/or properties in Styles sidebar. This formatting is visible in the design, and, in the output, it will be applied to the contents of the node.

Replacing contents

If another node from the schema tree is dropped into a node containing a (content) placeholder, then the existing (content) placeholder is replaced by the new node.

Deleting contents

The (content) placeholder can be deleted by selecting it and pressing the Delete key on the keyboard.

Note: You can create an empty template rule by deleting the (content) placeholder of a node. An empty template rule is useful if you wish to define that some node have no template applied to it, i.e. produce no output.
Rest of Contents

The Rest of Contents command inserts the (rest-of-contents) placeholder for that node. This placeholder represents the content of unused child nodes of the current node; it corresponds to the xsl:apply-templates rule of XSLT applied to the unused elements and text nodes of the current element. Note that templates are not applied for child attributes. The (rest-of-contents) placeholder can also be inserted for an element by placing the cursor inside the element tags, right-clicking, and selecting Insert Rest of Contents.

Use the (rest-of-contents) placeholder in situations where you wish to process one child element in a specific way and apply templates to its siblings. It is important to apply templates to siblings in order to avoid the possibility that the siblings are not processed. This enables you to reach elements lower down in the document hierarchy.

The (rest-of-contents) placeholder can be deleted by selecting it and pressing the Delete key on the keyboard.

See also
- Inserting XML Content as Text
- Output Structure
- Rest-of-Contents
- Contents
RichEdit

The RichEdit command inserts a RichEdit component at the cursor location. For the first RichEdit component, the RichEdit Configuration dialog (screenshot below) pops up. This RichEdit configuration is valid for all RichEdit components in the document. As a result, for RichEdit components inserted subsequently, the RichEdit Configuration dialog does not appear.

In the RichEdit Configuration dialog, enter the name of the styling element and its attribute that is to contain the RichEdit styling properties. You can also select the required element and attribute from the schema tree. Click the respective Select buttons to open the schema tree. When done, click OK. The RichEdit component is created (see screenshot below), and an uneditable RichEdit global template having the name of the styling element (Style in the screenshot below) is created in the design.
For more information about the RichEdit feature in context, see Text-Styling Flexibility in Authentic.

See also
- RichEdit toolbar
- RichEdit
- Authentic Menu
Form Controls

Mousing over the **Form Controls** command rolls out a submenu (*screenshot below*) containing commands to insert various form controls (**data-entry devices**).

How to create each of these form controls is described in the section *Using Data-Entry Devices*. After a form control has been created, its properties can be edited by selecting it and then editing the required property in the **Properties sidebar**.

Form controls can also be inserted in the design by right-clicking at the insertion point and selecting **Insert | Contents**, or by clicking the respective Form Control icon in the **Insert Design Elements toolbar**, and then clicking the location in the design where the element is to be inserted.

**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.

---

**See also**

- *Using Data-Entry Devices*
- *Properties*
DB Control

Mousing over the DB Controls command rolls out a submenu containing commands to insert controls in Authentic View that enable the Authentic View user to navigate the display of records in Authentic View and to query the DB. These control can be inserted in the design and will appear in the Authentic View document at the corresponding locations.

The list of commands is as follows.

- Navigation
- Navigation + Goto
- Query Button

For details about how these controls are created and what they do, see the section SPS Design Features for DB.

See also

- SPS Design Features for DB
- Working with Databases
**Auto-Calculation**

An **Auto-Calculation** uses an XPath expression to calculate a value. This value is displayed at the point where the Auto-Calculation is inserted. An Auto-Calculation can be inserted in the SPS as a text value, input field, or multiline input field. Place the cursor at the location where the Auto-Calculation is to be inserted, then either right-click or use the command in the **Insert** menu. When the cursor is placed over **Insert | Auto-Calculation**, a menu pops out (screenshot below), enabling you to choose how the Auto-Calculation should be inserted. Alternatively, you can use the Auto-Calculation icon in the **Insert Design Elements toolbar**.

<table>
<thead>
<tr>
<th>Value…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Field…</td>
</tr>
<tr>
<td>Multiline Input Field…</td>
</tr>
</tbody>
</table>

The value of the Auto-Calculation will be displayed accordingly in Authentic View and the output document.

---

**The XPath expression for the Auto-Calculation**

On selecting how the Auto-Calculation should be represented, the **Edit XPath Expression dialog** (screenshot below) pops up.

The context node for the expression being built is highlighted in the schema tree in the pane at extreme left. You can enter the XPath expression directly in the text box, or you can double click an item (in any of the three panes) to insert that item. Nodes inserted from the schema tree in the left-hand pane are inserted relative to the context node (if the Absolute XPath check box is unchecked) or as an absolute expression starting from the document node (if the Absolute XPath
check box is checked).

After completing the XPath expression, click OK to finish inserting the Auto-Calculation.

---

**Updating an XML node with an Auto-Calculation**

A node in an XML document can be updated with the result of an Auto-Calculation. How to do this is described in the section, [Updating Nodes with Auto-Calculations](#).

- **See also**
  - [Edit XPath Expression Dialog](#)
  - [Auto-Calculations](#)
**Date Picker**

The **Date Picker** command inserts a Date Picker at the current cursor position. It will be enabled only when the cursor is within an `xs:date` or `xs:dateTime` node and if the element has been created as `(contents)` or an input field.

- **See also**
  - Using the Date-Picker
  - Auto-add Date Picker
  - Working with Dates
  - Formatting Dates
Paragraph, Special Paragraph

The **Paragraph** command inserts an HTML paragraph `<p>` element around the selected component. A component is considered selected for this purpose when the entire node is selected (by clicking either of its tags) or when static text is selected. If the cursor is placed within static text, the paragraph element is inserted (start and end tags) at this point. A paragraph can also be inserted by using the **Insert Paragraph** icon in the `Insert Design Elements` toolbar.

The **Special Paragraph** command allows you to assign a predefined format to the selected node. The available predefined formats can also be selected from the combo box in the toolbar.

Each paragraph type has particular formatting features that can be used to advantage. Note that the `pre` format type enables carriage returns to be output as such instead of them being normalized to whitespace.

- **See also**
  - [Predefined Formats](#)
  - [Working with CSS Styles](#)
Barcode

The **Insert Barcode** command pops up the Insert Barcode dialog (*screenshot below*).

The **Insert Barcode** dialog allows you to specify various properties for the barcode such as `Type`, `Text`, `setModuleWidth`, `setModuleHeight`, `doQuietZone`, `setQuietZone`, `setMsgPosition`, `setPattern`, `setFontName`, `setFontSize`, `setChecksumMode`, `orientation`, `pixelDensity`, and `generateImageSettings`. After setting the values for the mandatory properties and any other properties you wish to set, click **OK**. The barcode you have specified (such as that for the ISBN shown below) will be inserted in the design. For detailed information about the various barcode properties, see the section, **Barcodes**.

![Barcode example]

Two properties, **Type** and **Text**, are mandatory; the others are optional and/or have appropriate default values. After entering values for the mandatory properties and any other properties you wish to set, click **OK**. The barcode you have specified (such as that for the ISBN shown below) will be inserted in the design. For detailed information about the various barcode properties, see the section, **Barcodes**.

**Important:** For barcodes to work, a Java Runtime Environment must be installed. This must be version 1.4 or later in a bit version that corresponds to the bit version of the StyleVision package installed on your system: 32-bit or 64-bit.

**See also**
- **Barcodes**
• SPS File: Content
Chart

The **Insert Chart** command pops up the Chart Configuration dialog (screenshot below), in which you can specify the settings for the chart and make the data selection that determines what data is represented in the chart.

For a detailed description of how to work with charts, see the section **Charts**.

- **See also**
  - Charts
  - Chart Settings
- Chart Data Selection: Simple
- Chart Data Selection: Flexible
Image

The Image command pops up the Insert Image dialog (see screenshots below), in which you can specify the image to insert. The Insert Image icon in the Insert Design Elements toolbar also pops up the Insert Image dialog.

The Insert Image dialog has four tabs, each of which provides a different way to specify the image location. These are:

- **Static**: for entering the image URI directly
- **Dynamic**: for obtaining the image URI from the XML document or generating it with an XPath expression
- **Static and dynamic**: for combining the static and dynamic methods
- **Inline data**: for selecting an image that is stored in an XML file as Base-16 or Base-64 encoded text

The tabs are described in detail below.

---

**Static**

The image URI is entered directly in the Address field (see screenshot below). In the screenshot below the image URI is: http://www.altova.com/pix/Marketing/logo.png.

![Insert Image Dialog](image.png)

You can specify whether the URI is absolute (Absolute check box checked) or relative (Absolute check box unchecked). If a relative URI is entered, it will be resolved relative to the SPS file location. To enter the (absolute or relative) URI automatically, click Browse and browse for the image file.

---

**Dynamic**

An XPath expression returns the image URI. In the screenshot below, the XPath expression is @deptlogo. This assumes that the image URI is stored in the deptlogo attribute of the context node. The context node is the node within which the image is being created.
Click the **Edit XPath** button to pop up the **XPath Expression Builder**. In the schema tree of the XPath Expression Builder, the context node will be highlighted.

If the SPS is DTD-based and uses **unparsed entities**, then, an unparsed entity that references the image URI can be used. First, check the **Treat as unparsed entity** checkbox. Then enter an XPath expression that selects the node containing the unparsed entity. For details of how to use unparsed entities, see [Unparsed Entity URIs](#).

---

**Static and Dynamic**

Use both the static and dynamic mechanisms together to generate the URI.

If the `deptname` attribute of the context node has a value of `Marketing`, then the image URI composed in the screenshot above will be: `http://www.altova.com/pix/Marketing/logo.png`. Note that you can use the **XPath Expression Builder** for the dynamic part.

---

**Inline data**

An image can be stored in an XML file as Base-16 or Base-64 encoded text. The XPath expression in the Insert Image dialog (see screenshot below) selects the node containing the
encoded text. The Encoding combo box specifies the encoding used in the source XML so that StyleVision can correctly read the encoded text. And the Image Format combo box indicates in what format the image file must be generated. (An image file is generated from the encoded text data, and this file is then used in the output document.)

The Image File Settings dialog (accessed by clicking the Image File Settings button) enables you to give a name for the image file that will be created. You can choose not to provide a name, in which case StyleVision will generate a name.

- See also
- Using Graphics
Horizontal Line

The **Horizontal Line** command inserts a horizontal line at the cursor insertion point. This command is not available when an SPS component is selected. To set properties for the horizontal line, select the line in the design, and in the Properties sidebar, select *line*, and specify values for properties in the *HTML* group (see screenshot below).

You can specify the following properties for the line: its *color*, *size* (thickness), *width* (in the design), *alignment*, and the *noshade* property.

- **See also**
  - Properties sidebar
  - Working with CSS Styles
Table

The **Insert Table** command pops up the Create Table dialog (*screenshot below*).

![Create Table Dialog](image)

According to whether you wish to create a static table or a dynamic table, select the appropriate button. How to proceed with each type of table is described in the section: [Static SPS Tables](#) and [Dynamic SPS Tables](#).

Note that tables can also be created by using the **Table** | **Insert Table** menu command and the **Insert Table** icon in the Insert Design Elements toolbar.

---

**See also**

- Creating Static Tables
- Creating Dynamic Tables
- CALS/HTML tables
Bullets and Numbering

The Bullets and Numbering command allows you to create a list, either static or dynamic. The list items of a static list are entered in the SPS, while those of dynamic lists are the values of sibling nodes in the XML document.

To create a list do the following:

1. Place the cursor at the location where you wish to insert the list and click the Bullets and Numbering command. This pops up a dialog asking whether you wish to create a static list or dynamic list (screenshot below).

![Static List vs Dynamic List Dialog](image)

If you click Static List, the Bullets and Numbering dialog described in Step 3 pops up. If you click Dynamic List, the XPath Selector dialog pops up (screenshot below).

2. In the XPath Selector dialog, notice that the XPath Context is the context of the insertion location, and that it cannot be changed in the dialog. Select the node that is to be created as the dynamic list. In the screenshot below, the context node is `n1:Department`, and the `n1:Person` node has been selected as the node to be created as a list. This means that the content of each `n1:Person` node will be created as an item in the list.
If you select the User-defined XPath option, then you can enter an XPath expression to select the node to be created as the dynamic table. Clicking OK pops up the Bullets and Numbering dialog described in the next step.

3. In the Bullets and Numbering dialog, select the kind of list you wish to create. You can choose from a bulleted list (with a bullet, circle, or square as the list item marker), or a numbered list. Clicking OK creates the list with the type of list item marker you selected.
**Note:** A static list can also be created by placing the cursor at the location where the list is to be created and then clicking the Bulleted List icon or Numbered List icon in the Insert Design Elements toolbar as required. A dynamic list can also be created by dragging a node from the Schema Tree into the design.

- **See also**
  - Creating Lists
  - Enclose With | Bullets and Numbering
Bookmark

The Bookmark command allows you to insert a bookmark (or anchor) anywhere in the SPS. A bookmark can be referenced by a Hyperlink.

To insert a bookmark, do the following:

1. Place the cursor at the location where you wish to create the bookmark.
2. Select Insert | Bookmark, or right-click and select Insert | Bookmark. The Insert Bookmark dialog appears.

3. In the Insert Bookmark dialog, select a tab according to whether the name of the bookmark should be static (Static tab), dynamically obtained from the XML document (Dynamic), or composed of both static and dynamic parts (Static and Dynamic). In the screenshot above a dynamic bookmark is created, which has a name that is a unique ID for each Name child of the context node.
4. Click OK. The bookmark is defined.

Note: Bookmarks are created at the location specified in the design. If that location is within an element that repeats, a bookmark is created within each instance of that repeating element. If a static name is given, then each bookmark will have the same name. Therefore, it is better in such cases (of repeating elements) to give a dynamic name, which can be, for example, the name of a child element of the context node (the element within which the bookmark is created). If the node selected for the dynamic name might have the same content across multiple instances, then the uniqueness of the bookmark name can be ensured by using the generate-id() function to generate the name (see screenshot above). To reference such a bookmark, the same ID can be generated as the href value of a hyperlink. In this case make sure you use the fragment-identifier # in front of the generate-id() function. The XPath expression would be: concat('#', generate-id(nodeXXX)).

You can edit the name of a bookmark after it has been created. Do this by right-clicking the bookmark and selecting the Edit Bookmark Name command from the context menu that appears. Alternatively, in the Properties sidebar, in the Bookmark group of properties for the bookmark, you can click the Edit button of the bookmark name attribute and make the required changes.

Deleting a bookmark
To delete a bookmark, select it in the design and press the **Delete** key.

- **See also**
  - Inserting Bookmarks
  - Defining Hyperlinks
Hyperlink

The Hyperlink command enables you to insert a link from any part of the output document (HTML, RTF, PDF, or Word 2007+) to an anchor within the output document or to an external document or document fragment. Note that links are created only in the output document; linking is not available in Authentic View.

To insert a hyperlink, do the following:

1. A hyperlink can be created around an existing design component or inserted at any point in the document (with the link text inserted subsequently). Select the SPS component or text fragment to be made into a hyperlink or place the cursor at the point where the link is to be inserted.
2. Click the Hyperlink icon in the toolbar, or select Insert | Hyperlink, or right-click and select Insert | Hyperlink (when no design component is selected) or Enclose With | Hyperlink (when a design component is selected). A hyperlink can also be inserted by using the Insert Hyperlink icon in the Insert Design Elements toolbar.
3. In the Insert Hyperlink dialog that appears, specify the document or document fragment you wish to link to. If you are linking to a document fragment (that is, to a bookmark within a document), remember to include the # symbol. The URI for the hyperlink is specified in one of the following forms:
   - As a static address (entered directly; you can select an HTML file via the Browse button, and a fragment in the current document via the Bookmark button). Examples would be: http://www.altova.com (static Web page URI); U:\documentation\index.html (via Browse button); or #top_of_page (via Bookmark button).
   - As a dynamic address (which comes from a node in the XML document; you specify the node). An example would be a node such as //otherdocs/doc1. If the name of a bookmark has been generated using the generate-id() function, then the href of the hyperlink should be generated using the same generate-id() function. For information, see Defining Hyperlinks.
   - As a combination of static and dynamic text for an address (you specify the static text and the XML document node). An example would be www.altova.com -- department/name -- #intropara.
4. Click OK. The hyperlink is created.

Note: When specifying the node for a dynamic hyperlink entry, you can enter the XPath expression as an absolute XPath expression by checking the Absolute Path check box. If this check box is not checked, the XPath expression will be relative to the context node, which is the node within which the hyperlink is being inserted.

Using unparsed entities

If you are using a DTD as your schema, then in the dynamic part of a hyperlink address, you can use the URI declared for an unparsed entity in the DTD. For details of how to use unparsed entities, see Using unparsed entity URIs.
**Editing a hyperlink**
You can edit the `href` of a hyperlink after it has been created. Do this by right-clicking the hyperlink and selecting the **Edit URL** command. Alternatively, in the Properties sidebar, in the `Link` group of properties for the link, you can click the **Edit** button of the URL attribute and make the required changes.

---

**Deleting a hyperlink**
To delete a hyperlink, select it in the design and press the **Delete** key.

- **See also**
  - Defining Hyperlinks
  - Unparsed Entity URIs
  - Inserting Bookmarks
Footnote

You can insert footnotes in a document by adding the Footnote component (Insert | Insert Footnote) at the location where you want the footnote number to be. Footnotes are available in paged media output (PDF, RTF, and Word 2007+ in the Enterprise Edition; and RTF in the Professional Edition).

Note the following points:

- The text of the footnote must be placed within the tags of the footnote component, and the footnote text can be formatted.
- In the output, the footnote number appears at the location where the footnote was added. The footnote text appears at the bottom of the page, together with the corresponding footnote number.
- In the output, footnote text will be formatted according to the formatting of the text within the footnote component in the design.
- In the output, footnotes are numbered automatically through to the end of the document.
- In the case of multiple output documents, numbering is re-started for each output document.

In the screenshot below, two footnote components (Insert | Insert Footnote) have been inserted. The footnote text has been placed within the tags of the component, and the text has been formatted.

The screenshots below show the output. The screenshot at left shows the complete page, while the screenshots at right show closeups of the footnote numbers (top) and footnote texts (bottom).
Note: Formatting of footnote numbers is not supported.
Condition, Output-Based Condition

The **Condition** command enables you to insert a condition at the cursor point or around the selection. A condition consists of one or more branches, with each branch containing a specific set of processing rules. In this way, different sets of processing rules can be specified for different branches. For example, if the content of a node is the string *Stop*, the branch can test this, and specify that the contents of the node be colored red; a second branch can test whether the contents of the node is the string *Go*, and, if yes, color the contents of the node green; a third branch can specify that if the contents of the node is neither the string *Stop* nor the string *Go*, the contents of the node should be colored black.

To insert a condition, do the following:

1. Place the cursor at the desired location in the design or select the component around which the condition is to be inserted.
2. Select the menu command **Insert | Condition** or right-click and select the context menu command **Insert | Condition**.
3. In the **Edit XPath Expression dialog** that pops up (screenshot below), enter the XPath expression.

![Edit XPath Expression dialog](image)

The context node for the expression being built is highlighted in the schema tree in the pane at extreme left. You can enter the XPath expression directly in the text box, or you can double click an item (in any of the three panes) to insert that item. Nodes inserted from the schema tree in the left-hand pane are inserted relative to the context node (if the Absolute XPath check box is unchecked) or as an absolute expression starting from the document node (if the Absolute XPath check box is checked).
4. Click **OK** to finish inserting the condition. The condition is created with one branch, the test for which is the XPath expression you entered.
**Insert Output-Based Condition**
This command inserts an output based-condition at the cursor location or around the selected component. Each branch of the condition represents a single output (Authentic View, RTF, PDF, Word 2007, or HTML). To determine which branch represents which output, mouseover the branch tag or check the XPath expression of the selected branch (in the Properties sidebar, in the Condition Branch entry, click the Edit button). If the output-based condition was created at a cursor insertion point, all branches will be empty and content will have to be inserted for each branch. If the output-based condition was created around a component, each branch will contain that component. For more details about output-based conditions, see [Output-Based Conditions](#). You can edit, move, and delete output-based conditions in the same way you would with a standard condition.

---

**Editing the XPath expressions of branches**
To edit the XPath expression of a branch, select the branch in Design View. Then, in the Properties sidebar, select `condition branch | when`. Click the Edit button for the XPath item. This pops up the Edit XPath Expression dialog ([screenshot above](#)), in which you can edit the expression. Click OK when done.

---

**Adding branches, changing the order of branches, and deleting branches**
To add new branches, change the order of branches, and delete branches, right-click the required branch and select the relevant item from the context menu.

- **See also**
  - Conditions
Template

The **Template** command inserts, at the cursor insertion point, an empty template for the schema tree node you select. Insert a template as follows.

1. Place the cursor in the design at the location where the template is to be inserted.
2. Click the **Insert | Template** command. This pops up the Insert Template dialog (screenshot below).

![Insert Template(s) for Schema Path](image)

3. The XPath Context field contains the context node of the cursor insertion point and will be the context node for the template when it is created. Select the node for which you wish to create the template. In the screenshot above the **strings** node is selected as the node for which the template is being created.
4. Click **OK** to finish.

An empty template for the selected node will be created (in the screenshot below, an empty template for the **strings** node has been created).

```
[]strings  []strings
```

**See also**
- Inserting XML Content as Text
- Output Structure
- Insert Design Elements
User-Defined Template

The User-Defined Template command inserts, at the cursor insertion point, an empty template that selects a node the user specifies in an XPath expression. Insert a user-defined template as follows.

1. Place the cursor in the design at the location where the template is to be inserted.
2. Click the Insert | User-Defined Template command. This pops up the Edit XPath Expression dialog.
3. Enter the XPath expression to select the node you want. There are a few points to note in this connection: (i) The XPath expression will be evaluated in the context of the node within which the user-defined template is being created; (ii) The XPath expression can select any node anywhere in the document as well as in another XML document.
4. After you have entered the XPath expression, click OK to finish.

An empty user-defined template for the targeted node will be created.

For more detailed information, see the section, SPS File: Contents | User-Defined Templates.

See also

- SPS File: Contents | User-Defined Templates
- Insert Design Elements
- Insert | Template
- Enclose with | User-Defined Template
Variable Template

A Variable Template is a template that targets a variable and, by default outputs its content. It is inserted with the Insert | Variable Template or Enclose with | Variable command, which inserts, at the cursor insertion point, a template for a variable defined in the SPS. The variable template (screenshot below) contains a content placeholder by default, and this serves to output the contents of the variable. You can insert additional content (static as well as dynamic) in the variable template as required, or modify it as you would any other template. A variable template is indicated with a dollar symbol in its start and end tags.

![Variable Template Screenshot]

To insert a variable template, do the following:

1. Place the cursor in the design at the location where the template is to be inserted.
2. Click the Insert | Variable Template command. This pops up the Insert Variable Template dialog (screenshot below).

![Insert Variable Template Dialog]

3. The dialog contains a list of all the user-declared parameters and variables defined in the SPS. Select the variable for which you wish to add a variable template.
4. Click OK to finish.

- See also
  - Insert | Template
  - SPS File: Contents | User-Defined Templates
Design Fragment

Mousing over the Design Fragment command rolls out a submenu containing all the Design Fragments currently in the design. Clicking a Design Fragment in the submenu inserts it at the cursor insertion point.

See also
- Design Fragments
Layout Container, Layout Box, Line

The Insert | Layout Container command enables a Layout Container to be inserted anywhere in the design. A Layout Box and a Line can be inserted in a Layout Container, and both these commands are enabled only when a Layout Container is selected.

Layout Containers, Layout Boxes, and Lines can also be inserted via the respective icons in the Insert Design Elements toolbar. To insert via the toolbar icons, you must first select the appropriate toolbar icon and then click in the design at the location where you wish to insert the layout item.

For a detailed description of Layout modules and how to insert and use them in the design, see the section Layout Modules.

- See also
  - Layout Modules
  - Toolbars | Insert Design Elements
Table of Contents

Mousing over the Table of Contents command rolls out a submenu containing commands to insert various commands relating to the creation of a Table of Contents (TOC) template, TOC bookmarks, and a design document structure for the TOC.

The list of commands is as follows. For the details of how to use them click on the respective links, which will take you to the section on how to use that particular TOC component.

- Insert Table of Contents
- TOC Bookmark
- TOC Bookmark (Wizard)
- TOC Reference
- TOC Reference | Entry Text / Leader / Page Reference
- Hierarchical Numbering
- Sequential Numbering
- Level
- Level Reference
- Template Serves as Level

Note: These commands are also available as commands in a context menu, depending on where you right click in the design.

See also
- Table of Contents (TOC)
New Document

The Insert New Document command inserts a New Document template (screenshot below) at the cursor insertion point.

![New Document Template](image)

The New Document template contains an empty Initial Document Section. Content can now be entered in the Initial Document Section. If desired, additional Document Sections can be appended to the Initial Document Section via the Insert | Insert Page / Column / Document Section command.

A New Document template creates a new document in the output. As a result, the output will consist of multiple output-documents.

For a detailed description of how to work with multiple output-documents, see the section, Multiple Document Output.

See also

- Multiple Document Output
- Inserting a New Document Template
- New Document Templates and Design Structure
- Document Properties and Styles
Page / Column / Document Section

With the Page / Column / Document Section command you can insert, for paged media output, a page break (HTML printouts, RTF, PDF, and Word 2007+ outputs) and page number (RTF, PDF, and Word 2007+ outputs). Such insertions are possible only at cursor insertion points.

New Page
Click New Page to insert a page break at the cursor insertion point. The page break is displayed as a dashed line across the whole of the Design window. In HTML output, while the page break has no effect in the browser view, a page break will be inserted when the browser view of the HTML file is printed out. In PDF, Word 2007+ and RTF output, a page break is inserted at the specified locations.

Page Number
Click Page | Number to insert the current page number in the RTF, PDF, and Word 2007+ outputs. The page number appears as a block (i.e. as a separate line) or as an inline (embedded in document text), depending on where in the document the page number has been inserted. For example, if the page number is inserted within a paragraph element, then the page number appears inline within the paragraph. If, on the other hand, the page number is inserted, say, between two elements, then it appears on a separate line by itself.

Page Total
Click Page | Total to insert the total number of pages in the PDF output. The page total can be inserted anywhere in the document design, including in headers and footers. It is particularly useful when numbering pages. For example, the page total can be inserted in a header design as follows: Page: (page number)/(page total). This would produce output in the form: Page: 1/25.

New Column
The number of columns that a page in a given section must have is specified in the page properties of that section. In the output, text will fill the columns on a multi-column page one by one. Text can however be forced into a new column by inserting a column break (new column) in the design. To insert a new column in a document, place the cursor at the location in the design where the new section is to be added and click the New Column command, which is also available via the context menu.
**New Document Section**
A document is made up of one initial section and, optionally, additional sections. Each section has its own page properties. To insert a new section in a document, place the cursor at the location in the design where the new section is to be added and click the **New Document Section** command, which is also available via the context menu.

---

**Deleting Page Breaks, Page Numbers, and Page Total**
To delete page breaks, page numbers, and page total, select the placeholder and click **Delete**.

- **See also**
  - File | Print
  - Designing Print Output
**XBRL Element**

The **Insert XBRL Element** command inserts any of the following XBRL elements as a template:

- **Context**: Creates an empty `xbrli:context` element template. The location where the template is inserted determines what `context` element is selected. The content of the template will have to be created.
- **Periods**: The three empty period templates match three period elements: `period`, `startDate`, and `endDate`. Note that `startDate` and `endDate` are children of the `period` element. There is no template for the `instant` element, which is also an allowed child of `period`; however, since `instant` can only occur as an only child of `period`, the built-in XSLT templates will select the `instant` template automatically if templates are applied (with the `contents` placeholder, for example).
- **Identifier**: Creates an empty template for the `identifier` element.
- **Unit**: Creates an empty template to match the `unit` element. You can create a global template for `xbrli:measure`, and use this locally within the Unit template to actually generate the monetary unit.
- **Footnote**: Creates a template that matches the `footnote` element. Footnotes have two parts: a reference and the actual footnote text.

For more information, see the section **XBRL Templates**.

---

**See also**
- **XBRL Templates**
- **SPS File and XBRL**
User-Defined Item

Mousing over the Insert | User-Defined Item command causes a sub-menu to roll out that contains commands to insert a User-Defined Element or a User-Defined XML Text Block. How to use these two components is described in the section SPS File: Content | User-Defined Elements, XML Text Blocks.

- See also
  - User-Defined Elements
  - Enclose with | User-Defined Element
  - User-Defined XML Text Block
  - User-Defined Templates
20.9 **Enclose With Menu**

The **Enclose with** menu provides commands enabling you to enclose a selection in the design with a variety of design components. Some of these commands are available as toolbar icons that enable you to insert the component in the design (equivalent commands are available in the **Insert menu**). Additionally, **Enclose with** menu commands are also available via context menus which appear when, in the SPS design, you right-click a selection. In the menus and context menus, commands that are not available at that location in the SPS are disabled.

**Note:** Since the **Enclose with** commands are used for constructing the SPS, they are available in Design View only.

- **See also**
  - Content Editing Procedures
  - Toolbars
Template

The Enclose with | Template command encloses the selected design component or text with a template for the schema tree node you select. Do this as follows.

1. Select the design component or text you wish to enclose with a template.
2. Click the Enclose with | Template command. This pops up the Schema Selector dialog (screenshot below).

3. The XPath Context field contains the context node of the selection and will be the context node of the template when it is created. Select the node for which you wish to create the template. In the screenshot above the n1:Name node is selected as the node for which the template is being created.
4. Click OK to finish.

A template for the selected node will be created around the selection.

See also
- Inserting XML Content as Text
- Output Structure
- Insert Design Elements
User-Defined Template

The *Enclose with | User-Defined Template* command encloses the selection with a template for a node the user specifies in an XPath expression. Insert a user-defined template as follows.

1. Select the component in the design that you wish to enclose with a user-defined template.
2. Click the *Enclose with | User-Defined Template* command. This pops up the *Edit XPath Expression* dialog.
3. Enter the XPath expression to select the node you want. There are a few points to note in this connection: (i) The XPath expression will be evaluated in the context of the node within which the user-defined template is being created; (ii) The XPath expression can select any node anywhere in the document as well as in another XML document.
4. After you have entered the XPath expression, click *OK* to finish.

A user-defined template for the targeted node will be created around the selection.

For more information, see the section, *SPS File: Structure | Templates and Design Fragments | Variable Templates*.

- **See also**
  - *SPS File: Contents | User-Defined Templates*
  - *Insert Design Elements*
  - *Insert | Template*
  - *Insert | User-Defined Template*
Variable Template

The **Enclose with | Variable Template** command encloses the selection with a template for a variable defined in the SPS design.

1. Select the component in the design that you wish to enclose with a variable template.
2. Click the **Enclose with | Variable Template** command. This pops up the **Enclose with Variable Template dialog**.
3. From the list in the dialog, select the variable for which you wish to create the template.
4. Click **OK** to finish.

A variable template will be created around the selection.

For more information, see the section, **SPS File: Structure | Templates and Design Fragments | Variable Templates**.

- See also
  - **Insert | Template**
  - **SPS File: Contents | User-Defined Templates**
Paragraph, Special Paragraph

The Paragraph command inserts an HTML paragraph \(<p>\) element around the selected component. A component is considered selected for this purpose when the entire node is selected (by clicking either of its tags) or when static text is selected. If the cursor is placed within static text, the paragraph element is inserted (start and end tags) at this point. A paragraph can also be inserted by using the Insert Paragraph icon in the Insert Design Elements toolbar.

The Special Paragraph command allows you to assign a predefined format to the selected node. The available predefined formats can also be selected from the combo box in the toolbar.

Each paragraph type has particular formatting features that can be used to advantage. Note that the \(\text{pre}\) format type enables carriage returns to be output as such instead of them being normalized to whitespace.

- See also
  - Predefined Formats
  - Working with CSS Styles
Bullets and Numbering

The Enclose with | Bullets and Numbering command creates a static list and list items around the selection. If the selection contains a CR-LF, carriage-return and/or linefeed (inserted by pressing the Enter key), then separate list items are created for each text fragment separated by a CR-LF.

When this command is selected, the Bullets and Numbering dialog (screenshot below) pops up.

Select the list item marker you want and click OK. A list is created. The number of list items in the list corresponds to the number of CR-LFs (carriage-returns and/or linefeeds) in the selection. You can add more list items to the list by pressing Enter.

Note: You can obtain the same results by selecting static content and then clicking the Bulleted List or Numbered List icons in the Insert Design Elements toolbar.

See also
- Creating Lists
- Enclose With | Bullets and Numbering
Bookmarks and Hyperlinks

The **Enclose with | Bookmark** and **Enclose With | Hyperlink** commands are enabled when some text or component in the SPS design is selected. These commands enable a bookmark and hyperlink, respectively, to be created around the selection. For more information about how bookmarks and hyperlinks work and how to create them, see the section [Advanced Features | Table of Contents, Referencing, Bookmarks](#).

- See also
  - Inserting Bookmarks
  - Defining Hyperlinks
Condition, Output-Based Condition

The **Condition** command enables you to insert a condition at the cursor point or around the selection. A condition consists of one or more branches, with each branch containing a specific set of processing rules. In this way, different sets of processing rules can be specified for different branches. For example, if the content of a node is the string *Stop*, the branch can test this, and specify that the contents of the node be colored red; a second branch can test whether the contents of the node is the string *Go*, and, if yes, color the contents of the node green; a third branch can specify that if the contents of the node is neither the string *Stop* nor the string *Go*, the contents of the node should be colored black.

To insert a condition, do the following:

1. Place the cursor at the desired location in the design or select the component around which the condition is to be inserted.
2. Select the menu command **Insert | Condition** or right-click and select the context menu command **Insert | Condition**.
3. In the **Edit XPath Expression dialog** that pops up (screenshot below), enter the XPath expression.

![Edit XPath Expression dialog](image)

The context node for the expression being built is highlighted in the schema tree in the pane at extreme left. You can enter the XPath expression directly in the text box, or you can double click an item (in any of the three panes) to insert that item. Nodes inserted from the schema tree in the left-hand pane are inserted relative to the context node (if the Absolute XPath check box is unchecked) or as an absolute expression starting from the document node (if the Absolute XPath check box is checked).

4. Click **OK** to finish inserting the condition. The condition is created with one branch, the test for which is the XPath expression you entered.
**Insert Output-Based Condition**
This command inserts an output based-condition at the cursor location or around the selected component. Each branch of the condition represents a single output (Authentic View, RTF, PDF, Word 2007, or HTML). To determine which branch represents which output, mouseover the branch tag or check the XPath expression of the selected branch (in the Properties sidebar, in the Condition Branch entry, click the Edit button). If the output-based condition was created at a cursor insertion point, all branches will be empty and content will have to be inserted for each branch. If the output-based condition was created around a component, each branch will contain that component. For more details about output-based conditions, see Output-Based Conditions. You can edit, move, and delete output-based conditions in the same way you would with a standard condition.

---

**Editing the XPath expressions of branches**
To edit the XPath expression of a branch, select the branch in Design View. Then, in the Properties sidebar, select condition branch | when. Click the Edit button for the XPath item. This pops up the Edit XPath Expression dialog (screenshot above), in which you can edit the expression. Click OK when done.

---

**Adding branches, changing the order of branches, and deleting branches**
To add new branches, change the order of branches, and delete branches, right-click the required branch and select the relevant item from the context menu.

- **See also**
  - Conditions
TOC Bookmarks and TOC Levels

When a component in the design is selected, it can be enclosed with one or more relevant Table of Contents (TOC) components. The list of TOC commands is as follows. For the details of how to use them click on the respective links, which will take you to the section on how to use that particular TOC component.

- TOC Bookmark
- TOC Bookmark (Wizard)
- Level
- Level Reference

Note: These commands are also available as commands in a context menu, depending on where you right click in the design.

- See also
  - Table of Contents (TOC)
New Document

The Enclose With New Document command encloses the current selection with a New Document template (screenshot below).

The New Document template contains an Initial Document Section that contains the design selection that was highlighted when the Enclose With New Document command was selected. In the screenshot above, the TOC design component was selected and enclosed with a New Document template. Content can now be entered in the Initial Document Section. If desired, additional Document Sections can be appended to the Initial Document Section via the Insert | Insert Page / Column / Document Section command.

A New Document template creates a new document in the output. As a result, the output will consist of multiple output-documents.

For a detailed description of how to work with multiple output-documents, see the section, Multiple Document Output.

See also

- Multiple Document Output
- Inserting a New Document Template
- New Document Templates and Design Structure
- Document Properties and Styles
User-Defined Element

The Enclose with | User-Defined Element command creates a User-Defined Element around the selection in the design. How to use user-defined elements is described in the section SPS File: Content | User-Defined Elements.

- See also
  - User-Defined Elements
  - Enclose with | User-Defined Element
  - User-Defined XML Text Block
  - User-Defined Templates
20.10 Table Menu

The Table menu provides commands enabling you to insert a static or dynamic table and to change the structure and properties of static and dynamic tables. You can edit table structure by appending, inserting, deleting, joining, and splitting rows and columns. Properties of the table as well as of individual columns, rows, and cells are defined using CSS styles and HTML properties for tables and its sub-components.

The Table commands are available in the Table menu (see list below) and as icons in the Table toolbar. The availability of various table commands depends on the current cursor position. A table can be inserted at any location in the SPS by clicking the Insert Table command. To edit the table structure, place the cursor in the appropriate cell, column, or row, and select the required editing command. To edit a formatting property, place the cursor in the appropriate cell, column, row, or table, and, in the Styles sidebar and/or Properties sidebar, define the required property for that table component.

The following commands are available in the Table menu:

- Insert Table, Delete Table
- Add Table Headers, Footers
- Append/Insert Row/Column
- Delete Row, Column
- Join Cell Left, Right, Below, Above
- Split Cell Horizontally, Vertically
- View Cell Bounds, Table Markup
- Table Properties
- Vertical Alignment of Cell Content

Headers and footers
When you create a dynamic table, you can specify whether you wish to include headers and/or footers. (Footers are allowed only when the table grows top–down.) You can create a header and footer in a static table by manually inserting a top and bottom row, respectively. The structures of headers and footers in both static and dynamic tables can be modified by splitting and joining cells.

Navigating in tables
Use the Tab and arrow keys to navigate the table cells.

Adding cell content
Any type of SPS component can be inserted as the content of a cell. The component should be formatted using the standard formatting tools.

See also
- Table toolbar
- Working with Tables
**Insert Table, Delete Table**

The **Insert Table** command inserts an empty table in the design tab. Selecting this command opens a dialog box in which you select whether you wish to create a static or dynamic table.

- If you choose to create a static table, a dialog prompts you for the size of the table (in terms of its rows and columns).
- If you choose to create a dynamic, the XPath Selector dialog pops up, in which you can select the node that is to be created as a dynamic table. On clicking **OK**, the Create Dynamic Table dialog pops up, in which you can select the child nodes you wish to display as the fields of each table item. For details, see Creating dynamic tables.

You can change the structure of a table subsequently by appending, inserting, and deleting rows and/or columns.

The **Delete Table** command deletes the static or dynamic table in which the cursor is.

- **See also**
  - Table toolbar
  - Working with Tables
Add Table Headers, Footers

Table headers can appear as a header row (above the table body) or as a header column (to the left of the table body, though markup-wise a header column might be placed inside the table body). Similarly, table footers can appear as a footer row (below the table body) or as a footer column (to the right of the table body, though markup-wise a footer might be placed inside the table body).

Note: In the HTML output since table headers are enclosed in \texttt{th} elements, they appear bold (because the bold formatting is inherent in the \texttt{th} element).

The Add Table Header and Add Table Footer commands add table headers and footers as columns and rows, as follows:

- Add Table Header Column: Adds a header column to the left of the table body.
- Add Table Footer Column: Adds a footer column to the right of the table body.
- Add Table Header Row: Adds a header row above the table body.
- Add Table Footer Row: Adds a footer row below the table body.

See also

- Table toolbar
- Working with Tables
Append/Insert Row/Column

The **Append Row** command appends a row to the static or dynamic table in which the cursor is.

The **Insert Row** command inserts a row above the row in which the cursor is. This command applies to both static and dynamic tables.

The **Append Column** command appends a column to the static or dynamic table in which the cursor is.

The **Insert Column** command inserts a column to the left of the column in which the cursor is. This command applies to both static and dynamic tables.

- **See also**
  - Table toolbar
  - Working with Tables
Delete Row, Column

The **Delete Row** command deletes the row in which the cursor is. This command applies to both static and dynamic tables.

The **Delete Column** command deletes the column in which the cursor is. This command applies to both static and dynamic tables.

- **See also**
  - Table toolbar
  - Working with Tables
Join Cell Left, Right, Below, Above

The **Join Cell Left** command joins the cell in which the cursor is to the adjacent cell on the left. The contents of both cells are concatenated in the new cell. All property values of the cell to the left are passed to the new cell. This command applies to both static and dynamic tables.

The **Join Cell Right** command joins the cell in which the cursor is to the cell on the right. The contents of both cells are concatenated in the new cell. All property values of the cell to the left are passed to the new cell. This command applies to both static and dynamic tables.

The **Join Cell Below** command joins the cell in which the cursor is to the cell below. The contents of both cells are concatenated in the new cell. All property values of the cell on the top are passed to the new cell. This command applies to both static and dynamic tables.

The **Join Cell Above** command joins the cell in which the cursor is to the cell above. The contents of both cells are concatenated in the new cell. All property values of the cell on top are passed to the new cell. This command applies to both static and dynamic tables.

See also

- **Table toolbar**
- **Working with Tables**
Split Cell Horizontally, Vertically

The Split Cell Horizontally command \( \text{Split Cell Horizontally} \) creates a new cell to the right of the cell in which the cursor is. The contents of the original cell stay in the original cell. All properties of the original cell are passed to the new cell. This command applies to both static and dynamic tables.

The Split Cell Vertically command \( \text{Split Cell Vertically} \) creates a new cell below the cell in which the cursor is. The contents of the original cell remain in the upper cell. All properties of the original cell are passed to the new cell. This command applies to both static and dynamic tables.

- See also
  - Table toolbar
  - Working with Tables
View Cell Bounds, Table Markup

The View Cell Bounds and View Table Markup commands display the boundaries of cells and table column and row markup, respectively. With these two options switched on, you can better understand the structure of the table. Switched off, however, you can visualize the table more accurately.

The View Cell Bounds command toggles the display of table boundaries (borders) on and off for tables that have a table border value of 0.

The View Table Markup command toggles the display of the blue column and row markers on and off.

See also

- Table toolbar
- Working with Tables
Table Properties

The Table Properties command is enabled when the cursor is placed inside a static or dynamic table. Clicking the command, pops up the Properties sidebar, with the Table component selected (screenshot below).

You can now edit the properties of the table. Click OK when done.

- See also
  - Table toolbar
  - Working with Tables
Edit CALS/HTML Tables

The **Edit CALS/HTML Tables** command enables data structures in the XML document that follow the CALS table model or HTML table model to be generated in the output as tables. The table markup in the output formats is derived directly from the XML document. However, additional table formatting styles can be added via the SPS.

Selecting this command pops up the Edit CALS/HTML Tables dialog (*screenshot below*).

For details about CALS/HTML tables, see the section **Tables**.

**See also**

- [Working with Tables](#)
- [Creating Static Tables](#)
- [Creating Dynamic Tables](#)
**Vertical Alignment of Cell Content**

Commands to set the vertical alignment of cell content are available as icons in the Table toolbar. Place the cursor anywhere in the cell, and click the required icon.

- **Vertically Align Top** vertically aligns cell content with the top of the cell.
- **Vertically Align Middle** vertically aligns cell content with the middle of the cell.
- **Vertically Align Bottom** vertically aligns cell content with the bottom of the cell.

---

**See also**

- [Table toolbar](#)
- [Working with Tables](#)
20.11 Authentic Menu

The Authentic menu contains commands that enable you to:

- Customize aspects of the Authentic View of an XML document that will be displayed using the SPS.
- Edit documents in the Authentic View preview of StyleVision.

The commands in the Authentic menu are listed below:

- Edit Authentic Scripts
- Custom Toolbar Buttons
- Check Macro References
- Auto-Add Date Picker
- Auto-Add DB Controls
- Reload, Validate XML
- Select New Row with XML Data for Editing
- XML Signature
- Define XML Entities
- Markup Commands
- (Dynamic Table) Row Commands

Each of these commands is described in detail in the sub-sections of this section.

See also

- Authentic View interface
- SPS and Authentic View
Edit Authentic Scripts

The Edit Authentic Scripts command pops up StyleVision's Scripting Editor (screenshot below), in which you can create Forms, Event, and Macros for use in Authentic View.

![Scripting Editor](image)

For an overview of how scripts can be used in Authentic View, see the section, Scripting for Authentic. For a description of how the Scripting Editor works, see the section, Scripting in the Programmers' Reference.

- See also
  - Scripting Editor
  - Scripting for Authentic
  - SPS and Authentic View
Custom Toolbar Buttons

Clicking the **Custom Toolbar Buttons** command pops up the Authentic Custom Buttons dialog (*screenshot below*), in which you can design a customized Authentic toolbar. After the Authentic toolbar has been saved with an SPS, the toolbar will appear in the Authentic View of Enterprise and Professional editions of Altova products whenever an XML file associated with this SPS is edited in Authentic View.

**Note:** Altova products that have an Authentic View window are: the Enterprise and Professional editions of XMLSpy, and StyleVision and the Enterprise edition of Authentic Desktop and Authentic Browser.

![Authentic Custom Buttons dialog](image)

Adding a button

To add a button or separator to the Authentic toolbar, click the **Add** button at the top right of the dialog. This pops up a menu in which you can select what you wish to add: (i) a text state icon, (ii) a custom button to execute a macro, or (iii) a separator line in the toolbar to separate groups of buttons. **Text state icons** and **custom buttons** are described in detail below.

Moving a button and deleting a button

To move a button or deleting a button in the toolbar, select it and drag it to the new
position. To delete a button or separator, select it and click the **Delete** button at the top right of the dialog.

---

**Text state icons**
A text state icon defines an icon for a global element. When the Authentic View user selects text in the Authentic View document and clicks a text state icon, then the element that the icon defines is inserted around the selected text. Text state icons are intended for elements that provide inline formatting, such as bold and italic formatting.

To add a text state icon to the Authentic toolbar, click the **Add** button at the top right of the Authentic Custom Buttons dialog (see screenshot below) and select **Add Text State Icon**. Enter the name of the XML element for which the text state icon is being created, then browse for a bitmap image file for the button or enter a text for the button. You can optionally enter a tooltip as a guide for the Authentic View user when he or she mouses over the text state icon. Click **OK** to add the button to the Authentic toolbar.

![Authentic Custom Buttons](image)

The screenshot above shows the text state icon for the **bold** element selected. This text state icon uses an image named **bold.bmp**. Text for the button can be entered as a fallback. For the text state icon defined in the screenshot above, if the image cannot be found, then the text **Bold** will be used as the button text.
Custom buttons
When an Authentic View user clicks a custom button in the Authentic toolbar, a macro is executed. In the SPS design you can create a custom button and specify what macro it will trigger.

To add a custom button to the Authentic toolbar, click the Add button at the top right of the Authentic Custom Buttons dialog (see screenshot below) and select Add Custom Toolbar Button.

Custom buttons take the following parameters:

- The location of an image for the button (in the Bitmap field) or text for the button (in the Text field).
- In the Macro Name combo box select a macro from the dropdown list. The macros listed here are those that have been saved with the SPS. When you click the New button, the Scripting Editor of StyleVision opens in its own window, enabling you to quickly and easily create a macro and save it with the SPS. Clicking the Edit button opens the selected macro for editing in the Scripting Editor.
- You can optionally enter a tooltip as a guide for the Authentic View user when he or she mouses over the custom button.
- In the Identifier field enter a text string that will be used as the identifier of the custom button. This identifier can then be used in scripting code.

The screenshot above shows custom button for the ChangeAddressToUS macro. This custom button uses an image named US.bmp. Text for the button can be entered as a fallback. A tooltip
has been entered and the custom button has the identifier Address US. These examples are from the file ToolbarButtons.sps is in the Authentic\Scripting folder of the Examples project in the Project window.

- See also
  - Scripting Editor
  - Scripting for Authentic
  - SPS and Authentic View
Check Macro References

The Check Macro References command checks that references from toolbar buttons and scripts to macros are correct. If any incorrect reference is found an error message is displayed.

See also
- Scripting Editor
- Scripting for Authentic
- SPS and Authentic View
Auto-Add Date Picker

This is a toggle command that switches the Auto-Add Date Picker feature on and off. When the Auto-Add Date Picker feature is ON, any \texttt{xs:date} or \texttt{xs:dateTime} datatype element that is created as contents or as an input field will have the Date Picker automatically inserted within the element tags and after the \texttt{contents} placeholder or input field.

- **See also**
  - Using the Date-Picker
  - Insert Date Picker
  - Working with Dates
  - Formatting Dates
**Auto-Add DB Controls**

This is a toggle command that switches the Auto-Add DB Controls feature on and off.

When the Auto-Add DB Controls is on, then, whenever a DB table element is dropped into the design, the DB Controls panel (shown below) is inserted immediately before the Row child element of that DB table element.

The DB Controls panel enables the Authentic View user to navigate the rows of the DB table in Authentic View. The first (leftmost) button navigates to the first record; the second button navigates to the previous record; the third button is the Goto button; it pops up a dialog (screenshot below) that prompts you for the number of the record to which you wish to go; the fourth button navigates to the next record; and the fifth button navigates to the last record.

When the Auto-Add DB Controls toggle is turned off, the DB Controls panel is not inserted when a DB table is dropped into the Design document.

**Note:** You can manually insert navigation buttons by placing the cursor anywhere between the start and end tags of the DB table and selecting the required option from the **Insert | DB Controls** submenu. Note that in this submenu the DB Controls panel can be inserted as the four navigation buttons or as the four navigation buttons plus the button that calls the Goto Record dialog.
Reload Authentic View, Validate XML

The **Reload** command reloads the Authentic XML data file. This can be useful if the file has been modified outside StyleVision, especially by another user working from another machine.

The **Validate XML (F8)** command checks the validity of the XML file against the associated schema. Whether StyleVision's XSD 1.0 or XSD 1.1 validator is used can be specified in the Properties dialog. Any additional validation requirement that you have entered for individual nodes (Properties sidebar: Additional validation in the Authentic group of properties) is also checked. The result of the validation check is displayed in a pop-up message box.

- **See also**
  - Authentic View interface
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.

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.
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.

![XML Signature dialog](image)

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).

![XML Signature dialog](image)
Authentication information is either the signing certificate's key information or the signing password. The SPS designer can specify, in the XML Signature Settings, that the certificate's key information be saved in the signature, or that the password be 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 (password not saved) or SPS file (certificate's key information not saved). 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.

![Signature Password](screenshot)

To verify this signature a password is required.

The password must be at least 5 and at most 16 characters long.

- **See also**
  - XML Signatures
Define XML Entities

The **Define XML Entities** command is available only in Authentic View. With the **Define Entities** command in Authentic View, you can define entities that you want to add to your XML document. After an entity has been defined, it can be inserted in the XML document by right-clicking at the location where you wish to insert the entity, and, from the context menu that pops up, selecting **Insert Entity**, and then the name of the entity to be inserted.

An entity that you define with this command can be any of three types:

- **Internal parsed entity.** The value of the entity is a text string that usually occurs frequently in the document. Using an entity ensures that all occurrences are expanded to the value defined here.
- **External parsed entity.** This is an external XML file that will replace each occurrence of the entity. The value of the entity is the URI of the external XML file.
- **External unparsed entity.** This is an external resource that will be called when the entity is processed. The value of the entity is the URI of the external resource.

Clicking the command, pops up the Define Entities dialog (*screenshot below*).

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value/Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>nano_dc</td>
<td>Internal</td>
<td>Nanonull, Inc</td>
</tr>
<tr>
<td>nano_eu</td>
<td>Internal</td>
<td>Nanonull Europe, A/S</td>
</tr>
<tr>
<td>nano_na</td>
<td>Internal</td>
<td>Nanonull Partners, Inc</td>
</tr>
<tr>
<td>website</td>
<td>Internal</td>
<td><a href="http://www.nanonull.com/">http://www.nanonull.com/</a></td>
</tr>
<tr>
<td>branches</td>
<td>SYSTEM</td>
<td>branches.xml</td>
</tr>
<tr>
<td>logo</td>
<td>SYSTEM</td>
<td>nanonull.gif</td>
</tr>
</tbody>
</table>

For a description of how to use this dialog, see Define Entities in the Authentic View documentation.

- **See also**
  - **Unparsed Entity URIs**
View Markup

The View Markup command has a submenu with options to control markup in the Authentic XML document. With the four markup commands (screenshot below), you can select between the Hide Markup and the various Show Markup modes.

The markup refers to how the various node tags are displayed in Authentic View. These are mutually exclusive options, and one option must be selected at any given time. With Hide Markup selected node tags are not displayed. Small Markup shows opening and closing tags without node names. Large Markup shows opening and closing node tags with their respective node names. Mixed Markup refers to the markup specified in the Authentic Node Properties of individual nodes. Since the default markup property for individual nodes is Hide Markup, no markup (either small or large) will be displayed—unless you have specified (as a node property) small or large markup for some node/s.

See also

- Authentic Node Properties
- Working with Tables
- Authentic View interface
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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Toggle Bold</td>
</tr>
<tr>
<td>I</td>
<td>Toggle Italic</td>
</tr>
<tr>
<td>U</td>
<td>Toggle Underline</td>
</tr>
<tr>
<td>S</td>
<td>Toggle Strikethrough</td>
</tr>
<tr>
<td>F</td>
<td>Foreground Color</td>
</tr>
<tr>
<td>Bc</td>
<td>Background Color</td>
</tr>
<tr>
<td>A</td>
<td>Align Left</td>
</tr>
<tr>
<td>C</td>
<td>Center</td>
</tr>
<tr>
<td>R</td>
<td>Align Right</td>
</tr>
</tbody>
</table>

The text-styling properties of the RichEdit menu will be applied to the selected text when a RichEdit command is clicked. The Authentic View user can specify the font, font-weight, font-style, font-decoration, font-size, color, background color and alignment of the selected text.

For more information about the RichEdit feature in context, see Text-Styling Flexibility in Authentic.

See also
- RichEdit toolbar
- RichEdit
- Authentic Menu
(Dynamic Table) Row Commands

The (Dynamic Table) Row commands are enabled in Authentic View when the cursor is placed inside the row of a dynamic table. They enable you to manipulate the rows of a dynamic table. You can append, insert, duplicate, and delete rows, and you can move the selected row up and down relative to the other rows of the table. Since a row in a dynamic table represents a fixed data structure, the Authentic View user will be manipulating units of a data structure in the context of the data structure represented by the dynamic table.

A row is selected by placing the cursor inside it. An empty row can then be inserted (before) or appended (after) the selected row. A row can be duplicated, in the sense that a copy of the row plus its content is created after the selected. A row can also be moved up or down relative to adjacent rows.

See also
- Table toolbar
- Working with Tables
- Authentic View interface
20.12 Database Menu

The Database menu contains commands to query the connected database and to edit and clear the filters applied to the connected database.

- **Query Database** starts up the [Connect to Database](#) process and opens the Database Query window.
- **Edit DB Filter, Clear DB Filter**, to access the Edit DB Filters dialog and clear DB Filters, respectively.

**See also**
- DBs and StyleVision
- Connecting to a Database
- Query Database
- Datatype Conversions: DB to XSD
Query Database

The **Query Database** command pops up the [Database Query window](#) (screenshot below), via which you can connect to a database and query it. How to use the Database Query window is explained in the section [Query Database](#).

The Database Query window is toggled on and off by clicking the **Query Database** command.

- **See also**
  - [DBs and StyleVision](#)
  - [Connecting to a Database](#)
  - [Query Database](#)
  - [Datatype Conversions: DB to XSD](#)
Edit DB Filter, Clear DB Filter

The **Edit DB Filter** command allows you to create and edit a filter for a database table (a DB Filter). A DB Filter determines what data from the selected database table is imported and displayed. A DB Filter consists of one or more criteria. When you specify criteria, you use an expression, which is a combination of operators (= or >) and values (text or numbers). Additionally, criteria can be joined by the logical operators **AND** or **OR**.

To create or edit a DB Filter, do the following:

1. Select the top-level data table element for which you wish to create or edit a DB Filter. Do this by clicking either the element tag in Design View or the element name in the schema tree.
2. Select **Database | Edit DB Filter** or click the toolbar icon for the command. This pops up the Edit Database Filters dialog.

3. To add criteria use the **Append AND** and **Append OR** buttons. To move a criterion up or down, use the arrow buttons. To delete a criterion, use the Delete button.
4. Specify the criteria for the DB Filter. Each criterion consists of three parts: **Field Name** + **Operator** + **Value**. The options for Field Names and Operators are available in combo boxes. The value of the expression must be keyed in, and may be a parameter (indicated by a preceding $ character).

**Clear DB Filter command**

The **Clear DB Filter** command deletes the filter after asking for and receiving a confirmation from you.

---

*See also*

- **DBs and StyleVision**
- Connecting to a Database
- Query Database
- Datatype Conversions: DB to XSD
20.13 Properties Menu

The Properties menu contains commands that enable you to insert lists and define datatype formats for the input formatting feature. The description of the commands is organized into the following sub-sections:

- Bullets and Numbering command, to insert lists.
- Predefined Format Strings command, to define numeric datatype formats for a given SPS.

See also
- Properties sidebar
Edit Bullets and Numbering

The Edit Bullets and Numbering command enables you to insert a list at the cursor location. Clicking the command pops up the Bullets and Numbering dialog (screenshot below), in which you can select the list style; in the case of a numbered list, the initial number can also be specified.

See also
- Creating Lists
Predefined Value Formatting Strings

Any (content) placeholder, input field, or Auto-Calculation which is of a numeric, date, time, dateTime or duration datatype can be assigned a custom format with the Value Formatting dialog. In the Value Formatting dialog, you can either create a format directly or select from a drop-down list of predefined formats.

The predefined formats that are available in the dropdown list are of two types:

- Predefined formats that have been delivered with StyleVision, and
- Predefined formats that the user creates with the Predefined Value Formatting Strings command (this command). When a user creates predefined value formats, these are created for the currently open SPS file—not for the entire application. After the user creates predefined value formats, the SPS file must be saved in order for the formats to be available when the file is next opened.

Creating a predefined value formatting string

A predefined value format string is specific to a datatype. To create a predefined value formatting string, do the following:

1. Click Properties | Predefined Value Formatting Strings. The following dialog appears:

   ![Predefined Value Formatting Strings]

2. Select a datatype from the drop-down list in the combo box, and then click the Append or Insert icon as required. This pops up the Edit Format String dialog:

   ![Edit Format String]

   OK    Cancel

   Select a datatype from the drop-down list in the combo box, and then click the Append or Insert icon as required. This pops up the Edit Format String dialog:
If you click the down arrow of the combo box, a drop-down list with the StyleVision-supplied predefined formats for that datatype is displayed (shown in the screenshot below).

You can either select a format from the list and modify it, or you can enter a format directly into the input field. The syntax for defining a format is explained in the section, Value Formatting. If you need help with the syntax, use the Insert Field and Field Options buttons.

3. After you have defined a format, click OK and save the SPS file. The formatting string is added to the list of predefined formats for that datatype, and it will appear as an option in the Value Formatting dialog (of the current SPS file) when the selected element is of the corresponding datatype.

Note:

- You can add as many custom format strings for different datatypes as you want.
- The sequential order of format strings in the Predefined Format Strings dialog determines the order in which these format strings appear in the Value Formatting dialog. The customized format strings appear above the supplied predefined formats.
- To edit a custom format string, double-click the entry in the Predefined Format Strings dialog.
- To delete a custom format string, select it, and click the Delete icon in the Predefined Value Formatting Strings dialog.

See also

- Value Formatting (Formatting Numeric Datatypes)
20.14 Tools Menu

The Tools menu contains the spell-check command and commands that enable you to customize StyleVision. Among other options, you can customize the toolbar, and specify spell-checking and FO processor options.

The description of the Tools menu commands is organized into the following sub-sections:

- Spelling
- Spelling Options
- Global Resources
- Active Configuration
- Customize
- Options

See also
- Setting up StyleVision
Spelling

The **Spelling** command runs a spelling check on the SPS (in Design View) or the document in Authentic View, depending on which is active. You can use what language to use from the spellchecker's built-in language dictionaries (*see note below*).

**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.

On clicking this command, the dialog shown below appears. Words that are not present in the selected dictionary are displayed, in document order and one at a time, in the Not in Dictionary field of the dialog and highlighted in the Design Document.

You can then select an entry from the list in the Suggestions pane and click **Change** or **Change All** to change the highlighted instance of this spelling or all its instances, respectively. (Double-clicking a word in the Suggestions list causes it to replace the unknown word.) Alternatively, you can ignore *this instance* of the unknown word (**Ignore Once**); or ignore *all instances* of this unknown word (**Ignore All**); or add this unknown word to the user dictionary (**Add to Dictionary**). Adding the unknown word to the dictionary causes the spell-checker to treat the word as correct and to pass on to the next word not found in the dictionary. You can recheck the document from the beginning (**Recheck Document**) or close the dialog (**Close**) at any time.

The **Options** button opens the **Spelling Options** dialog, in which you can specify options for the spelling check.
See also

- Spelling Options
Spelling Options

The Spelling options command opens a dialog box (shown below) in which you specify options for the spelling check.

![Spelling Options dialog box]

- **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:

On Windows 7 and Windows 8: C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons
On Windows XP: C:\Documents and Settings\All Users\Application Data\Altova
\SharedBetweenVersions\SpellChecker\Lexicons

Within the Lexicons folder, different language dictionaries are each stored in different folder: <language name>\<dictionary files>. For example, on a Windows 7 or Windows 8 machine, 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 http://wiki.services.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:

On Windows 7 and Windows 8:  C:\Users<user>\Documents\Altova\SpellChecker\Lexicons\user.dic
On Windows XP:  C:\Documents and Settings<user>\My Documents\Altova\SpellChecker\Lexicons\user.dic
- See also
  - Spelling
Global Resources

The Global Resources command pops up the Altova Global Resources dialog (screenshot below), in which you can:

- Specify the Altova 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 (View | Toolbars | Global Resources).

See also
- Altova Global Resources
- Toolbars and Status Bar
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 (View | Toolbars | Global Resources).

- See also
  - Altova Global Resources
  - Toolbars and Status Bar
Customize

The customize command lets you customize StyleVision to suit your personal needs.

Commands tab

The Commands tab of the Customize dialog allows you to place individual commands in the menu bar and the toolbar.

To add a command to the menu bar or toolbar, select the command in the Commands pane of the Commands tab, and drag it to the menu bar or toolbar. When the cursor is placed over a valid position an I-beam appears, and the command can be dropped at this location. If the location is invalid, a check mark appears. When you drop the command it is created as an icon if the command already has an associated icon; otherwise the command is created as text. After adding a command to the menu bar or toolbar, you can edit its appearance by right-clicking it and then selecting the required action.

To delete a menu bar or toolbar item, with the Customize dialog open, right-click the item to be deleted, and select Delete.

Note:

- The customization described above applies to the application, and applies whether a document is open in StyleVision or not.
- To reset menus and toolbars to the state they were in when StyleVision was installed, go to the Toolbars tab and click the appropriate Reset button.
Toolbars tab
The Toolbars tab allows you to activate or deactivate specific toolbars, to show text labels for toolbar items, and to reset the menu bar and toolbars to their installation state.

The StyleVision interface displays a fixed menu bar and several optional toolbars (Authentic, Design Filter, Format, Standard, Table, and Table of Contents).

Each toolbar can be divided into groups of commands. Commands can be added to a toolbar via the Commands tab. A toolbar can be dragged from its docked position to any location on the screen. Double-clicking a toolbar's (maximized or minimized) title bar docks and undocks the toolbar.

In the Toolbars tab of the Customize dialog, you can toggle a toolbar on and off by clicking in its checkbox. When a toolbar is selected (in the Toolbars tab), you can cause the text labels of that toolbar's items to be displayed by clicking the Show text labels check box. You can also reset a selected toolbar to the state it was in when StyleVision was installed by clicking the Reset button. You can reset all toolbars and the menu bar by clicking the Reset All button.

Note about Menu Bar
Commands can be added to, and items deleted from, the menu bar: see Commands above. To reset the menu bar to the state it was in when StyleVision was installed, select Menu Bar in the Toolbars tab of the Customize dialog, and click the Reset button. (Clicking the Reset All button will reset the toolbars as well.)
Keyboard tab
The Keyboard tab allows you to define (or change) keyboard shortcuts for any StyleVision command.

To assign a shortcut to a command
1. Select the category in which the command is by using the Category combo box.
2. Select the command you want to assign a shortcut to in the Commands list box.
3. Click in the Press New Shortcut Key input field, and press the shortcut keys that are to activate the command. The shortcut immediately appears in the Press New Shortcut Key input field. If this shortcut has already been assigned to a command, then that command is displayed below the input field. (For example, in the screenshot above, Ctrl+C has already been assigned to the Copy command and cannot be assigned to the Open File command.) To clear the New Shortcut Key input field, press any of the control keys, Ctrl, Alt, or Shift.
4. Click the Assign button to permanently assign the shortcut. The shortcut now appears in the Current Keys list box.

To de-assign (or delete) a shortcut
1. Select the command for which the shortcut is to be deleted.
2. Click the shortcut you want to delete in the Current Keys list box.
3. Click the Remove button (which has now become active).

To reset all keyboard assignments
1. Click the Reset All button to go back to the original, installation-time shortcuts. A dialog box appears prompting you to confirm whether you want to reset all keyboard assignments.
2. Click Yes if you want to reset all keyboard assignments.

*Set accelerator for*
Currently no function is available.

---

**Menu tab**
The **Menu** tab allows you to customize the main menu bar as well as the context menus (right-click menus). There are two types of main menu bar: **Default** (which appears when no document is open), and **SPS** (which appears when an SPS document is open).

![Customize dialog](image)

**To customize a menu**
1. Select the menu bar you want to customize (**SPS menu in the screenshot above**).
2. Click the **Commands** tab, and drag the commands to the menu bar of your choice.

**To delete commands from a menu**
1. Right-click the command or the icon in the toolbar representing the command, and
2a. Select the **Delete** option from the popup menu,
or,
2b. Drag the command away from the menu and drop it as soon as the check mark icon appears below the mouse pointer.

**To reset either of the menu bars**
1. Select the menu entry in the combo box.
2. Click the **Reset** button just below the menu name. A prompt appears asking if you are sure you want to reset the menu bar.
To customize a context menu (a right-click menu)
1. Select the context menu from the combo box.
2. Click the **Commands** tab and drag the commands to the context menu that is now open.

To delete commands from a context menu
1. Click right on the command or icon representing the command, and
2. Select the **Delete** option from the popup menu
   or
2. Drag the command away from the context menu and drop it as soon as the check mark icon appears below the mouse pointer.

To reset a context menu
1. Select the context menu from the combo box, and
2. Click the **Reset** button just below the context menu name. A prompt appears asking if you are sure you want to reset the context menu.

To close a context menu window
- Click on the **Close** icon at the top right of the title bar, or
- Click the Close button of the Customize dialog box.

**Menu animations**
The menu animation option specifies the way a menu is displayed when a menu is clicked. Select an option from the drop-down list of menu animations.

**Menu shadows**
If you wish to have menus displayed with a shadow around it, select this option. All menus will then have a shadow.

---

**Options tab**
The **Options** tab allows you to customize additional features of the toolbar.

Screen Tips for toolbar items will be displayed if the Show Screen Tips option is checked. The Screen Tips option has a sub-option for whether shortcuts (where available) are displayed in the Screen Tips or not.

- **See also**
  - Toolbars
  - View | Toolbars
  - User Interface
Restore Toolbars and Windows

This command restores toolbars, windows, entry helpers and other GUI components to their default state. You will need to restart StyleVision for the changes to take effect.

- See also
  - Toolbars
  - View | Toolbars
  - User Interface
Options

The **Options** command opens a dialog in which you specify settings for:

- The maximum width of markup tags in Design View.
- Actions to undertake when a file is double-clicked from within a project window.
- Sorting of attributes and elements in the Schema Tree View,
- The encoding of the output documents, and
- The processing of the internally generated XSL-FO document, which is required in order to enable the PDF Preview and the generation of PDF files in StyleVision.

---

**Design View options**

In the Design tab *(screenshot below)*, you can set the application-wide general options for designs.

![Design View options dialog](image-url)

- **Maximum width of markups**: 400 px
- **Default Unit**: in

**Layout Container Options**

- **Grid Size X**: 0.2 in
- **Grid Size Y**: 0.2 in

Layout boxes that exactly accommodate text in Design View and HTML output might require additional width and height to accommodate the same text in print output formats. The additional width/height specified below will automatically be added to all newly created layout boxes.

- **Default Additional Width**: 20%
- **Default Additional Height**: 10%

Create templates at locations with unknown context node

- Create single template
- Create templates with absolute schema path
- Always ask

---

© 2015 Altova GmbH

Altova StyleVision 2015 Enterprise Edition
The following options can be set:

- Maximum width (in pixels) of markup tags. Enter the positive integer that is the required number of pixels.
- Grid size of layout containers in absolute length units. The specified lengths are the distances between two points on the respective grid axis.
- Default additional width and height of Layout Boxes. These additional lengths are added to all layout boxes in order to provide the extra length that is often required to accommodate the bigger text renditions of print formats. These values can be specified as percentage values or as absolute length units.
- The default behavior when a node-template is created at a location where the context node is not known. This option typically applies to User-Defined Templates in which the template has been created for items that cannot be placed in context in the schema source of the design. If a node is created within such a user-defined template, then the node can be created with (i) only its name, or (ii) with the full path to it from the schema root. You can set one of these options as the default behavior, or, alternatively, ask to be prompted each time this situation arises. The default selection for this option is Always Ask.

**Schema Tree options**

In the Schema Tree, elements and attributes can be listed alphabetically in ascending order. To do this, check the respective check boxes in the Schema tab (screenshot below). By default, attributes are listed alphabetically and elements are listed in an order corresponding to the schema structure, as far as this is possible.

![Schema Tree options screenshot](screenshot)

**Project options**

In the Project sidebar, when an XML file or XSD file is double-clicked, one of three actions is executed depending on the options set in the Project tab of the Options dialog (screenshot below): (i) Edit the file in XMLSpy; (ii) Create a new design based on the selected file; (iii) Ask the
user which action to execute.

If the Save Without Confirmation setting is checked, then any changes to the project are automatically saved when the project or StyleVision is closed. Otherwise, you will be prompted to save any unsaved changes.

**Encoding of output files**

In the Default Encoding tab (**screenshot below**), you can set default encodings for the various outputs separately. The encoding specifies the codepoints sets for various character sets. The dropdown list of each combo box displays a list of encoding options. Select the encoding you require for each output type, and click **OK**. Every new SPS you create from this point on will set the respective output encodings as defined in this tab.
In the XSLT-for-HTML, the output encoding information is registered at the following locations:

- In the encoding attribute of the stylesheet’s xsl:output element:
  
  `<xsl:output version="1.0" encoding="UTF-8" indent="no" omit-xml-declaration="no" media-type="text/html" />`

- In the charset attribute of the content-type meta element in the HTML header:
  
  `<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />`

In the XSLT-for-RTF, the output encoding information is registered in the encoding attribute of the stylesheet’s xsl:output element:

- `<xsl:output version="1.0" encoding="ISO-8859-1" indent="no" method="text" omit-xml-declaration="yes" media-type="text/rtf" />`

In the XSLT-for-PDF, the output encoding information is registered in the encoding attribute of the stylesheet’s xsl:output element:

- `<xsl:output version="1.0" encoding="UTF-8" indent="no" omit-xml-declaration="no" media-type="text/html" />`

Note: These settings are the default encodings, and will be used for new SPSs. You cannot change the encoding of the currently open SPS using this dialog. To change the encoding of the currently open SPS, use the File | Properties command.

---

**XSL options**

In the meta information of HTML and PDF output files, the line ‘Generated by StyleVision’ will be generated by default. Purchased versions of the product provide an option to disable the generation of this line.

Once the XSL-FO settings are correctly made, each time the PDF Preview tab is clicked, the following happens:

1. An XSL-FO document is generated (behind the interface) by processing the Working XML
2. The XSL-FO is then processed by an FO processor to generate the PDF that is displayed in the PDF Preview window.

In the XSL-FO tab, the following settings must be made in order for the PDF Preview to be enabled and for PDF files to be generated.

**FO Processor**
You can use any FO processor of your choice. All you have to do is enter the path to the processor's executable in this dialog. For details, see Setting up StyleVision.

**XSL-FO Generation**
Selecting the FOP 0.93/1.0 compliant option filters out those FOP properties that have not yet been implemented in the latest version of FOP. Selecting the Full XSL-FO standard option will not filter out any properties when the XSLT-for-FO is being generated.

Depending on your selection, the XSLT-for-FO will be created with or without the objects, properties, and values that are beyond the compliance level of FOP 0.93/1.0. You should be aware of the possible outcomes from this point in the processing. These are as follows:

- If you have selected FOP compliance and use FOP as your processor, then the PDF generation should work fine.
- If you have selected full usage of the XSL-FO standard and use FOP as your processor, then any formatting object or property in the FO document that is not supported by the current version of FOP will cause FOP to generate either a warning or an error. An error is fatal, and no PDF will be generated.
- If you use a processor other than FOP, the success of the transformation will depend upon the compliance level of the processor.
For standard use, we recommend:

- FOP 1.0. If you notice any problems with FOP 1.0, we suggest you use FOP 0.93.
- Selecting FOP 0.93/1.0 compliant in the Options dialog.

**Note:** The XSLT-for-FO that you generate with the **File | Save Generated Files | Save Generated XSL-FO File** command will be either in compliance with FOP 0.93/1.0 or with the full XSL-FO standard, depending on the selection you make for the XSL-FO Generation option in this dialog. Any error will only be reported when the FO document is processed by the FO processor.

**Technical Note:** StyleVision opens the executable of the FO processor using the path you enter in the FO Processor pane of the Options dialog. It sends the following arguments to the executable:

```
%1 : -fo
%2 : filename.fo
%3 : -pdf
%4 : filename.pdf
```

These arguments are as used in the FOP command: `FOP -fo filename.fo -pdf filename.pdf`. You can use these arguments to invoke any FO processor, if the arguments suffice to make a successful call.

---

**See also**

- **File | Properties**
- **Setting up StyleVision**
- **Projects in StyleVision**
20.15 Window Menu

The Window menu has commands to specify how StyleVision windows should be displayed in the GUI (cascaded, tiled, or maximized). To maximize a window, click the maximize button of that window.

Additionally, all currently open document windows are listed in this menu by document name, with the active window being checked. To make another window active, click the name of the window you wish to make active.

Windows dialog
At the bottom of the list of open windows is an entry for the Windows dialog. Clicking this entry opens the Windows dialog, which displays a list of all open windows and provides commands that can be applied to the selected window/s. (A window is selected by clicking on its name.)

Warning: To exit the Windows dialog, click OK; do not click the Close Window(s) button. The Close Window(s) button closes the window/s currently selected in the Windows dialog.

See also
- User Interface
20.16 Help Menu

The Help menu contains commands to access the onscreen help manual for StyleVision, commands to provide information about StyleVision, and links to support pages on the Altova website. The Help menu also contains the Registration dialog, which lets you enter your license key-code once you have purchased the product.

The description of the Help menu commands is organized into the following sub-sections:

- Table of Contents, Index, Search
- Activation, Order Form, Registration, Updates
- Other Commands

See also
- About this Documentation
Table of Contents, Index, Search

♦ Table of Contents
  ✓ Description
  Opens the onscreen help manual of StyleVision 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 StyleVision with the Keyword Index displayed in the left-hand-side pane of the Help window. The index lists keywords and lets you navigate to a topic by double-clicking the keyword. If a keyword is linked to more than one topic, a list of these topics is displayed.

♦ Search
  ✓ Description
  Opens the onscreen help manual of StyleVision with the Search dialog displayed in the left-hand-side pane of the Help window. To search for a term, enter the term in the input field, and press Return. The Help system performs a full-text search on the entire Help documentation and returns a list of hits. Double-click any item to display that item.
Activation, Order Form, Registration, Updates

**Software Activation**

- **Description**

  After you download your Altova product software, you can 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.

  The Software Activation dialog can be accessed at any time by clicking the Help | Software Activation command.

**Order Form**

- **Description**

  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.
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.

- Components Download
  - *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.

- StyleVision on the Internet
  - *Description*
  
  A link to the Altova website on the Internet. You can learn more about StyleVision and related technologies and products at the Altova website.

- About StyleVision
  - *Description*
  
  Displays the splash window and version number of your product. If you are using the 64-bit version of StyleVision, this is indicated with the suffix (x64) after the application name. There is no suffix for the 32-bit version.
Chapter 21

Programmers' Reference
21 Programmers' Reference

StyleVision as an Automation Server
StyleVision is an Automation Server. That is, it is an application that exposes programmable objects to other applications (called Automation Clients). As a result, an Automation Client can directly access the objects and functionality that the Automation Server makes available. This is beneficial to an Automation Client because it can make use of the functionality of StyleVision. For example, an Automation Client can generate an XSLT file from an SPS via StyleVision. Developers can therefore improve their applications by using the ready-made functionality of StyleVision.

The programmable objects of StyleVision are made available to Automation Clients via the **StyleVision API**, which is a COM API. A complete description of all available objects are provided in this documentation (see the section Application API).
21.1 Scripting Editor

The Scripting Editor of StyleVision uses the Form Editor components of the Microsoft .NET Framework, and thus provides access to the Microsoft .NET Framework.

You can therefore create and use your own macros and forms to use in the Authentic View of the Altova products StyleVision, Authentic Desktop, Authentic Browser, and XMLSpy.

Note: Microsoft’s .NET Framework 2.0 or higher is a system prerequisite for Scripting Editor, and it must be installed before StyleVision is installed.

The Scripting Editor
The Scripting Editor (screenshot below) opens in a separate window and is accessed via the Authentic | Edit Authentic Script menu command in the StyleVision GUI. It can also be accessed by selecting the Authentic Script item in the Design tab at the bottom of the 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. The Scripting Project can be saved temporarily in memory, either by clicking the Save button or Save & Close button. The Scripting Project is saved with the SPS only when the SPS is subsequently saved. Scripts that have been created in the Scripting Project can then be used in the SPS design to produce effects in the Authentic View output.
**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 StyleVision-specific event handlers](#).
- [An explanation of how to use macros](#) in the Scripting Editor and in StyleVision.
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 saved in a single scripting project, which can be accessed via the **Authentic | Edit Authentic Scripts** command. The scripting project is opened in StyleVision's **Scripting Editor** and can be edited there. On saving the scripting project, the Forms, Event Handlers and Macros in it are available for use in the SPS design of the Authentic View output.

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:

- **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.
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
- A Form Object Palette (right-hand side)

Scripting Editor toolbar

The Scripting Editor toolbar contains icons for:

- Standard file and editing commands: **Undo**, **Redo**, **Cut**, **Copy**, **Paste**, **Print**, **Find**, and **Replace**. These commands are used for various editing operations and print scripting projects. 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 screenshot above), 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 and creating a new form. 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
  ```java
  if ( x == y.GetVal(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.
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 StyleVision API to access StyleVision 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 in Authentic View to start execution of the delete functionality, which is available in the StyleVision 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 in Authentic View. 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. 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 StyleVision interface by binding it to an Authentic event and triggering that event.
Creating a Scripting Project

Each SPS can have a single scripting project assigned to it. To access the scripting project associated with an SPS, select the command Tools | Edit Authentic Scripts. This pops up the Scripting Editor window, in which Forms, Event Handlers, and Macros can be created and edited. After the scripting project has been created or edited, it must be saved in memory (by clicking the Save button in the Scripting Editor toolbar (or Ctrl+S)). The scripts in the saved scripting project, however, will only be saved to the SPS when the SPS is saved. So, make sure to: (i) save the scripting project (to memory), and (ii) subsequently save the SPS (to a file location).

To change the scripting language and/or the target .NET Framework, right-click the project, select Project Settings, and make the appropriate selections.

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 Authentic View. If an Event Handler for a single event is defined in both the Global Scripting Project and the StyleVision-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 StyleVision interface by binding it to an Authentic event and triggering that event. In a Macro, it is possible to access all variables and functions declared in the Global Declarations and to display Forms for user input.
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 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-initialized.
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.
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. Specifying 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. The Form itself will load with all the Form objects present. However, note that code associated with various form controls will **not** be executed. To test the code you will have to run a macro containing the form, outside the Scripting Editor.
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.

<table>
<thead>
<tr>
<th>&lt;Pointer&gt;</th>
<th>A Label</th>
<th>A LinkLabel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button</td>
<td>abl TextBox</td>
<td>CheckBox</td>
</tr>
<tr>
<td>RadioButton</td>
<td>GroupBox</td>
<td>PictureBox</td>
</tr>
<tr>
<td>Panel</td>
<td>DataGridView</td>
<td>ListBox</td>
</tr>
<tr>
<td>CheckedListBox</td>
<td>ComboBox</td>
<td>DateTimePicker</td>
</tr>
<tr>
<td>TreeView</td>
<td>TabControl</td>
<td>ScrollBar</td>
</tr>
<tr>
<td>MonthCalendar</td>
<td>HScrollBar</td>
<td>VScrollBar</td>
</tr>
<tr>
<td>Timer</td>
<td>Splitter</td>
<td>DomainUpDown</td>
</tr>
<tr>
<td>NumericUpDown</td>
<td>TrackBar</td>
<td>ProgressBar</td>
</tr>
<tr>
<td>RichTextBox</td>
<td>ImageList</td>
<td>HelpProvider</td>
</tr>
<tr>
<td>ToolTip</td>
<td>ToolBar</td>
<td>StatusBar</td>
</tr>
<tr>
<td>UserControl</td>
<td>NotifyIcon</td>
<td>OpenFileDialog</td>
</tr>
<tr>
<td>SaveFileDialog</td>
<td>FontDialog</td>
<td>ColorDialog</td>
</tr>
<tr>
<td>PrintDialog</td>
<td>PrintPreviewDialog</td>
<td>PrintPreviewControl</td>
</tr>
<tr>
<td>ErrorProvider</td>
<td>PrintDocument</td>
<td>PageSetupDialog</td>
</tr>
</tbody>
</table>

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.
Check Box: Adds a check box, which enables Yes/No type selections.

Combo Box: 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. The Form itself will load with all the Form objects present. However, note that code associated with various form controls will **not** be executed. To test the code you will have to run a macro containing the form, outside the Scripting Editor.
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.

```plaintext
//Occurs when the component is clicked.
function FormExample_Label1_Click(e)
{
}
```

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):

```javascript
function FormExample_Button1_Click( objSender, e_EventArgs )
{
    // Sets the ForeColor (red) of the button.
    // Sets the BackColor (blue) of the button.
    // Sets the form BackColor (green).
}
```
Events

The Events folder of the scripting project (see screenshot below) contains a folder for Authentic View Events. For each event listed in the folder, Event Handler scripts can be written.

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.
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 StyleVision. 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 in StyleVision.
- 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.
- StyleVision 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.
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 macros is run.

```csharp
objForm = CreateForm ( 'Form1' );
objForm.ShowDialog();
```

Note: Macros do not support parameters or return values.
Running a Macro

To run a macro, you need to first bind the macro to an Authentic event. The macro will run when that Authentic event is triggered. For example, say you wish to run a macro before editing text in Authentic. You must go about this as follows:

1. In Design View, select the editable content for which the macro is to be set.
2. In the Properties sidebar (screenshot below), select the Authentic group of properties of the relevant design component.

```
<table>
<thead>
<tr>
<th>Group</th>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>content</td>
<td></td>
</tr>
<tr>
<td></td>
<td>additional validation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>editable</td>
<td>inherit</td>
</tr>
<tr>
<td>common</td>
<td>OnAfterChange</td>
<td>ShowDialog</td>
</tr>
<tr>
<td>event</td>
<td>OnBeforeChange</td>
<td>SayHello</td>
</tr>
<tr>
<td></td>
<td>OnKillFocus</td>
<td>ShowDialog</td>
</tr>
<tr>
<td></td>
<td>OnSetFocus</td>
<td>SayHello</td>
</tr>
<tr>
<td></td>
<td>user input</td>
<td></td>
</tr>
</tbody>
</table>
```

3. Select the Authentic event that will trigger the macro and open the dropdown list of this event's Value combo box. All the macros defined in the Scripting Project of the SPS will be listed. Select the macro you wish to associate with this function.
4. In Authentic View, when any Authentic event that has a macro assigned to it is triggered, then the macro will be executed.

See the section, **Authentic Scripting >> Macros**, for a detailed description of how to use macros in a StyleVision design document.
Debugging a Macro

You can debug a macro using an installed debugger. To do this, click the dropdown icon of the Authentic View tab (screenshot below) and toggle on the option Run Scripts in Debug Mode.

From now on, whenever a macro is triggered from Authentic View, the macro will be run in a debugger that you select. If you are running scripts outside StyleVision, say, for example, in Authentic Browser, and wish to run the script in debugger mode, check the Run Authentic Scripts in Debug Mode Outside StyleVision check box, which is in the Authentic tab of the Properties dialog (accessed with the File | Properties command).

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.

The selected debugger starts.
21.2 Application API

The COM-based API of StyleVision (also called the Application API from now on) enables other applications to use the functionality of StyleVision.

StyleVision 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 StyleVision. For a description of the scripting environment, see the section, Scripting Editor.
- 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 StyleVision can be accessed and automated from external programs.

Using the Application API from outside StyleVision requires an instance of StyleVision 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, StyleVision will be started via its COM registration. Then the Application object associated with the StyleVision instance is returned. Depending on the COM settings, an object associated with an already running StyleVision 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 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 StyleVision 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.
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.
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.

To create an instance of the Application object, call CreateObject("StyleVision.Application") from VisualBasic or a similar function from your preferred development environment to create a COM object. There is no need to create any other objects in order to use the complete Application API; it is in fact not even possible. All other interfaces are accessed through other objects with the Application object as the starting point.

The chart below shows the links between the main objects of the Application API

Application
 |-- Documents
     |-- Document
         |-- Methods for assigning working XML file, saving files
         |-- Methods for saving generated files
         |-- Methods for activating, saving, closing SPS file

In addition there are objects for schema sources and properties, as well as methods for AuthenticView events.

Once you have created an Application object you can start using the functionality of StyleVision.
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:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows XP</td>
<td>C:/Documents and Settings/&lt;username&gt;/My Documents/Altova/StyleVision2015/StyleVisionExamples/API/</td>
</tr>
</tbody>
</table>

JScript

The JScript listings demonstrate the following basic functionality:

- Start application or attach to a running instance
- Simple document access
- Iteration

VBScript

VBScript is different than JScript only syntactically; otherwise it works in the same way. For more information, refer to the JScript examples.

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 StyleVision: Starts StyleVision, which is registered as an automation server, or activates StyleVision if it is already running.
- Open OrgChart.pxf: Locates one of the example documents installed with StyleVision and opens it. If this document is already open it becomes the active document.
- OnDocumentOpened Event On/Off: Shows how to listen to StyleVision events. When turned on, a message box will pop up after a document has been opened.
- Open ExpReport.xml: Opens another example document.
- Shutdown StyleVision: Stops StyleVision.

Java

The StyleVision API can be accessed from Java code. The Java sub-section of this section explains how some basic StyleVision 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
- 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**

**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:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows XP</td>
<td><code>C:/Documents and Settings/&lt;username&gt;/My Documents/Altova/StyleVision2015/StyleVisionExamples/API/</code></td>
</tr>
</tbody>
</table>

The example files can be run in one of two ways:

- **From the command line**: Open a command prompt window and type the name of one of the example scripts (for example, `Start.js`). The Windows Scripting Host that is packaged with newer versions of Windows (XP, Vista, 7) will execute the script.
- **From Windows Explorer**: In Windows Explorer, browse for the JScript file and double-click it. The Windows Scripting Host that is packaged with newer versions of Windows (XP, Vista, 7) will execute the script. After the script is executed, the command console gets closed automatically.

**Start Application**

The JScript below starts the application and shuts it down. If an instance of the application is already running, the running instance will be returned.

**Script listing**

The JScript listing below is explained with comments in the code.

```javascript
// Initialize application's COM object. This will start a new instance of the application and return its main COM object. Depending on COM settings, a the main COM object of an already running application might be returned.
try
{
    objStyleVision = WScript.GetObject("", "StyleVision.Application");
}
catch(err)
{
    Exit("Can't access or create StyleVision.Application");
}

// if newly started, the application will start without its UI visible. Set it
to visible.
objStyleVision.Visible = true;

WScript.Echo("Hello");

objStyleVision.Visible = false; // will shutdown application if it has no more COM connections
//objStyleVision.Visible = true; // will keep application running with UI visible

Running the script
The JScript code listed above is available in the file Start.js located in the JScript folder of the API Examples folder:

<table>
<thead>
<tr>
<th>Windows XP</th>
<th>C:/Documents and Settings/&lt;username&gt;/My Documents/Altova/StyleVision2015/StyleVisionExamples/API/</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Vista, Windows 7,</td>
<td>C:/Users/&lt;username&gt;/Documents/Altova/StyleVision2015/StyleVisionExamples/API/</td>
</tr>
<tr>
<td>Windows 8</td>
<td></td>
</tr>
</tbody>
</table>

To run the script, start it from a command prompt window or from Windows Explorer.

Simple Document Access
Enter topic text here

Running the script
The JScript code listed below shows how to open documents, set a document as the active document, iterate through the open documents, and close documents.

Running the script
The JScript code listed below is available in the file DocumentAccess.js located in the JScript folder of the API Examples folder:

<table>
<thead>
<tr>
<th>Windows XP</th>
<th>C:/Documents and Settings/&lt;username&gt;/My Documents/Altova/StyleVision2015/StyleVisionExamples/API/</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Vista, Windows 7,</td>
<td>C:/Users/&lt;username&gt;/Documents/Altova/StyleVision2015/StyleVisionExamples/API/</td>
</tr>
<tr>
<td>Windows 8</td>
<td></td>
</tr>
</tbody>
</table>

To run the script, start it from a command prompt window or from Windows Explorer.

Script listing
The JScript listing below is explained with comments in the code.

// Initialize application's COM object. This will start a new instance of the application and
// return its main COM object. Depending on COM settings, a the main COM object of an already
// running application might be returned.
try {
    objStyleVision = WScript.GetObject("", "StyleVision.Application");
catch(err) {
    Exit("Can't access or create StyleVision.Application");
}

// if newly started, the application will start without its UI visible. Set it to visible.
objStyleVision.Visible = true;

// *************************** code snippet for "Simple Document Access"
***************************

// Locate examples via USERPROFILE shell variable. The path needs to be adapted to major release versions.
objWshShell = WScript.CreateObject("WScript.Shell");
strExampleFolder = objWshShell-expandEnvironmentStrings("%USERPROFILE%") + "\My Documents\Altova\StyleVision2012\StyleVisionExamples\";

// Tell Authentic to open two documents. No dialogs
objDoc1 = objStyleVision.Documents.OpenDocument(strExampleFolder + "OrgChart.pxf");
objStyleVision.Documents.OpenDocument(strExampleFolder + "BiggestCitiesPerContinent.sps");

// *************************** code snippet for "Simple Document Access"
***************************

// *************************** code snippet for "Iteration"
***************************

// go through all open documents using a JScript Enumerator
for (var iterDocs = new Enumerator(objStyleVision.Documents); !iterDocs.atEnd(); iterDocs.moveNext()) {
    objName = iterDocs.item().Name;
    WScript.Echo("Document name: " + objName);
}

// go through all open documents using index-based access to the document collection
for (i = objStyleVision.Documents.Count; i > 0; i--)
    objStyleVision.Documents.Item(i).Close();

// *************************** code snippet for "Iteration"
***************************

//objStyleVision.Visible = false; // will shutdown application if it has no more COM connections
objStyleVision.Visible = true; // will keep application running with UI visible

Iteration

The JScript listing below shows how to iterate through the open documents.

Running the script
You can test this script by copying the listing below to a file, naming the file with a .js extension, and running the file from either the command line or Windows Explorer. You could copy the file to the JScript folder of the API Examples folder:

| Windows XP | C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2015/StyleVisionExamples/API/ |

To run the script, start it from a command prompt window or from Windows Explorer.

**Script listing**
The JScript listing below is explained with comments in the code.

```javascript
// Initialize application's COM object. This will start a new instance of the application and return its main COM object. Depending on COM settings, a the main COM object of an already running application might be returned.
try {
    objStyleVision = WScript.GetObject("", "StyleVision.Application");
} catch(err) {
    Exit("Can't access or create StyleVision.Application");
}

// if newly started, the application will start without its UI visible. Set it to visible.
objStyleVision.Visible = true;

// **************************** code snippet for "Simple Document Access"

// Locate examples via USERPROFILE shell variable. The path needs to be adapted to major release versions.
objWshShell = WScript.CreateObject("WScript.Shell");
strExampleFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\My Documents\Altova\StyleVision2012\StyleVisionExamples\"

// Tell Authentic to open two documents. No dialogs
objDoc1 = objStyleVision.Documents.OpenDocument(strExampleFolder + "OrgChart.pxf");
objStyleVision.Documents.OpenDocument(strExampleFolder + "BiggestCitiesPerContinent.sps");

// **************************** code snippet for "Simple Document Access"

// **************************** code snippet for "Iteration"

// go through all open documents using a JScript Enumerator
for (var iterDocs = new Enumerator(objStyleVision.Documents); !iterDocs.atEnd();)
```

© 2015 Altova GmbHAltova StyleVision 2015 Enterprise Edition
C# programming language can be used to access the Application API functionality. You could use Visual Studio 2008 or Visual Studio 2010 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 StyleVision Type Library by clicking Project | Add Reference. The Add Reference dialog pops up, displaying a list of installed COM components. Select the StyleVision Type Library component from the list to add it.
3. Enter the code you want.
4. Compile the code and run it.

Example C# project

Your StyleVision package contains an example C# project, which is located in the C# folder of the API Examples folder:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Folder Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows XP</td>
<td>C:/Documents and Settings/&lt;username&gt;/My Documents/Altova/StyleVision2015/StyleVisionExamples/API/</td>
</tr>
</tbody>
</table>
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 StyleVision operations:

![Screenshot](image)

- **Start StyleVision**: Starts StyleVision, 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 StyleVision and opens it. If this document is already open it becomes the active document.
- **Open MultiFileOutput.sps**: Opens another example document.
- **Shut down StyleVision**: Stops StyleVision.

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 AutomateStyleVision_VS2008.sln (to open it in Visual Studio 2008) or the file AutomateStyleVision_VS2010.sln (to open it in Visual Studio 2010). 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 AutomateStyleVision 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 StyleVision is accessed via its automation interface.
        StyleVisionLib.Application StyleVision;

        // Location of examples installed with StyleVision
        String strExamplesFolder;

        private void Form1_Load(object sender, EventArgs e)
        {
            // Locate examples installed with StyleVision.
            // REMARK: You might need to adapt this if you have a different major version of the product.
            strExamplesFolder = Environment.GetEnvironmentVariable("USERPROFILE") + "\My Documents\Altova\StyleVision2012\StyleVisionExamples\";
        }

        // Handler for the "Start StyleVision" button
        private void StartStyleVision_Click(object sender, EventArgs e)
        {
            if (StyleVision == null)
            {
                Cursor.Current = Cursors.WaitCursor;

                // If there is no StyleVision instance, create one and make it visible.
                StyleVision = new StyleVisionLib.Application();
                StyleVision.Visible = true;
            }
            else
            {
                // If a StyleVision instance is already running, make sure it's visible.
                if (!StyleVision.Visible)
                    StyleVision.Visible = true;
            }
        }

        // Handler for the "Open OrgChart.pxf" button
        private void openOrgChart_Click(object sender, EventArgs e)
        {
            // Make sure there's a running StyleVision instance, and that it's
```
visible
  StartStyleVision_Click(null, null);

  // Open a sample file installed with the product.
  StyleVision.Documents.OpenDocument(strExamplesFolder + "OrgChart.pxf");
  updateListBox();

  // Handler for the "Open MultiFileOutput.sps" button
private void openMultiFileOutput_Click(object sender, EventArgs e)
{
  // Make sure there's a running StyleVision instance, and that it's visible
  StartStyleVision_Click(null, null);

  // Open one of the sample files installed with the product.
  updateListBox();
}

  // Handler for the "Shutdown StyleVision" button
  // Shut down the application instance by explicitly releasing the COM object.
private void shutdownStyleVision_Click(object sender, EventArgs e)
{
  if (StyleVision != null)
  {
    // Allow shut-down of StyleVision by releasing UI
    StyleVision.Visible = false;

    // Explicitly release the COM object
    try
    {
      int i = System.Runtime.InteropServices.Marshal.ReleaseComObject(StyleVision);
      while (System.Runtime.InteropServices.Marshal.ReleaseComObject(StyleVision) > 0) ;
    }
    finally
    {
      // Disallow subsequent access to this object.
      StyleVision = null;
    }
  }
}

delegate void addListBoxItem_delegate(string sText);
  // Called from the UI thread
private void addListBoxItem(string sText)
{
  listBoxMessages.Items.Add(sText);
}
  // Wrapper method to call UI control methods from a worker thread
void syncWithUIThread(Control ctrl, addListBoxItem_delegate methodToInvoke, String sText)
{
  // Control.Invoke: Executes on the UI thread, but calling thread waits for completion before continuing.
// Control.BeginInvoke: Executes on the UI thread, and calling
// thread doesn't wait for completion.
if (ctrl.InvokeRequired)
    ctrl.BeginInvoke(methodToInvoke, new Object[]{sText});
}

// Event handler for OnDocumentClosed event
private void handleOnDocumentClosed(StyleVisionLib.Document i_ipDocument)
{
    String sText = "";
    if (i_ipDocument.Name.Length > 0)
        sText = "Document " + i_ipDocument.Name + " was closed!";

    // Synchronize the calling thread with the UI thread because
    // COM events are triggered from a working thread
    addListBoxItem delegate methodToInvoke = new
    addListBoxItem delegate(addListBoxItem);
    // Call syncWithUIthread with the following arguments:
    // 1 - listBoxMessages - list box control to display messages from
    // 2 - methodToInvoke - a C# delegate which points to the method
    // which will be called from the UI thread
    // 3 - sText           - the text to be displayed in the list box
    syncWithUIthread(listBoxMessages, methodToInvoke, sText);
}

private void updateListBox()
{
    // Iterate through all open documents
    listBoxMessages.Items.Clear();

    for (int i = 1; i <= StyleVision.Documents.Count; i++)
    {
        StyleVisionLib.Document doc = StyleVision.Documents[i];

        if (doc != null)
        {
            if (checkBoxEventOnOff.Checked)
                doc.OnDocumentClosed += new
                StyleVisionLib._IDocumentEvents_OnDocumentClosedEventHandler(handleOnDocumentClosed);
            else
                doc.OnDocumentClosed -= new
                StyleVisionLib._IDocumentEvents_OnDocumentClosedEventHandler(handleOnDocumentClosed);

            listBoxMessages.Items.Add(doc.Name);
            StyleVisionLib.SchemaSources sources = doc.SchemaSources;

            for (int j = 1; j <= sources.Count; j++)
            {
                StyleVisionLib.SchemaSource source = sources[j];

                if (source != null)
Add Reference to StyleVision 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 StyleVision.tlb, and is located in the StyleVision application folder.

Then declare a variable to access the StyleVision API:

```csharp
// An instance of StyleVision is accessed via its automation interface.
StyleVisionLib.Application StyleVision;
```

Application Startup and Shutdown

In the code snippets below, the methods StartStyleVision_Click and ShutdownStyleVision_Click are those assigned to buttons in the AutomateStyleVision 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 XP</th>
<th>C:/Documents and Settings/username/My Documents/Altova/StyleVision2015/StyleVisionExamples/API/</th>
</tr>
</thead>
</table>

You can compile and run the project from within Visual Studio 2008 or Visual Studio 2010.

Starting StyleVision

The following code snippet from the AutomateStyleVision example shows how to start up the application.

```csharp
// Handler for the "Start StyleVision" button
private void StartStyleVision_Click(object sender, EventArgs e)
{
    if (StyleVision == null)
    {
        Cursor.Current = Cursors.WaitCursor;
        // If there is no StyleVision instance, create one and make it visible.
```

```csharp
```
StyleVision = new StyleVisionLib.Application();
StyleVision.Visible = true;

Cursor.Current = Cursors.Default;
}
else{
    // If an instance of StyleVision is already running, make sure it's visible
    if (!StyleVision.Visible)
    StyleVision.Visible = true;
}

---

**Shutting down StyleVision**

The following code snippet from the [AutomateStyleVision example](#) shows how to shut down the application.

```csharp
// Handler for the "Shutdown StyleVision" button
// Shut down the application instance by explicitly releasing the COM object.
private void shutdownStyleVision_Click(object sender, EventArgs e)
{
    if (StyleVision != null)
    {
        // Allow shut-down of StyleVision by releasing UI
        StyleVision.Visible = false;

        // Explicitly release the COM object
        try
        {
            int i = System.Runtime.InteropServices.Marshal.ReleaseComObject(StyleVision);
            while (System.Runtime.InteropServices.Marshal.ReleaseComObject(StyleVision) > 0) ;
        }
        finally
        {
            // Disallow subsequent access to this object.
            StyleVision = null;
        }
    }
}
```

**Opening Documents**

The code snippets below (from the [AutomateStyleVision 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: `Documents.OpenDocument(string)`.

The [AutomateStyleVision example](#) (see the file `Form1.cs`) is located in the C# folder of the API Examples folder:

| Windows XP | C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2015/StyleVisionExamples/API/ |
You can compile and run the project from within Visual Studio 2008 or Visual Studio 2010.

**Code snippet**

```csharp
// Handler for the "Open OrgChart.pxf" button
private void openOrgChart_Click(object sender, EventArgs e)
{
    // Make sure there's a running StyleVision instance, and that it's visible
    StartStyleVision_Click(null, null);

    // Open a sample files installed with the product.
    StyleVision.Documents.OpenDocument(strExamplesFolder + "OrgChart.pxf");
    updateListBox();
}

// Handler for the "Open MultiFileOutput.sps" button
private void openMultiFileOutput_Click(object sender, EventArgs e)
{
    if (StyleVision == null)
        StartStyleVision_Click(null, null);

    // Open one of the sample files installed with the product.
    updateListBox();
}
```

The file opened last will be the active file.

**Events**

The code snippet below (from the AutomateStyleVision example) lists the code for two event handlers. The AutomateStyleVision 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 2008 or Visual Studio 2010.

```csharp
delegate void addListBoxItem_delegate(string sText);
// Called from the UI thread
private void addListBoxItem(string sText)
{
    listBoxMessages.Items.Add(sText);
}
```
// Wrapper method to call UI control methods from a worker thread
void syncWithUIthread(Control ctrl, addListBoxItem_delegate methodToInvoke, String sText)
{
    // Control.Invoke: Executes on the UI thread, but calling thread waits for completion before continuing.
    // Control.BeginInvoke: Executes on the UI thread, and calling thread doesn't wait for completion.
    if (ctrl.InvokeRequired)
        ctrl.BeginInvoke(methodToInvoke, new Object[]{sText});
}

// Event handler for OnDocumentClosed event
private void handleOnDocumentClosed(StyleVisionLib.Document i_ipDocument)
{
    String sText = "";
    if (i_ipDocument.Name.Length > 0)
        sText = "Document " + i_ipDocument.Name + " was closed!";

    // Synchronize the calling thread with the UI thread because
    // COM events are triggered from a working thread
    addListBoxItem_delegate methodToInvoke = new addListBoxItem_delegate(addListBoxItem);
    // Call syncWithUIthread with the following arguments:
    // 1 - listBoxMessages - list box control to display messages from
    // COM events
    // 2 - methodToInvoke - a C# delegate which points to the method
    // which will be called from the UI thread
    // 3 - sText - the text to be displayed in the list box
    syncWithUIthread(listBoxMessages, methodToInvoke, sText);
}

private void updateListBox()
{
    // Iterate through all open documents
    listBoxMessages.Items.Clear();

    for (int i = 1; i <= StyleVision.Documents.Count; i++)
    {
        StyleVisionLib.Document doc = StyleVision.Documents[i];

        if (doc != null)
        {
            if (checkBoxEventOnOff.Checked)
            else

            listBoxMessages.Items.Add(doc.Name);
            StyleVisionLib.SchemaSources sources = doc.SchemaSources;
for (int j = 1; j <= sources.Count; j++)
{
    StyleVisionLib.SchemaSource source = sources[j];

    if (source != null)
    {
        listBoxMessages.Items.Add("\tSchema file name : " + source.SchemaFileName + "\n");
        listBoxMessages.Items.Add("\tWorking XML file name : " + source.WorkingXMLFileName + "\n");
        listBoxMessages.Items.Add("\tIs main schema source : " + source.IsMainSchemaSource + "\n");
        listBoxMessages.Items.Add("\tType name : " + source.TypeName + "\n");
    }
}
}

Java

The Application API can be accessed from Java code. To allow accessing the StyleVision automation server directly from Java code, the libraries listed below must reside in the classpath. They are installed in the folder: JavaAPI in the StyleVision 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
- StyleVisionAPI.jar: Java classes that wrap the StyleVision automation interface
- StyleVisionAPI_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 StyleVision 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:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Java name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document, method SetEncoding</td>
<td>setFileEncoding</td>
</tr>
<tr>
<td>AuthenticView, method Goto</td>
<td>gotoElement</td>
</tr>
<tr>
<td>AuthenticRange, method Goto</td>
<td>gotoElement</td>
</tr>
<tr>
<td>AuthenticRange, method Clone</td>
<td>cloneRange</td>
</tr>
</tbody>
</table>

**This section**

This section explains how some basic StyleVision 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**
- **Event Handlers**

**Example Java Project**

The StyleVision installation package contains an example Java project, located in the Java folder of the API Examples folder:

<table>
<thead>
<tr>
<th>Windows XP</th>
<th>C:/Documents and Settings/&lt;username&gt;/My Documents/Altova/StyleVision2015/StyleVisionExamples/API/</th>
</tr>
</thead>
</table>

This folder contains Java examples for the StyleVision 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
- StyleVisionAPI.jar: Java classes of the StyleVision API
- RunStyleVision.java: Java example source code
- BuildAndRun.bat: Batch file to compile and run example code from the command line prompt. Expects folder where Java Virtual Machine resides as parameter.
- .classpath: Eclipse project helper file
- .project: Eclipse project file
- StyleVision_JavaDoc.zip: Javadoc file containing help documentation for the Java API

**What the example does**
The example starts up StyleVision and performs a few operations, including opening and closing documents. When done, StyleVision stays open. You must close it manually.

- **Start StyleVision**: Starts StyleVision, which is registered as an automation server, or activates StyleVision if it is already running.
- **Open OrgChart.pxf**: Locates one of the example documents installed with StyleVision 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.
- **Event Handling**: Shows how to handle StyleVision events.
- **Shut down StyleVision**: Shuts down StyleVision.

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:

```
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 RunStyleVision.java will be compiled and then
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 RunStyleVision 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 RunStyleVision.java is listed below with comments.

```java
01 // Access general JAVA-COM bridge classes
02 import com.altova.automation.libs.*;
03 // Access StyleVision Java-COM bridge
04 import com.altova.automation.StyleVision.*;
05 import com.altova.automation.StyleVision.Enums.ENUMApplicationStatus;
06 /*
07 * A simple example that starts the COM server and performs a View operations
08 * on it.
09 * Feel free to extend.
10 */
11 public class RunStyleVision
12 {
13   public static void main(String[] args)
14   {
15     // An instance of the application.
16     Application stylevision = null;
17     // Instead of COM error-handling, use Java exception mechanism.
18     try
19     {
20       // Start StyleVision as COM server.
21       stylevision = new Application();
22       ENUMApplicationStatus status = ENUMApplicationStatus.eApplicationStatus.eApplicationRunning;
23       do{
24         // Check the application status
25         status = stylevision.getStatus();
26         System.out.println("status : " + status + "\n");
27       } while (status != ENUMApplicationStatus.eApplicationRunning);
28  
29  
30  
31  
32  
33  
34  
35  
```
String strExamplesFolder = System.getenv("USERPROFILE") + "\My Documents\Altova\StyleVision2012\StyleVisionExamples\";

// Open two files from the product samples.
stylevision.getDocuments().openDocument(strExamplesFolder + "OrgChart.pxf");
stylevision.getDocuments().openDocument(strExamplesFolder + "BiggestCitiesPerContinent.sps");

// Iterate through all open documents
for (Document doc:stylevision.getDocuments())
{
    System.out.println("Document name : " + doc.getName() + "\n");
    SchemaSources sources = doc.getSchemaSources();

    for (int i = 1; i <= sources.getCount(); i++)
    {
        SchemaSource source = sources.getItem(i);
        System.out.println("\tSchema file name : " + source.getSchemaFileName() + "\n");
        System.out.println("\tWorking XML file name : " + source.getWorkingXMLFileName() + "\n");
        System.out.println("\tIs main schema source : " + source.getIsMainSchemaSource() + "\tType name : " + source.getTypeName() + "\n");
    }
}

// The following lines attach to the document events using a default implementation
// for the events and override one of its methods.
// If you want to override all document events it is better to derive your listener class
// from DocumentEvents and implement all methods of this interface.
Document doc = stylevision.getActiveDocument();
doc.addListener(new DocumentEventsDefaultHandler()
{
    @Override
    public void onDocumentClosed(Document i_ipDoc)
throws AutomationException
    {
        System.out.println("Document " + i_ipDoc.getName() + " requested closing.");
    }
});
doc.close();
doc = null;

System.out.println("Watch StyleVision!");

} catch (AutomationException e)
{
    e.printStackTrace();
}
finally
{
    // Make sure that StyleVision can shut down properly.
    if (stylevision != null)
    stylevision.dispose();

    // Since the COM server was made visible and still is visible, it will
keep running
85       // and needs to be closed manually.
86       System.out.println("Now close StyleVision!");
87   }
88 
89 }

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
// Access general JAVA-COM bridge classes
import com.altova.automation.libs.*;

// Access StyleVision Java-COM bridge
import com.altova.automation.StyleVision.*;
import com.altova.automation.StyleVision.Enums.ENUMApplicationStatus;

/**
  * A simple example that starts the COM server and performs a View operations
  * on it.
  * Feel free to extend.
  */
public class RunStyleVision{
  public static void main(String[] args)
  {
    // An instance of the application.
    Application stylevision = null;
    try
    {
      // Start StyleVision as COM server.
      stylevision = new Application();
      ENUMApplicationStatus status = ENUMApplicationStatus.eApplicationRunning;
      do
      {
        // Check the application status
        status = stylevision.getStatus();
        System.out.println("status : " + status + "\n");
      } while (status != ENUMApplicationStatus.eApplicationRunning);
      // COM servers start up invisible, so we make the server visible
      stylevision.setVisible(true);
    }
  }
}"
```
Application shutdown

The application can be shut down as shown below.

```java
1 { 
2     // Make sure that StyleVision can shut down properly. 
3     if (stylevision != null) 
4         stylevision.dispose(); 
5 
6     // Since the COM server was made visible and still is visible, it will 
7     // keep running 
8     System.out.println("Now close StyleVision!"); 
9 }
```

Simple Document Access

The code listing below shows how to open a document.

```java
1 // Locate samples installed with the product. 
2 String strExamplesFolder = System.getenv("USERPROFILE") + "\My Documents\ 
\Altova\StyleVision2012\StyleVisionExamples\"; 
3 
4 // We open two files form the product samples. 
5 stylevision.getDocuments().openDocument(strExamplesFolder + "OrgChart.pxf"); 
6 stylevision.getDocuments().openDocument(strExamplesFolder + "BiggestCitiesPerContinent.sps");
```

Iterations

The listing below shows how to iterate through open documents.

```java
01 // Iterate through all open documents 
02 for (Document doc:stylevision.getDocuments()) 
03 { 
04     System.out.println("Document name : " + doc.getName() + "\n"); 
05     SchemaSources sources = doc.getSchemaSources(); 
06 
07     for (int i = 1; i <= sources.getCount(); i++) 
08     { 
09         SchemaSource source = sources.getItem(i); 
10         System.out.println("\tSchema file name : " + 
source.getSchemaFileName() + "\n"); 
11         System.out.println("\tWorking XML file name : " + 
source.getWorkingXMLFileName() + "\n"); 
12         System.out.println("\tIs main schema source : " + 
source.getIsMainSchemaSource() + "\tType name : " + source.getTypeName() + 
"\n"); 
13     } 
14 }
```

Event Handlers

The listing below shows how to listen for and use events.
// The following lines attach to the document events using a default implementation
// for the events and override one of its methods.
// If you want to override all document events it is better to derive your listener class.
// from DocumentEvents and implement all methods of this interface.
Document doc = stylevision.getActiveDocument();
doc.addListener(new DocumentEventsDefaultHandler() {
    @Override
    public void onDocumentClosed(Document i_ipDoc) throws AutomationException {
        System.out.println("Document " + i_ipDoc.getName() + " requested closing.");
    }
});
doc.close();
doc = null;
Interfaces

This chapter contains the reference of the StyleVision Type Library.
Application

See also

Methods

Quit

Properties

Application
Parent

ActiveDocument
Documents

Status
MajorVersion
MinorVersion
Edition
IsAPISupported
ServicePackVersion

Description
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

Dim objSpy As Application
Set objSpy = CreateObject("XMLSpy.Application")

Events

OnShutDown

Event: OnShutDown()

Description
Sent just before StyleVision shuts down.

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.
Application

See also

**Property:** Application as Application (read-only)

Description
Accesses the StyleVision application object.

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 StyleVision Enterprise Edition for the Enterprise edition.

Errors
- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

IsAPISupported

See also

**Property:** IsAPISupported as Boolean

Description
Returns whether the API is supported in this version or not.

Errors
- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

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**
- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

**MinorVersion**

**See also**

**Property:** `MinorVersion` as Integer

**Description**
Returns the application version's minor number.

**Errors**
- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

**NewDocument**

**Method:** `NewDocument` as `Document`

**Return Value**
None

**Description**
Creates a new empty document based on the previous template.

**Errors**
- 1000 The application object is invalid.
- 1005 Error when creating a new document
- 1006 Cannot create document

**OpenDocument**

**Method:** `OpenDocument(strFileName as String)` as `Document`

**Return Value**
None

**Description**
Opens an existing SPS file.

**Errors**
- 1000 The application object is invalid.
- 1002 Invalid file extension.
- 1003 Error when opening document.
- 1004 Cannot open document.
Parent

See also

Property: Parent as Application (read-only)

Description
Accesses the StyleVision 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 StyleVision. All modified documents will be closed without saving the changes. This is also true for an open project.

If StyleVision 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.

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.

Status

See also

Property: Status as ENUMApplicationStatus

Description
Returns the current status of the running application.

Errors
Visible

See also

Property: Visible as VARIANT_BOOL

Description
Sets or gets the visibility attribute of StyleVision.

Errors

1111  The application object is no longer valid.
1100  Invalid address for the return parameter was specified.
**AppOutputLine**

 Represents a message line. Its structure is more detailed and can contain a collection of child lines, thereby forming a tree of message lines.

**Properties and Methods**

Properties to navigate the object model:

- Application
- Parent

Line access:

- GetLineSeverity
- GetLineSymbol
- GetLineText
- GetLineTextEx
- GetLineTextWithChildren
- GetLineTextWithChildrenEx

A single AppOutputLine consists of one or more sub-lines. Sub-line access:

- GetLineCount

A sub-line consists of one or more cells. Cell access:

- GetCellCountInLine
- GetCellIcon
- GetCellSymbol
- GetCellText
- GetCellTextDecoration
- GetIsCellText

Below an AppOutputLine there can be zero, one, or more child lines which themselves are of type AppOutputLine, which thus form a tree structure.

Child lines access:

- ChildLines

**Application**

*Property:* Application as Application (read-only)

**Description**

Retrieves the application's top-level object.

**Errors**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The object is no longer valid.</td>
</tr>
<tr>
<td>4101</td>
<td>Invalid address for the return parameter was specified.</td>
</tr>
</tbody>
</table>
ChildLines

*Property:* ChildLines as AppOutputLines (read-only)

*Description*
Returns a collection of the current line's direct child lines.

*Errors*
- 4100  The application object is no longer valid.
- 4101  Invalid address for the return parameter was specified.

GetCellCountInLine

*Method:* GetCellCountInLine (nLine as Long) as Long

*Description*
Gets the number of cells in the sub-line indicated by nLine in the current AppOutputLine.

*Errors*
- 4100  The object is no longer valid.
- 4101  Invalid address for the return parameter was specified.

GetCellIcon

*Method:* GetCellIcon (nLine as Long, nCell as Long) as Long

*Description*
Gets the icon of the cell indicated by nCell in the current AppOutputLine's sub-line indicated by nLine.

*Errors*
- 4100  The object is no longer valid.
- 4101  Invalid address for the return parameter was specified.

GetCellSymbol

*Method:* GetCellSymbol (nLine as Long, nCell as Long) as AppOutputLineSymbol

*Description*
Gets the symbol of the cell indicated by nCell in the current AppOutputLine's sub-line indicated by nLine.

*Errors*
- 4100  The object is no longer valid.
- 4101  Invalid address for the return parameter was specified.

GetCellText

*Method:* GetCellText (nLine as Long, nCell as Long) as String

*Description*
Gets the text of the cell indicated by nCell in the current AppOutputLine's sub-line indicated by nLine.
Errors
   4100  The object is no longer valid.
   4101  Invalid address for the return parameter was specified.

GetCellTextDecoration

Method: GetCellTextDecoration \((nLine\) as Long, \(nCell\) as Long) as Long

Description
Gets the decoration of the text cell indicated by \(nCell\) in the current AppOutputLine's sub-line indicated by \(nLine\).
It can be one of the \texttt{ENUMAppOutputLine\_TextDecoration} values.

Errors
   4100  The object is no longer valid.
   4101  Invalid address for the return parameter was specified.

GetIsCellText

Method: GetIsCellText \((nLine\) as Long, \(nCell\) as Long) as String

Description
Gets the text of the cell indicated by \(nCell\) in the current AppOutputLine's sub-line indicated by \(nLine\).

Errors
   4100  The object is no longer valid.
   4101  Invalid address for the return parameter was specified.

GetLineCount

Method: GetLineCount () as Long

Description
Gets the number of sub-lines the current line consists of.

Errors
   4100  The object is no longer valid.
   4101  Invalid address for the return parameter was specified.

GetLineSeverity

Method: GetLineSeverity () as Long

Description
Gets the severity of the line. It can be one of the \texttt{ENUMAppOutputLine\_Severity} values:

Errors
   4100  The object is no longer valid.
   4101  Invalid address for the return parameter was specified.
GetLineSymbol

**Method:** GetLineSymbol () as AppOutputLineSymbol

**Description**
Gets the symbol assigned to the whole line.

**Errors**
- 4100  The object is no longer valid.
- 4101  Invalid address for the return parameter was specified.

GetLineText

**Method:** GetLineText () as String

**Description**
Gets the contents of the line as text.

**Errors**
- 4100  The object is no longer valid.
- 4101  Invalid address for the return parameter was specified.

GetLineTextEx

**Method:** GetLineTextEx (psTextPartSeparator as String, psLineSeparator as String) as String

**Description**
Gets the contents of the line as text using the specified part and line separators.

**Errors**
- 4100  The object is no longer valid.
- 4101  Invalid address for the return parameter was specified.

GetLineTextWithChildren

**Method:** GetLineTextWithChildren () as String

**Description**
Gets the contents of the line including all child and descendant lines as text.

**Errors**
- 4100  The object is no longer valid.
- 4101  Invalid address for the return parameter was specified.

GetLineTextWithChildrenEx

**Method:** GetLineTextWithChildrenEx (psPartSep as String, psLineSep as String, psTabSep as String, psItemSep as String) as String

**Description**
Gets the contents of the line including all child and descendant lines as text using the specified part, line, tab and item separators.
Errors
   4100  The object is no longer valid.
   4101  Invalid address for the return parameter was specified.

Property: Parent as AppOutputLines (read-only)

Description
The parent object according to the object model.

Errors
   4100  The object is no longer valid.
   4101  Invalid address for the return parameter was specified.
AppOutputLines

Represents a collection of AppOutputLine message lines.

Properties and Methods
Properties to navigate the object model:
- Application
- Parent

Iterating through the collection:
- Count
- Item

Application

Property: Application as Application (read-only)

Description
Retrieves the application's top-level object.

Errors
- 4000 The object is no longer valid.
- 4001 Invalid address for the return parameter was specified.

Count

Property: Count as Integer (read-only)

Description
Retrieves the number of lines in the collection.

Errors
- 4000 The object is no longer valid.
- 4001 Invalid address for the return parameter was specified.

Item

Property: Item (nIndex as Integer) as AppOutputLine (read-only)

Description
Retrieves the line at nIndex from the collection. Indices start with 1.

Errors
- 4000 The object is no longer valid.
- 4001 Invalid address for the return parameter was specified.

Parent

Property: Parent as AppOutputLine (read-only)

Description
The parent object according to the object model.

Errors
4000  The object is no longer valid.
4001  Invalid address for the return parameter was specified.
AppOutputLineSymbol

An AppOutputLineSymbol represents a link in an AppOutputLine message line which can be clicked in the StyleVision Messages window. It is applied to a cell of an AppOutputLine or to the whole line itself.

Properties and Methods
Properties to navigate the object model:

Application
Parent

Access to AppOutputLineSymbol methods:

GetSymbolHREF
GetSymbolID
IsSymbolHREF

Application

Property: Application as Application (read-only)

Description
Retrieves the application's top-level object.

Errors

4200 The object is no longer valid.
4201 Invalid address for the return parameter was specified.

GetSymbolHREF

Method: GetSymbolHREF () as String

Description
If the symbol is of type URL, returns the URL as a string.

Errors

4200 The object is no longer valid.
4201 Invalid address for the return parameter was specified.

GetSymbolID

Method: GetSymbolHREF () as Long

Description
Gets the ID of the symbol.

Errors

4200 The object is no longer valid.
4201 Invalid address for the return parameter was specified.
IsSymbolHREF

Method: IsSymbolHREF() as Boolean

Description
Indicates if the symbol is of kind URL.

Errors
4200  The object is no longer valid.
4201  Invalid address for the return parameter was specified.

Parent

Property: Parent as Application (read-only)

Description
The parent object according to the object model.

Errors
4200  The object is no longer valid.
4201  Invalid address for the return parameter was specified.
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
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.

```
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
**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>MoveBegin</td>
</tr>
<tr>
<td>FirstXMLData</td>
<td>MoveEnd</td>
</tr>
<tr>
<td>FirstXMLDataOffset</td>
<td>NextCursorPosition</td>
</tr>
<tr>
<td>LastTextPosition</td>
<td>Select</td>
</tr>
<tr>
<td>LastXMLData</td>
<td>SelectNext</td>
</tr>
<tr>
<td>LastXMLDataOffset</td>
<td>SelectPrevious</td>
</tr>
<tr>
<td>Parent</td>
<td>SetFromRange</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>GetElementAttributeNames</td>
<td></td>
</tr>
<tr>
<td>PerformAction</td>
<td>GetElementHierarchy</td>
<td>InsertEntity</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-only 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 StyleVision 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

' ---------------------------------------------------------
' 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 text 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.

HI NT: Use text cursor positions with care, since this is a costly operation compared to XMLData based cursor positioning.

**Errors**
The authentic range object, or its related view object is not valid.
Invalid address for the return parameter was specified.
A text position outside the document was specified.

Examples
'D ------------------------------
'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 "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

' -----------------------------------------------
' 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 FirstXMLData/FirstXMLDataOffset selects a position after the current 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.
Invalid address for the return parameter was specified.

Examples

' -----------------------------------------------
' 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: 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.

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
Goto

See also

Method: Goto (eKind as SPLYAuthenticElementKind, nCount as Long, eFrom as SPLYAuthenticDocumentPosition) 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.
- 2003 Target lies after end of document.
- 2004 Target lies before begin of document.
- 2005 Invalid element kind specified.
- 2006 Variable cannot be evaluated
- 2009 Invalid parameter

GotoNext

See also

Method: GotoNext (eKind as SPLYAuthenticElementKind) 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.
- 2003 Target lies after end of document.
- 2005 Invalid element kind specified.
- 2006 Variable returns sequence
- 2009 Invalid parameter

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
Wend
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.
2005 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
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**

```vbnet
' ---------------------------------------------------------
' 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**

```vbnet
' -------------------------------------------------------------
' 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
```
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
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 than 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.
- **2005**: Invalid offset was specified.
- **2009**: Invalid address for the return parameter was specified.

**Examples**

```vbnet
' ---------------------------------------------
' 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 "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**
- 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 beginning of document.
- 2005 Invalid element kind specified.
  Invalid address for the return parameter was specified.

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.
2005 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 spyAuthenticationInsertBefore, arrElements

' we insert the first (innermost) element
If UBound(arrElements) >= 0 Then
    objRange.PerformAction spyAuthenticationInsertBefore, 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 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 - 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.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

Method: GotoPrevious (eKind as SPYAuthenticElementKind) as AuthenticRange

Description
Selects the element of type 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 bBeginOfDocument
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

See also

SetElementAttribute

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() As String
    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.
2204 No such variable in scope
2205 Variable cannot be evaluated
2206 Variable returns sequence
2207 Variable read-only
2208 No modification allowed

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.
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></td>
</tr>
<tr>
<td>Parent</td>
<td>UpdateXMLInstanceEntities</td>
<td></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.

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:

function On_AuthenticBeforeCopy()
{
    // return false; /* to disable operation */
}
OnBeforeCut

See also

Event: OnBeforeCut() as Boolean

Scripting environment - VBScript:
Function On_AuthenticBeforeCut()
    ' On_AuthenticBeforeCut = False ' to disable operation
End Function

Scripting environment - JScript:
function On_AuthenticBeforeCut()
{
    // return false; /* to disable operation */
}

OnBeforeDelete

See also

Event: OnBeforeDelete() as Boolean

Scripting environment - VBScript:
Function On_AuthenticBeforeDelete()
    ' On_AuthenticBeforeDelete = False ' to disable operation
End Function

Scripting environment - JScript:
function On_AuthenticBeforeDelete()
{
    // 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 */
}

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 cancel the drop operation. Return `True` (or nothing) to continue normal operation.

### OnBeforePaste

**See also**

**Event:** `OnBeforePaste` *(objData as Variant, strType as String) as Boolean*

**Scripting environment - VBScript:**

```vb
Function On_AuthenticBeforePaste(objData, strType)
  ' On_AuthenticBeforePaste = False ' to disable operation
End Function
```

**Scripting environment - JScript:**

```javascript
function On_AuthenticBeforePaste(objData, strType)
{
  // return false; /* to disable operation */
}
```

**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. (Please note, that the event fires when saving the XML document, and not when saving the SPS design in StyleVision.)

### 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.DisableViewElementEntryHelper();
}
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.

OnMouseEventHandler

See also

Event: OnMouseEventHandler (nXPos as Long, nYPos as Long, eMouseEvent as SPYMouseEvent, objRange as AuthenticRange) as Boolean

Scripting environment - VBScript:
Function On_AuthenticMouseEventHandler (nXPos, nYPos, eMouseEvent, objRange)
   ' On_AuthenticMouseEventHandler = True ' to cancel bubbling of event
End Function

Scripting environment - JScript:
function On_AuthenticMouseEventHandler (nXPos, nYPos, eMouseEvent, objRange)
{
   // return true; /* to cancel bubbling of event */
}

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) {
}

**Description**
This event gets triggered whenever the selection in the user interface changes.

**OnToolbarButtonClicked**

**Event:** OnToolbarButtonClicked (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. 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
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 StyleVision 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.
Invalid element kind specified.
The document position to start from is not one of spyAuthenticDocumentBegin or 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 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 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 at 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**

```javascript
// -----------------------------------------
// 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.
Document

The Document interface has the following methods and properties:

Methods

- **Activate**
- **AssignWorkingXMLFile**
- **Close**
- **Save**
- **SaveAs**
- **Saved**
- **SaveGeneratedFOFile**
- **SaveGeneratedFOFileEx**
- **SaveGeneratedHTMLFile**
- **SaveGeneratedHTMLFileEx**
- **SaveGeneratedPDFFile**
- **SaveGeneratedPDFFileEx**
- **SaveGeneratedRTFFile**
- **SaveGeneratedRTFFileEx**
- **SaveGeneratedWord2007File**
- **SaveGeneratedWord2007FileEx**
- **SaveGeneratedXSLTOFile**
- **SaveGeneratedXSLTOFileEx**
- **SaveGeneratedXSLTHTMLFile**
- **SaveGeneratedXSLTHTMLFileEx**
- **SaveGeneratedXSLTRTFFile**
- **SaveGeneratedXSLTRTFFileEx**
- **SaveGeneratedXSLTWord2007File**
- **SaveGeneratedXSLTWord2007FileEx**

Properties

- **Application**
- **FullName**
- **Name**
- **Parameters**
- **Parent**
- **Path**
- **SchemaSources**

Events

OnDocumentClosed

**Event:** `OnDocumentClosed(Document as Document)`

**Description**

This event gets fired as a result of closing a document.

OnModifiedFlagChanged

**Event:** `OnModifiedFlagChanged(as Boolean)`
Description
Returns true if the Modified flag has been changed.

Activate

Method: Activate()

Description
Activate document frame.

Errors
- 1200 Document object is invalid.
- 1201 Invalid input parameter.

Application

See also

Property: Application as Application (read-only)

Description
Accesses the StyleVision application object.

Errors
- 140011 The object is no longer valid.
- 11
- 140711 Invalid address for the return parameter was specified.
- 00

AssignWorkingXMLFile

Method: AssignWorkingXMLFile (strWorkingXMLFileName as String, strSchemaSourceName as String)

Description
Assigns the Working XML File by supplying its URI as a string and the Schema Source Name as a string. The Schema Source Name is the name assigned to the schema in the SPS; (the default schema name in an SPS created new in StyleVision is: XML).

Errors
- 1200 The document object is invalid.
- 1201 Invalid input parameter.
- 1203 Error assigning Working XML File
- 1404 Missing XML Schema or DTD

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.

**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.

**GetPathName (obsolete)**

*Superseded by **Document.FullName***

```javascript
// ----- javascript sample -----  
// instead of:
// strPathName = Application.ActiveDocument.GetPathName();
// use now:  
strPathName = Application.ActiveDocument.FullName;
```

*See also*

*Method: GetPathName() as String*

**Description**
The method **GetPathName** gets the path of the active document.

See also **Document.SetPathName** (obsolete).

**IsValid**

*See also*

*Method: IsValid (strError as Variant) as Boolean*

**Return Value**
True if the document is valid, false if not.

**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.

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.
Parameters

**Property:** Parameters as Parameters (read-only)

**Description**
Reference to the current Parameters object.

**Errors**
- 1200 Document object is invalid.
- 1201 Invalid input parameter.

Parent

**Description**
Access the StyleVision application object.

**Errors**
- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

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.

Save

**See also**

**Method:** Save()

**Description**
The method writes any modifications of the document to the associated file.

**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.

**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.

**Errors**
- 1400 The document object is no longer valid.
- 1407 Invalid address for the return parameter was specified.

SaveGeneratedFOFile

**Method:** **SaveGeneratedFOFile** *(strFileName as String)*

**Description**
Saves the generated file to the location specified.

**Errors**
- 1201 Invalid document object.
- 1204 Cannot generate output file.

SaveGeneratedFOFileEx

**Method:** **SaveGeneratedFOFileEx** *(strFileName as String, pbError as Variant) as AppOutputLines*

**Description**
Saves the generated file to the location specified and gives error description.

**Errors**
- 1201 Invalid document object.
- 1204 Cannot generate output file.

SaveGeneratedHTMLFile

**Method:** **SaveGeneratedHTMLFile** *(strFileName as String)*

**Description**
Saves the generated file to the location specified.
SaveGeneratedHTMLFileEx

**Method:** SaveGeneratedHTMLFileEx *(strFileName as String, pbError as Variant) as AppOutputLines*

**Description**
Saves the generated file to the location specified and gives error description.

**Errors**
- 1201 Invalid document object.
- 1204 Cannot generate output file.

SaveGeneratedPDFFile

**Method:** SaveGeneratedPDFFile *(strFileName as String)*

**Description**
Saves the generated file to the location specified.

**Errors**
- 1201 Invalid document object.
- 1204 Cannot generate output file.

SaveGeneratedPDFFileEx

**Method:** SaveGeneratedPDFFileEx *(strFileName as String, pbError as Variant) as AppOutputLines*

**Description**
Saves the generated file to the location specified and gives error description.

**Errors**
- 1201 Invalid document object.
- 1204 Cannot generate output file.

SaveGeneratedRTFFile

**Method:** SaveGeneratedRTFFile *(strFileName as String)*

**Description**
Saves the generated file to the location specified.

**Errors**
- 1201 Invalid document object.
- 1204 Cannot generate output file.
$\text{SaveGeneratedRTFFileEx}$

**Method:** $\text{SaveGeneratedRTFFileEx} (strFileName \text{ as String}, pbError \text{ as Variant})$ as AppOutputLines

**Description**
Saves the generated file to the location specified and gives error description.

**Errors**
- 1201 Invalid document object.
- 1204 Cannot generate output file.

$\text{SaveGeneratedWord2007File}$

**Method:** $\text{SaveGeneratedWord2007File} (strFileName \text{ as String})$

**Description**
Saves the generated file to the location specified.

**Errors**
- 1201 Invalid document object.
- 1204 Cannot generate output file.

$\text{SaveGeneratedWord2007FileEx}$

**Method:** $\text{SaveGeneratedWord2007FileEx} (strFileName \text{ as String}, pbError \text{ as Variant})$ as AppOutputLines

**Description**
Saves the generated file to the location specified and gives error description.

**Errors**
- 1201 Invalid document object.
- 1204 Cannot generate output file.

$\text{SaveGeneratedXSLTFOFile}$

**Method:** $\text{SaveGeneratedXSLTFOFile} (strFileName \text{ as String})$

**Description**
Saves the generated file to the location specified.

**Errors**
- 1201 Invalid document object.
- 1204 Cannot generate output file.

$\text{SaveGeneratedXSLTFOFileEx}$

**Method:** $\text{SaveGeneratedXSLTFOFileEx} (strFileName \text{ as String}, pbError \text{ as Variant})$ as AppOutputLines

**Description**
Saves the generated file to the location specified and gives error description.
Errors
1201 Invalid document object.
1204 Cannot generate output file.

SaveGeneratedXSLTHTMLFile

*Method:* `SaveGeneratedXSLTHTMLFile(strFileName as String)`

*Description*
Saves the generated file to the location specified.

Errors
1201 Invalid document object.
1204 Cannot generate output file.

SaveGeneratedXSLTHTMLFileEx

*Method:* `SaveGeneratedXSLTHTMLFileEx(strFileName as String, pbError as Variant)` as `AppOutputLines`

*Description*
Saves the generated file to the location specified and gives error description.

Errors
1201 Invalid document object.
1204 Cannot generate output file.

SaveGeneratedXSLTRTFFile

*Method:* `SaveGeneratedXSLTRTFFile(strFileName as String)`

*Description*
Saves the generated file to the location specified.

Errors
1201 Invalid document object.
1204 Cannot generate output file.

SaveGeneratedXSLTRTFFileEx

*Method:* `SaveGeneratedXSLTRTFFileEx(strFileName as String, pbError as Variant)` as `AppOutputLines`

*Description*
Saves the generated file to the location specified and gives error description.

Errors
1201 Invalid document object.
1204 Cannot generate output file.
SaveGeneratedXSLTWord2007File

**Method:** SaveGeneratedXSLTWord2007File *(strFileName as String)*

**Description**
Saves the generated file to the location specified.

**Errors**
- 1201 Invalid document object.
- 1204 Cannot generate output file.

SaveGeneratedXSLTWord2007FileEx

**Method:** SaveGeneratedXSLTWord2007FileEx *(strFileName as String, pbError as Variant) as AppOutputLines*

**Description**
Saves the generated file to the location specified and gives error description.

**Errors**
- 1201 Invalid document object.
- 1204 Cannot generate output file.

SchemaSources

**Property:** SchemaSources as SchemaSources *(read-only)*

**Description**
Reference to the current SchemaSources object.

**Errors**
- 1200 Document object is invalid.
- 1201 Invalid input parameter.

SetPathName (obsolete)

**Superseded by** Document.FullName

```javascript
// ----- 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.
Documents

The Documents interface has the following methods and properties:

Methods

- **ActiveDocument**
- **Item**
- **NewDocument**
- **OpenDocument**

Properties

- **Application**
- **Count**
- **Parent**

ActiveDocument

*Property:* `ActiveDocument` as `Document`

Description

Reference to the active document. If no document is open, `ActiveDocument` is null (nothing).

Errors

<table>
<thead>
<tr>
<th>Error 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>
<tr>
<td>1600</td>
<td>Invalid Documents object</td>
</tr>
<tr>
<td>1601</td>
<td>Invalid input parameter</td>
</tr>
</tbody>
</table>

Application

*Property:* `Application` as `Application` (read-only)

Description

Access the StyleVision application object.

Errors

<table>
<thead>
<tr>
<th>Error 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>

Count

See also

*Property:* `Count` as `long`

Description

Count of open documents.

Errors

<table>
<thead>
<tr>
<th>Error 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**
- 1600 Invalid Documents object
- 1601 Invalid input parameter

NewDocument

*Method:* NewDocument() as Document

**Return Value**
None

**Description**
Creates a new empty document based on the previous template.

**Errors**
- 1000 The application object is invalid.
- 1005 Error when creating a new document
- 1006 Cannot create document
- 1600 Invalid Documents object
- 1601 Invalid input parameter

OpenDocument

*Method:* OpenDocument(strFileName as String) as Document

**Return Value**
None

**Description**
Opens an existing SPS file.

**Errors**
- 1000 The application object is invalid.
- 1002 Invalid file extension.
- 1003 Error when opening document.
- 1004 Cannot open document.
- 1600 Invalid Documents object
- 1601 Invalid input parameter
Parent

**Property:** Parent as Application (read-only)

**Description**
Access the StyleVision application object.

**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>
Parameter

The Parameter interface has the following properties:

Properties

- Application
- Name
- Parent
- Value

Application

Property: Application as Application (read-only)

Description
Access the StyleVision application object.

Errors

1111 The application object is no longer valid.
1100 Invalid address for the return parameter was specified.

Name

Property: Name as String (read-only)

Errors

1600 Invalid Documents object
1601 Invalid input parameter

Parent

Property: Parent as Application (read-only)

Description
Access the StyleVision application object.

Errors

1111 The application object is no longer valid.
1100 Invalid address for the return parameter was specified.

Value

Property: Value as String (read-only)

Errors

1600 Invalid Documents object
1601 Invalid input parameter
Parameters

The Parameters interface has the following properties:

Properties

- **Application**
- **Count**
- **Item**
- **Parent**

**Application**

*Property:* `Application` as `Application` (read-only)

*Description*

Access the StyleVision application object.

*Errors*

- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

**Count**

*Property:* `Count` as long

*Description*

Count of parameters.

*Errors*

- 1600 Invalid Parameters object
- 1601 Invalid input parameter

**Item**

*Method:* `Item` (n as long) as `Document`

*Description*

Gets the document with the index n in this collection. Index is 1-based.

*Errors*

- 1600 Invalid Documents object
- 1601 Invalid input parameter

**Parent**

*Property:* `Parent` as `Application` (read-only)

*Description*

Access the StyleVision application object.

*Errors*

- 1111 The application object is no longer valid.
1100 Invalid address for the return parameter was specified.
SchemaSource

The SchemaSource interface has the following properties:

Properties

- **Application**
- **IsMainSchemaSource**
- **Name**
- **Parent**
- **SchemaFileName**
- **TemplateFileName**
- **Type**
- **TypeName**
- **WorkingXMLFileName**

**Application**

*Property:* Application as Application (read-only)

**Description**
Access the StyleVision application object.

**Errors**
- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

**IsMainSchemaSource**

*Property:* IsMainSchemaSource as Boolean (read-only)

**Description**
Returns true if schema source is the main schema source.

**Errors**
- 1400 Invalid schema source object.
- 1401 Invalid parameter.

**Name**

*Property:* Name as String (read-only)

**Description**
Use this property to retrieve the name of the schema source.

**Errors**
- 1400 The schema source object is not valid.
- 1401 Invalid parameter.

**Parent**

*Property:* Parent as SchemaSource (read-only)
Description
The parent object according to the object model.

Errors
1111  The application object is no longer valid.
1100  Invalid address for the return parameter was specified.

SchemaFileName

Property: SchemaFileName as String

Description
Use this property to retrieve the name of the Schema File.

Errors
1400  Invalid schema source object.
1401  Invalid parameter.
1403  Missing XMLSchema or DTD
1406  Error assigning schema File

TemplateFileName

Property: TemplateFileName as String

Description
Use this property to retrieve the name of the Working XML File.

Errors
1400  Invalid schema source object.
1401  Invalid parameter.
1403  Missing XMLSchema or DTD
1407  Error assigning Template XML File

Type

Property: Type as ENUMSchemaSourceType (read-only)

Description
Use this property to retrieve the type of the schema source.

Errors
1400  Invalid schema source object.
1401  Invalid parameter.

TypeName

Property: TypeName as String (read-only)

Description
Use this property to retrieve the type of the schema source.

Errors
1400  Invalid schema source object.
1401  Invalid parameter.

WorkingXMLFileName

Property: WorkingXMLFileName as String

Description
Use this property to retrieve the name of the Working XML File.

Errors
1400  Invalid schema source object.
1401  Invalid parameter.
1403  Missing XMLSchema or DTD
1203  Error assigning Working XML File
The `SchemaSources` interface has the following properties:

**Properties**

- **Application**
- **MainSchemaSource**
- **Parent**

**Application**

**Property:** `Application` as `Application` (read-only)

**Description**

Access the StyleVision application object.

**Errors**

- 1111  The application object is no longer valid.
- 1100  Invalid address for the return parameter was specified.

**Count**

**Property:** `Count` as long (read-only)

**Description**

Count of schema sources.

**Errors**

- 1300  Invalid `SchemaSources` object
- 1301  Invalid input parameter

**Item**

**Method:** `Item` (n as long) as `Document`

**Description**

Gets the document with the index n in this collection. Index is 1-based.

**Errors**

- 1300  Invalid `SchemaSources` object
- 1301  Invalid input parameter

**MainSchemaSource**

**Property:** `MainSchemaSource` as `SchemaSource` (read-only)

**Errors**

- 1300  Invalid `SchemaSources` object
- 1301  Invalid input parameter
Parent

**Property:** Parent as SchemaSources (read-only)

**Description**
The parent object according to the object model.

**Errors**
- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.
**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.

**appendChild**

**See also**

**Declaration:** `appendChild (pNewData as XMLData)`

**Description**

`appendChild` appends `pNewData` as last child to the `XMLData` object.

**Errors**

- 1500 The XMLData object is no longer valid.
- 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

**Example**

```vbnet
Dim objCurrentParent As XMLData
Dim objNewChild As XMLData

Set objNewChild = objSpy.ActiveDocument.CreateChild(spyXMLDataElement)
Set objCurrentParent = objSpy.ActiveDocumentRootElement

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**

- 1500 The XMLData object is no longer valid.
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.

```vba
    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 usefull 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 XLMD ata)

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.
Enumerations

This is a list of all enumerations used by the StyleVision API. If your scripting environment does not support enumerations use the number-values instead.
ENUMApplicationStatus

Description
Enumeration to specify the current Application status.

Possible values:

eApplicationRunning = 0
eApplicationAfterLicenseCheck = 1
eApplicationBeforeLicenseCheck = 2
eApplicationConcurrentLicenseCheckFailed = 3
eApplicationProcessingCommandLine = 4
ENUMAppOutputLine_Severity

Description
Enumeration values to identify the severity of an AppOutputLine.

Possible values:

- eSeverity_Undefined = -1
- eSeverity_Info = 0
- eSeverity_Warning = 1
- eSeverity_Error = 2
- eSeverity_CriticalError = 3
- eSeverity_Success = 4
- eSeverity_Summary = 5
- eSeverity_Progress = 6
- eSeverity_DataEdit = 7
- eSeverity_ParserInfo = 8
- eSeverity_PossibleInconsistencyWarning = 9
- eSeverity_Message = 10
- eSeverity_Document = 11
- eSeverity_Rest = 12
- eSeverity_NoSelect = 13
- eSeverity_Select = 14
- eSeverity_Autoinsertion = 15
- eSeverity_GlobalResources_DefaultWarning = 16
**ENUMAppOutputLine_TextDecoration**

**Description**
Enumeration values for the different kinds of text decoration of an AppOutputLine.

**Possible values:**
- eTextDecorationDefault = 0
- eTextDecorationBold = 1
- eTextDecorationDebugValues = 2
- eTextDecorationDB_ObjectName = 3
- eTextDecorationDB_ObjectLink = 4
- eTextDecorationDB_ObjectKind = 5
- eTextDecorationDB_TimeoutValue = 6
- eTextDecorationFind_MatchingString = 7
- eTextDecorationValidation_Speclink = 8
- eTextDecorationValidation_ErrorPosition = 9
- eTextDecorationValidation_UnknownParam = 10
**ENUMSchemaSourceType**

**Description**
Enumeration to specify the source schema type: XML Schema, DTD, DB, DB cell, User-Defined, or XBRL.

**Possible values:**

- eSchemaSourceType_XSDorDTD = 0
- eSchemaSourceType_DB = 1
- eSchemaSourceType_DBCell = 2
- eSchemaSourceType_User = 3
- eSchemaSourceType_XBRL = 4
ENUMSchemaType

Description
Enumeration to specify the schema type: W3C XML Schema or DTD.

Possible values:

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eSchemaTypeW3CSchema</td>
<td>0</td>
</tr>
<tr>
<td>eSchemaTypeDTD</td>
<td>1</td>
</tr>
</tbody>
</table>
**SPYAuthenticActions**

**Description**
Actions that can be performed on `AuthenticRange` objects.

**Possible values:**

- `spyAuthenticInsertAt` = 0
- `spyAuthenticApply` = 1
- `spyAuthenticClearSurround` = 2
- `spyAuthenticAppend` = 3
- `spyAuthenticInsertBefore` = 4
- `spyAuthenticRemove` = 5
**SPYAuthenticDocumentPosition**

**Description**
Relative and absolute positions used for navigating with AuthenticRange objects.

**Possible values:**
- spyAuthenticDocumentBegin = 0
- spyAuthenticDocumentEnd = 1
- spyAuthenticRangeBegin = 2
- spyAuthenticRangeEnd = 3
**SPYAuthenticateElementKind**

**Description**
Enumeration of the different kinds of elements used for navigation and selection within the `AuthenticRange` and `AuthenticView` objects.

**Possible values:**
- `spyAuthenticateChar` = 0
- `spyAuthenticateWord` = 1
- `spyAuthenticateLine` = 3
- `spyAuthenticateParagraph` = 4
- `spyAuthenticateTag` = 6
- `spyAuthenticateDocument` = 8
- `spyAuthenticateTable` = 9
- `spyAuthenticateTableRow` = 10
- `spyAuthenticateTableColumn` = 11
**SPYAuthenticMarkupVisibility**

**Description**
 Hưng quốcb dùng để thiết lập tính ẩn của markup với `MarkupVisibility`.

**Possible values:**
- spyAuthenticMarkupHidden = 0
- spyAuthenticMarkupSmall = 1
- spyAuthenticMarkupLarge = 2
- spyAuthenticMarkupMixed = 3
**SPYAuthenticToolbarButtonState**

**Description**
Authentic toolbar button states are given by the following enumeration:

**Possible values:**
- `authenticToolbarButtonDefault` = 0
- `authenticToolbarButtonEnabled` = 1
- `authenticToolbarButtonDisabled` = 2
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
spyButtonUpMask = 16
spyButtonDownMask = 32
spyDoubleClickMask = 64
spyShiftKeyDownMask = 128
spyCtrlKeyDownMask = 256
spyLeftButtonDownMask = 34 // spyLeftButtonMask | spyButtonDownMask
spyMiddleButtonDownMask = 36 // spyMiddleButtonMask | spyButtonDownMask
spyRightButtonDownMask = 40 // spyRightButtonMask | spyButtonDownMask
spyLeftButtonUpMask = 18 // spyLeftButtonMask | spyButtonUpMask
spyMiddleButtonUpMask = 20 // spyMiddleButtonMask | spyButtonUpMask
spyRightButtonUpMask = 24 // spyRightButtonMask | spyButtonUpMask
spyLeftDoubleClickMask = 66 // spyRightButtonMask | spyButtonUpMask
spyMiddleDoubleClickMask = 68 // spyMiddleButtonMask | spyDoubleClickMask
spyRightDoubleClickMask = 72 // spyRightButtonMask | spyDoubleClickMask
**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 `@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
**SPYValidateErrorFormat**

**Description**
Enumeration values that select the format of the error message.

**Possible values**

- `spyValidateErrorFormat_Text` = 0
- `spyValidateErrorFormat_ShortXML` = 1
- `spyValidateErrorFormat_LongXML` = 2
**SPYXMLDataKind**

**Description**
The different types of XMLData elements available for XML documents.

**Possible values:**

<table>
<thead>
<tr>
<th>XMLData Structure</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>spyXMLDataXMLDocStruct</td>
<td>0</td>
</tr>
<tr>
<td>spyXMLDataXMLEntityDocStruct</td>
<td>1</td>
</tr>
<tr>
<td>spyXMLDataDTDDocStruct</td>
<td>2</td>
</tr>
<tr>
<td>spyXMLDataXML</td>
<td>3</td>
</tr>
<tr>
<td>spyXMLDataElement</td>
<td>4</td>
</tr>
<tr>
<td>spyXMLDataAttr</td>
<td>5</td>
</tr>
<tr>
<td>spyXMLDataText</td>
<td>6</td>
</tr>
<tr>
<td>spyXMLDataCData</td>
<td>7</td>
</tr>
<tr>
<td>spyXMLDataComment</td>
<td>8</td>
</tr>
<tr>
<td>spyXMLDataPI</td>
<td>9</td>
</tr>
<tr>
<td>spyXMLDataDefDoctype</td>
<td>10</td>
</tr>
<tr>
<td>spyXMLDataDefExternalID</td>
<td>11</td>
</tr>
<tr>
<td>spyXMLDataDefElement</td>
<td>12</td>
</tr>
<tr>
<td>spyXMLDataDefAttlist</td>
<td>13</td>
</tr>
<tr>
<td>spyXMLDataDefEntity</td>
<td>14</td>
</tr>
<tr>
<td>spyXMLDataDefNotation</td>
<td>15</td>
</tr>
<tr>
<td>spyXMLDataKindsCount</td>
<td>16</td>
</tr>
</tbody>
</table>
21.3 ActiveX Integration

StyleVisionControl is a control that provides a means of integration of the StyleVision user interface and the functionality described in this section into most kinds of applications. ActiveX technology was chosen so as to allow integration using any of a wide variety of languages; this enables C++, C#, VisualBasic, or HTML to be used for integration (ActiveX components integrated in HTML officially only work with Microsoft Internet Explorer). The attached Java wrapper library allows integration into Java. All components are full OLE Controls, which makes integration as simple as possible. Two different levels of integration are provided, thus enabling the integration to be adapted to a wide range of needs.

To integrate StyleVision you must install the StyleVision Integration Package. Ensure that you install StyleVision first, and then the StyleVision Integration Package.

For a successful integration you have to consider the following main design factors:

- What technology or programming language can the hosting application use to integrate the StyleVisionControl?
- Should the integrated UI look exactly like StyleVision with all its menus, toolbars, and windows, or will a subset of these elements—like allowing only one document and a restricted set of commands—be more effective?
- How deep will the integration be? Should the StyleVision user interface be used as is? Are user interface extensions and/or restrictions required? Can some frequently used tasks be automated?

The sections, Integration at the Application Level and Integration at Document Level, describe the key steps at these respective levels. The Programming Languages section provides examples in C#, HTML, and Java. Looking through these examples will help you to make the right decisions quickly. The section, Object Reference, describes all COM objects that can be used for integration, together with their properties and methods.

For automation tasks, the StyleVision Automation Interface is accessible from the StyleVisionControl as well.

For information about how to integrate StyleVision into Microsoft Visual Studio see the section, StyleVision in Visual Studio.
Integration at Application Level

Integration at application level is simple and straightforward. It allows you to embed the complete interface of StyleVision into a window of your application. Since you get the whole user interface of StyleVision, you get all menus, toolbars, the status bar, document windows, and helper windows. Customization of the application’s user interface is restricted to what StyleVision provides. This includes rearrangement and resizing of helper windows and customization of menus and toolbars.

The only ActiveX control you need to integrate is `StyleVisionControl`. Its property `IntegrationLevel` defaults to application-level. You may use `Appearance` and `BorderStyle` to configure the appearance of the control's wrapper window. Do not instantiate or access `StyleVisionControlDocument` or `StyleVisionControlPlaceHolder` ActiveX controls when integrating at application-level.

If you have any initialization to do or if you want to automate some behaviour of StyleVision, use the properties, methods, and events described for `StyleVisionControl`. Consider using `StyleVisionControl.Application` for more complex access to StyleVision functionality.

In the Programming Languages | HTML section is an example (Integration at Application Level) that shows how the StyleVision application can be embedded in an HTML page. For usage with other programming languages, or more sophisticated access, see the Examples of integration at document-level (in C#, HTML, and Java).
**Integration at Document Level**

Integration at document level gives you freedom over instantiation and placement of the following parts of the StyleVision user interface:

If necessary, a replacement for the menus and toolbars of StyleVision must be provided by your application.

You will need to instantiate and access multiple ActiveX controls, depending on which user interface parts you want to re-use. All these controls are contained in the StyleVisionControl OCX.

- Use **StyleVisionControl** to set the integration level and access application wide functionality.
- Use **StyleVisionControlDocument** to create any number of editor windows. It may be sufficient to create only one window and re-use it, depending on your needs.
- Optionally use **StyleVisionControlPlaceholder** to embed StyleVision entry helper windows, validator output or other windows mentioned above.
- Access run-time information about commands, menus, and toolbars available in **StyleVisionControl** to seamlessly integrate these commands into your application's menus and toolbars. See **Query StyleVision Commands** for more information.

If you want to automate some behaviour of StyleVision use the properties, methods, and events described for the **StyleVisionControl**, **StyleVisionControlDocument** and **StyleVisionControlPlaceholder**. Consider using **StyleVisionControl.Application**, **StyleVisionControlDocument.Document** and **StyleVisionControlPlaceholder.Project** for more complex access to StyleVision functionality. However, to open a document always use **StyleVisionControlDocument.Open** or **StyleVisionControlDocument.New** on the appropriate document control. To open a project always use **StyleVisionControlPlaceholder.OpenProject** on a placeholder control embedding a StyleVision project window.

See **Examples** on how to instantiate and access the necessary controls in different programming environments.
Use StyleVisionControl

To integrate at document level, instantiate a `StyleVisionControl` first. Set the property `IntegrationLevel` to `ICActiveXIntegrationOnDocumentLevel (= 1)`. Set the window size of the embedding window to 0x0 to hide any user interface behind the control. You may use `Appearance` and `BorderStyle` to configure the appearance of the control's wrapper window.

Avoid using the method `Open` since this might lead to unexpected results. Use the corresponding open methods of `StyleVisionControlDocument` and `StyleVisionControlPlaceholder`, instead.

See `Query StyleVision Commands` for a description of how to integrate StyleVision commands into your application. Send commands to StyleVision via the method `Exec`. Query if a command is currently enabled or disabled using the method `QueryStatus`.
Use StyleVisionControlDocument

An instance of the StyleVisionControlDocument ActiveX control allows you to embed one StyleVision document editing window into your application. You can use any number of instances you need.

Use the method Open to load any other existing file.

The control does not support a read-only mode. The value of the property ReadOnly is ignored.

Use Path and Save or methods and properties accessible via the property Document to access document functionality.
Use StyleVisionControlPlaceHolder

Instances of StyleVisionControlPlaceHolder ActiveX controls allow you to selectively embed the additional helper windows of StyleVision into your application. The property PlaceholderWindowID selects the StyleVision helper window to be embedded. Use only one StyleVisionControlPlaceHolder for each window identifier. See for valid window identifiers.

For placeholder controls that select the StyleVision project window, additional methods are available. Use OpenProject to load a StyleVision project. Use the property Project and the methods and properties from the StyleVision automation interface to perform any other project related operations.
Query StyleVision Commands

When integrating at document-level, no menu or toolbar from StyleVision is available to your application. Instead, you can query all the commands and the structure of the application menu at runtime. Professional applications will need to integrate this menu in a sophisticated manner into their own menu structure. Your installation of StyleVision even provides you with command label images used within StyleVision. See the folder StyleVisionExamples\ActiveX\Images of your StyleVision installation for icons in GIF format. The file names correspond to the labels of commands.
Programming Languages

This section contains examples of StyleVision 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 StyleVision installation.
C#

The C# example shows how to integrate the StyleVisionControl in a common desktop application created with C# using Visual Studio 2008.

Source code for all examples is available in the folder `<ApplicationFolder>\Examples\ActiveX\C#` of your StyleVision installation. Please note that the example application is already complete. There is no need to change anything if you want to run it and see it working.

Introduction

Adding the StyleVision components to the Toolbox

Before you take a look at the sample project please add the assemblies to the .NET IDE Toolbox. The StyleVision Installer will have already installed the assemblies in the .NET Global Assembly Cache (GAC). If you open the Toolbox dialog under Tools | Add/Remove Toolbox Items the controls will appear as AxStyleVisionControl, AxStyleVisionControlDocument and AxStyleVisionControlPlaceholder on the .NET Framework Components tab. Check all to make them available to the IDE.

Now, to load the project, you can open the StyleVision.sln file in the StyleVisionExamples\ActiveX\C#\StyleVision folder of your application folder.

Placing the StyleVisionControl

It is necessary to have one StyleVisionControl instance to set the integration level and to manage the Document and Placeholder controls of the StyleVision library. The control is accessible via the General section of the Toolbox helper window in the IDE. To add it you need to select the component in the Toolbox window and drag a rectangle wherever you want to have it in the destination window. If you have an application which does not open a window on startup you can use a simple invisible Form with the control on it which is created manually in the code.

The example project adds this instance to the main MdiContainer MDIMain. If you open MDIMain in the Design View from the Solution Explorer you will see a light blue rectangle at the top-left side in the client area of the Frame window. Selecting this rectangle will show you the properties of the StyleVisionControl. It is important to set the IntegrationLevel property to ICAActiveXIntegrationOnDocumentLevel in order to turn on the Document and Placeholder support of the StyleVision library.
The code listings in this section show how to integrate the StyleVisionControl at **application level** and **document level**. Source code for all examples is available in the folder `<ApplicationFolder>\Examples\ActiveX\HTML` of your StyleVision installation.

**Integration at Application Level**

This example shows a simple integration of the StyleVision control at application-level into a HTML page. The integration is described in the following sections:

- Instantiate a StyleVisionControl 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 StyleVision installation: `StyleVisionExamples\ActiveX\HTML\StyleVisionActiveX_ApplicationLevel.htm`.

**Note:** This example works only in Internet Explorer.

**Instantiate the Control**

The HTML **Object** tag is used to create an instance of the StyleVisionControl. The **Classid** is that of StyleVisionControl. Width and height specify the window size. No additional parameters are necessary, since application-level is the default.

```html
<OBJECT id="objStyleVisionControl"
Classid="clsid:559db547-71fc-435d-aa8e-7faa321d0c6e"
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:

```html
<input type="button" value="Open Marketing Expenses" onclick="BtnOpenMEFile()">
```

When clicked, a predefined document will be opened in the StyleVisionControl. We use a method to locate the file relative to the StyleVisionControl so the example can run on different installations.

```javascript
// ---------------------------------
// open a pre-defined document
function BtnOpen() {
  if(strPath.value.length > 0) {
    var absolutePath = MakeAbsolutePath(strPath.value);
    var objDoc = objStyleVisionControl.Open(absolutePath);
    if (objDoc == null)
      alert("Unable to locate " + absolutePath + " at: " + objStyleVisionControl.BaseHref);
  }
  else
```
{  
    alert("Please set path for the document first!");
    strPath.focus();
}
</SCRIPT>

Check Validity of Current Document

The validity of the current document is checked using the following script:

```javascript
// check validity of current document.
// if validation fails, show validation result in alert box.
function BtnTest() {
    // get top-level object of automation interface
    var objApp = objStyleVisionControl.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 StyleVisionControl custom events to show the principle:
Integration at Document Level

This example shows an integration of the StyleVision control at document-level into a HTML page. The following topics are covered:

- Instantiate a StyleVisionControl ActiveX control object in HTML code
- Instantiate a StyleVisionControlDocument ActiveX control to allow editing a StyleVision file
- Instantiate one StyleVisionControlPlaceHolder for a StyleVisionControl project window
- Instantiate one StyleVisionControlPlaceHolder to alternatively host one of the StyleVision helper windows
- Create a simple customer toolbar for some heavy-used StyleVision commands
- Add some more buttons that use the COM automation interface of StyleVision
- Use event handlers to update command buttons

This example is available in its entirety in the file StyleVisionActiveX_ApplicationLevel.htm within the C:\Program Files\Altova\StyleVision2015\Examples\ActiveX\HTML\ folder of your StyleVision installation.

Note: This example works only in Internet Explorer.

Instantiate the StyleVisionControl

The HTML OBJECT tag is used to create an instance of the StyleVisionControl. The Classid is that of StyleVisionControl. Width and height are set to 0 since we use this control as manager control without use for its user interface. The integration level is specified as a parameter within the OBJECT tag.

```html
<Object id="" Classid="clsid:559db547-71fc-435d-aa8e-7faa321d0c6e"
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 mapping.

```
<OBJECT id="objDoc1"
   Classid="clsid:8EE4E77E-C793-468B-A0D8-5D6334CD6986"
   width="600"
   height="500"
   VIEWASTEXT>
   <PARAM NAME="NewDocument">
</OBJECT>
```

Create Project Window

The HTML `<OBJECT>` tag is used to create a `StyleVisionControlPlaceHolder` window. The first additional custom parameter defines the placeholder to show the `StyleVision` project window. The second parameter loads one of the example projects delivered with your `StyleVision` installation (located in the `<yourusername>/MyDocuments` folder).

```
<OBJECT id="objProjectWindow"
   Classid="clsid:E6ECBA9C-0E01-4FD5-98C3-C08B3D4824BC"
   width="200"
   height="200"
   VIEWASTEXT>
   <PARAM name="PlaceholderWindowID" value="-1">
   <PARAM name="FileName" value="StyleVisionExamples/Examples.svp">
</OBJECT>
```

Create Placeholder for Helper Windows

The HTML `<OBJECT>` tag is used to instantiate a `StyleVisionControlPlaceHolder` ActiveX control that can host the different `StyleVision` helper windows. Initially, no helper window is shown. See the example file.

```
<OBJECT id="objEHWindow"
   Classid="clsid:E6ECBA9C-0E01-4FD5-98C3-C08B3D4824BC"
   width="200"
   height="200"
   VIEWASTEXT>
   <PARAM name="PlaceholderWindowID" value="-1">
</OBJECT>
```

Three buttons allow us to switch the actual window that will be shown. The JavaScript execute on-button-click sets the property `PlaceholderWindowID` to the corresponding value defined in

```
<input type="button" value="New File" onclick="BtnNewFile(objDoc1)">
<input type="button" value="Open file" onclick="BtnOpenFile(objDoc1)">
<input type="button" value="Save File" onclick="BtnSaveFile(objDoc1)">
<input type="button" value="Open project" onclick="BtnOpenProject(objDoc1)">
<input type="button" value="Save project" onclick="BtnSaveProject()">
<input type="button" value="Close project" onclick="BtnCloseProject()">
```
Mandatory Event Handlers

A mandatory event handler, such as for an already opened file:

<SCRIPT LANGUAGE="javascript">
   function objStyleVisionControl::OnOpenedOrFocused( strFilePath,
      bOpenWithThisControl, bFileAlreadyOpened)
   {
      alert("Opening...");
      if (!bFileAlreadyOpened)
         DoOpenFile(objDoc1, strFilePath);
   }
</SCRIPT>
Java

StyleVision ActiveX components can be accessed from Java code. To allow this, the libraries listed below must reside in the classpath. These libraries are partly delivered with the StyleVision Integration Package and are placed in the folder JavaAPI in the StyleVision application folder.

- AltovaAutomation.dll: a JNI wrapper for Altova automation servers (in case of the 32-bit installation of StyleVision)
- AltovaAutomation_x64.dll: a JNI wrapper for Altova automation servers (in case of the 64-bit installation of StyleVision)
- AltovaAutomation.jar: Java classes to access Altova automation servers
- StyleVisionActiveX.jar: Java classes that wrap the StyleVision ActiveX interface
- StyleVisionActiveX_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 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 ActiveX Control names to Java

The rules for mapping between the ActiveX controls and the Java wrapper are as follows:

- **Classes and class names**
  For every component of the StyleVision 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 StyleVisionControl, 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: StyleVisionControl: Java class to access the application StyleVisionControlEvents: Events interface for the StyleVisionControl
StyleVisionControlEventsDefaultHandler: Default handler for StyleVisionControlEvents

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>StyleVisionControlDocument, New</td>
<td>newDocument</td>
</tr>
<tr>
<td>AuthenticView, method Goto</td>
<td>gotoElement</td>
</tr>
<tr>
<td>AuthenticRange, method Goto</td>
<td>gotoElement</td>
</tr>
<tr>
<td>AuthenticRange, method Clone</td>
<td>cloneRange</td>
</tr>
</tbody>
</table>

The StyleVisionActiveX.jar library
The classes and interfaces contained in the jar file are part of the com.altova.automation.StyleVision package. They are described in the StyleVisionActiveX_JavaDoc.zip file from the folder Examples\ActiveX\Java in the StyleVision application folder.

This section
This section shows how some basic StyleVision 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 a StyleVision Node Tree

Example Java Project
The StyleVision 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 StyleVisionControl 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:

AltovaAutomation.dll  
Java-COM bridge: DLL part (for the 32-bit installation)

AltovaAutomation_x64.dll  
Java-COM bridge: DLL part (for the 64-bit installation)

AltovaAutomation.jar  
Java-COM bridge: Java library part

StyleVisionActiveX.jar  
Java classes of the StyleVision ActiveX control

StyleVisionContainer.java  
Java example source code

StyleVisionContainerEventHandler.java  
Java example source code

StyleVisionContainerDialog.java  
Java example source code

BuildAndRun.bat  
Batch file to compile and run example code from the command line prompt. Expects folder where Java Virtual Machine resides as parameter.

.classpath  
Eclipse project helper file

.project  
Eclipse project file

StyleVisionActiveX_Javadoc.zip  
Javadoc file containing help documentation for the Java API

**What the example does**

The example places one StyleVision document editor window, the StyleVision project window, the StyleVision schema tree window and the StyleVision design overview window in an AWT frame window. It reads out the main menu defined for StyleVision 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 StyleVision, which is registered as an automation server, or activates StyleVision if it is already running.
- **Loading Data in the Controls**: Locates one of the example documents installed with StyleVision 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 StyleVision events.
- **Creating a StyleVision Node Tree**: Shows how to create a StyleVision node tree and prepare it for modal activation.

**Running the example from the command line**

Open a command prompt window and type:

```
buildAndRun.bat "<Path-to-the-Java-bin-folder>"
```
The Java binary folder must be that of a Java Development Kit (JDK) 7 or later installation on your computer.

Press Return. The Java source in StyleVisionContainer.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 located in the same folder as this Readme file to your workspace. Since you may not have write-access in this folder it is recommended to tell Eclipse to copy the project files into its workspace. The project **StyleVisionContainer** will then appear in your Package Explorer or Navigator.

If you want to use the 64-bit version of the StyleVision ActiveX control you need to use a 64-bit version of Eclipse.

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 **StyleVisionContainer.java** is listed below with comments.

```java
001 // access general JAVA-COM bridge classes
002 import com.altova.automation.libs.*;
003
004 //access StyleVision Java-COM bridge
005 import com.altova.automation.StyleVision.*;
007 import com.altova.automation.StyleVision.Enums.ICActiveXIntegrationLevel;
008
009 // access AWT components
010 import java.awt.*;
011 import java.awt.event.*;
012
013 import javax.swing.*;
014
015 016 /**<
017     * A simple example of a container for StyleVision document-level integration using Java AWT/Swing.
018     * The application's GUI shows a single document editing window and 3 different tool windows:
019     * The project window, the design overview window and the schema tree window.
020     */
```
021 * The application's menu gets created by reading it out from the
022 * StyleVision control.
023 * Communication between the project window and the document editing window
024 * gets established
025 * by the event handler for the onOpenedOrFocused event. See
026 * StyleVisionControlEventsDefaultHandler
027 * for further events.
028 *
029 * @author Altova GmbH
030 */
031 public class StyleVisionContainer
032 {
033  /**
034   * StyleVision manager control - always needed
035   */
036  public static StyleVisionControl styleVisionControl = null;
037  
038  /**
039   * StyleVisionDocument editing control
040   */
041  public static StyleVisionControlDocument styleVisionDocument = null;
042  
043  /**
044   * Tool windows - StyleVision place-holder controls
045   */
046  private static StyleVisionControlPlaceHolder styleVisionProjectToolWindow = null;
047  private static StyleVisionControlPlaceHolder styleVisionDesignToolWindow = null;
048  private static StyleVisionControlPlaceHolder styleVisionSchemaTreeToolWindow = null;
049  
050  /**
051   * The hosting frame
052   */
053  private static Frame frame;
054  
055  /**
056   * Helper function that initializes the Document control
057   */
058  public static void initStyleVisionDocument()
059  {
060      try
061      {
062          if ( styleVisionDocument != null )
063              styleVisionDocument.requestFocusInWindow();
064          else
065              styleVisionDocument = new StyleVisionControlDocument();
066          frame.add( styleVisionDocument, BorderLayout.CENTER );
067          frame.validate();
068          // move the focus to the document control - used when querying for the
069          // command status while enabling/disabling menu items
070      }
071      catch ( Exception ex )
072      {
072     
073     ex.printStackTrace();  
074     
075     }
076     
077    /**
078    * Do something else
079    */
080  
081  private static void doSmthElse( Document doc )
082  {
083      StyleVisionDialog dlg = new StyleVisionDialog( doc, frame );
084      dlg.pack();
085      dlg.setBounds( 0, 0, 500, 300 );
086      dlg.setLocationRelativeTo(frame);
087      dlg.setVisible( true );
088  }
089  
090    /**
091    * The main entry point.
092    * Creates the application's frame and the StyleVision windows.
093    * StyleVisionControl, although not visible, is important for the
094    * coordination between the different ActiveX controls of StyleVision.
095    */
096  
097  public static void main( String[] args )
098  {
099    try
100    {
101      frame = new Frame( "Java ActiveX host window" );
102      frame.setLayout( new BorderLayout() );
103      
104      // Create the main frame of the application
105      Dimension btnDim = new Dimension( 130, 25 );
106      JPanel westPanel = new JPanel();
107      westPanel.setPreferredSize( new Dimension( 140, 400 ) );
108      westPanel.setMaximumSize( new Dimension( 140, 800 ) );
109      westPanel.add( new Label( "Open documents") );
110      JButton btnOpenPxf = new JButton( "Open OrgChart" );
111      westPanel.add( btnOpenPxf );
112      JButton btnOpenLang = new JButton( "Open MultiLang" );
113      westPanel.add( btnOpenLang );
114      westPanel.add( new Label( "Create/destroy") );
115      JButton btnProject = new JButton( "Project window" );
116      westPanel.add( btnProject );
117      JButton btnDesign = new JButton( "Design overview" );
118      westPanel.add( btnDesign );
119      JButton btnSchtree = new JButton( "Schema tree" );
120      westPanel.add( btnSchtree );
121      westPanel.add( new Label( "Create menu" ) );
122      JButton btnMenu = new JButton( "Load file menu" );
123      westPanel.add( btnMenu );
124      JButton btnTree = new JButton( "Show properties" );
125      westPanel.add( btnTree );
126      // Create the StyleVision ActiveX controls. First should be
127      StyleVisionControl determining that we want to place document controls and
// controls individually. It gives us full control over the menu, as well.
styleVisionControl = new StyleVisionControl
( IActiveXIntegrationLevel.IActiveXIntegrationOnDocumentLevel.getValue() );

// Panel that will hold the Project/Xpath/Attributes windows
final JPanel southPanel = new JPanel();
frame.add( southPanel, BorderLayout.SOUTH );
frame.add( styleVisionControl, BorderLayout.NORTH );
frame.add( westPanel, BorderLayout.WEST );
initStyleVisionDocument();

// Listen in this class to communication events ( e.g. onOpenedOrFocused is handled )
final StyleVisionContainerEventHandler handlerObject = new StyleVisionContainerEventHandler();
styleVisionControl.addListener( handlerObject );

// Prepare a shutdown mechanism
frame.addWindowListener( new WindowAdapter()
{
public void windowClosing( WindowEvent ev )
{
frame.dispose();
System.exit(0); }
} );
frame.setVisible( true );

// Locate samples installed with the product.
final String strExamplesFolder = System.getenv( "USERPROFILE" ) + "\My Documents\Altova\StyleVision2013\StyleVisionExamples\";

// Create a project window and open the sample project in it
styleVisionProjectToolWindow = new StyleVisionControlPlaceHolder
( StyleVisionControlPlaceHolderWindow.StyleVisionControlProjectWnd.getValue() );
styleVisionProjectToolWindow.setPreferredSize( new Dimension( 200, 200 ) );
// For the beginning hide the project window
styleVisionProjectToolWindow.setVisible( false );
frame.add( styleVisionProjectToolWindow, BorderLayout.NORTH );
styleVisionProjectToolWindow.openProject( strExamplesFolder + "Examples.svp" );

// Open the PXF file when button is pressed
btnOpenPxf.addActionListener( new ActionListener() { public void actionPerformed(ActionEvent e) { try
{ styleVisionControl.open( strExamplesFolder + "OrgChart.pxf" ); } catch (AutomationException e1) { e1.printStackTrace(); } } });

// Open the XML file when button is pressed
btnOpenLang.addActionListener( new ActionListener() { public void actionPerformed(ActionEvent e) { try
{ } } });
173     styleVisionControl.open(strExamplesFolder + "MultiLangBy2ndFile.sps");
174     } catch (AutomationException e1) {
175         e1.printStackTrace();
176     }
177 }
178 }
179 }
180 // Show/hide the project window when button is pressed
181 btnProject.addActionListener(new ActionListener() {
182     public void actionPerformed(ActionEvent e) {
183         if (!styleVisionProjectToolWindow.isVisible()) {
184             // remove the hidden window from the frame (which acted like a temporary parent)
185             frame.remove(styleVisionProjectToolWindow);
186             southPanel.add(styleVisionProjectToolWindow);
187             styleVisionProjectToolWindow.setVisible(true);
188         } else {
189             styleVisionProjectToolWindow.setVisible(false);
190             southPanel.remove(styleVisionProjectToolWindow);
191             // Add the hidden window to the frame (temporary parent)
192             frame.add(styleVisionProjectToolWindow, BorderLayout.NORTH);
193         }
194         frame.validate();
195     } } );
196 }
197 }
198 // Create/destroy the Design overview window when button is pressed
199 btnDesign.addActionListener(new ActionListener() {
200     public void actionPerformed(ActionEvent e) {
201         if (styleVisionDesignToolWindow == null) {
202             try {
203                 // Create a new window and add it to the south panel
204                 styleVisionDesignToolWindow = new StyleVisionControlPlaceHolder(StyleVisionControlPlaceholderWindow.StyleVisionControlDesignOverviewWnd.getValue());
205                 styleVisionDesignToolWindow.setPreferredSize(new Dimension(200, 200));
206                 southPanel.add(styleVisionDesignToolWindow);
207             } catch (AutomationException e1) {
208                 e1.printStackTrace();
209             }
210         } else {
211             // Remove the window and the reference
212             southPanel.remove(styleVisionDesignToolWindow);
213             styleVisionDesignToolWindow = null;
214         }
215         frame.validate();
216     } } );
217 }
218 // Create/destroy the Schema tree window when button is pressed
219 btnSchtree.addActionListener(new ActionListener() {
220     public void actionPerformed(ActionEvent e) {
221         if (styleVisionSchemaTreeToolWindow == null) {
222             try {
223                 // Create a new window and add it to the south panel
224                 styleVisionSchemaTreeToolWindow = new StyleVisionControlPlaceHolder(StyleVisionControlPlaceholderWindow.StyleVisionControlSchemaSourcesWnd.getValue());
225                 southPanel.add(styleVisionSchemaTreeToolWindow);
226             } catch (AutomationException e1) {
227                 e1.printStackTrace();
228             }
229         } else {
230             // Remove the window and the reference
231             southPanel.remove(styleVisionSchemaTreeToolWindow);
232             styleVisionSchemaTreeToolWindow = null;
233         }
234         frame.validate();
235     } } );
226   styleVisionSchemaTreeToolWindow.setPreferredSize(new Dimension(200, 200));
227   southPanel.add(styleVisionSchemaTreeToolWindow);
228 } catch(AutomationException e1) {
229   e1.printStackTrace();
230 }
231 } else {
232   southPanel.remove(styleVisionSchemaTreeToolWindow);
233   styleVisionSchemaTreeToolWindow = null;
234 }
235   frame.validate();
236 }
237 }
238
239 // Load the file menu when the button is pressed
240 btnMenu.addActionListener(new ActionListener() {
241   public void actionPerformed(ActionEvent e) {
242     try {
243       // Create the menubar that will be attached to the frame
244       MenuBar mb = new MenuBar();
245       // Load the main menu's first item - the File menu
246       StyleVisionCommand xmlSpyMenu = 
247         styleVisionControl.getMainMenu().getSubCommands().getItem(0);
248       // Create Java menu items from the Commands objects
249       Menu fileMenu = new Menu();
250       handlerObject.fillMenu(fileMenu, xmlSpyMenu.getSubCommands());
251       fileMenu.setLabel(xmlSpyMenu.getLabel().replace("&", "") );
252       mb.add(fileMenu);
253       frame.setMenuBar(mb);
254       frame.validate();
255   } catch(AutomationException e1) {
256     e1.printStackTrace();
257   }
258   }
259 }
260 }
261
262 // Do something else when the button is pushed
263 btnTree.addActionListener(new ActionListener() {
264   public void actionPerformed(ActionEvent e) {
265     try {
266       doSmthElse(styleVisionDocument.getDocument());
267     } catch(AutomationException e1) {
268       e1.printStackTrace();
269     }
270   }
271 }
272
273 // Show the main window
274 frame.setBounds(0, 0, 800, 600);
275 frame.validate();
276
277 // feel free to extend.
278 }
279 catch(AutomationException e) {
280   // Show what went wrong
281   e.printStackTrace();
282 }
283
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
/**
 * StyleVision manager control - always needed
 */
public static StyleVisionControl styleVisionControl = null;

/**
 * StyleVisionDocument editing control
 */
public static StyleVisionControlDocument styleVisionDocument = null;

/**
 * Tool windows - StyleVision place-holder controls
 */
private static StyleVisionControlPlaceHolder styleVisionProjectToolWindow = null;
private static StyleVisionControlPlaceHolder styleVisionDesignToolWindow = null;
private static StyleVisionControlPlaceHolder styleVisionSchemaTreeToolWindow = null;

// Create the StyleVision ActiveX controls. First should be StyleVisionControl
// determining that we want to place document controls and place-holder controls individually. It gives us full control over the menu, as well.
styleVisionControl = new StyleVisionControl(ICAActiveXIntegrationLevel.ICActiveXIntegrationOnDocumentLevel.getValue());

styleVisionDocument = new StyleVisionControlDocument();
frame.add( styleVisionDocument, BorderLayout.CENTER );

// Create a project window and open the sample project in it
styleVisionProjectToolWindow = new StyleVisionPlaceHolder(StyleVisionControlPlaceholderWindow.StyleVisionControlProjectWindowToolWindow.valueOf());
styleVisionProjectToolWindow.setPreferredSize( new Dimension( 200, 200 ) );
```

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" ) +
```
Basic Event Handling

The code listing below shows how basic events can be handled. When calling the StyleVisionControl’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 StyleVisionDocumentControl’s `open` method.

```java
// Open the PXF file when button is pressed
btnOpenPxf.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            styleVisionControl.open(strExamplesFolder + "OrgChart.pxf");
        } catch (AutomationException e1) {
            e1.printStackTrace();
        }
    }
});

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 {
            StyleVisionContainer.initStyleVisionDocument();
            StyleVisionContainer.styleVisionDocument.open(i_strFileName);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

Menus

The code listing below shows how menu items can be created. Each `StyleVisionCommand` 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 `StyleVisionControl` 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
tbtnMenu.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
StyleVisionCommand xmlSpyMenu = styleVisionControl.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();
}
} ) ;
/**
 * Populates a menu with the commands and submenus contained in a
 * StyleVisionCommands object
 */
public void fillMenu(Menu newMenu, StyleVisionCommands styleVisionMenu) throws AutomationException {
    // For each command/submenu in the xmlSpyMenu
    for ( int i = 0 ; i < styleVisionMenu.getCount() ; ++i )
    {
        StyleVisionCommand styleVisionCommand = styleVisionMenu.getItem( i );
        if ( styleVisionCommand.getIsSeparator() )
        {
            newMenu.addSeparator();
        }
        else
        {
            StyleVisionCommands subCommands = styleVisionCommand.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
                MenuItem mi = new MenuItem( styleVisionCommand.getLabel().replace( "&", "") );
                mi.setActionCommand( "" + styleVisionCommand.getID() );
                mi.addActionListener( this );
                newMenu.add(mi);
                menuMap.put( styleVisionCommand.getID(), mi );
            }
            else
            {
                // Submenu -> create submenu and repeat recursively
                Menu newSubMenu = new Menu();
                fillMenu( newSubMenu, subCommands );
                newSubMenu.setLabel( styleVisionCommand.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
                StyleVisionContainer.initStyleVisionDocument();
                break;
            default:
                StyleVisionContainer.styleVisionControl.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 StyleVisionControl.
* Called to enable/disable commands
* */
@Override
public void onUpdateCmdUI() throws AutomationException
{
    // A command should be enabled if the result of queryStatus contains the
    // Supported (1) and Enabled (2) flags
    for (java.util.Map.Entry<Integer, MenuItem> pair : menuMap.entrySet())
    {
        pair.getValue().setEnabled(StyleVisionContainer.styleVisionControl.queryStatus(pair.getKey()) > 2);
    }
}

/**
* Call-back from the StyleVisionControl.
* Usually called while enabling/disabling commands due to UI updates
* */
@Override
public boolean isActiveEditor(String i_strFilePath) throws AutomationException
{
    try
    {
        return StyleVisionContainer.styleVisionDocument.getDocument().getFullName().equalsIgnoreCase(i_strFilePath);
    }
}
Listing the Properties of a StyleVision Document

The listing below shows the properties of a StyleVision document (schemas, parameters, etc) can be loaded as nodes in a tree.

```java
//access StyleVision Java-COM bridge
import com.altova.automation.StyleVision.*;

// access AWT/Swing components
import java.awt.*;
import javax.swing.*;
import javax.swing.tree.*;

/**
 * A simple example of a tree control loading the structure from a StyleVision Document object.
 * The class receives a Document object, loads its nodes in a JTree, and prepares
 * for modal activation.
 * Feel free to modify and extend this sample.
 * @author Altova GmbH
 */
class StyleVisionDialog 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 StyleVisionDialog( Document doc, Frame parent ) {
        // Construct the modal dialog
        super( parent, "Data tree", true );
        // Arrange controls in the dialog
        top = new DefaultMutableTreeNode("root");
        myTree = new JTree(top);
        setContentPane( new JScrollPane( myTree ) );
        // Build up the tree; expand all nodes, hide root node
        fillTree( top, doc );
        //... (more code)
    }

    //... (catch block)
    return false;
}
```
045   for(int i = 0 ; i < myTree.getRowCount() ; ++i )
046       myTree.expandRow( i );
047   myTree.setRootVisible( false );
048 }
049
050 /**
051 * Loads the nodes of an XML element under a given tree node
052 * @param node Target tree node
053 * @param doc Source XML element
054 */
055 private void fillTree( DefaultMutableTreeNode node, Document doc)
056 {
057       try
058 {
059           DefaultMutableTreeNode titleNode = new  
060 DefaultMutableTreeNode( "SchemaSources" ) ;
061           for (SchemaSource schema : doc.getSchemaSources())
062 {
063               String nodeText = "$" + schema.getName() ;
064               if ( schema.getIsMainSchemaSource() )
065                   nodeText += " (main)";
066               DefaultMutableTreeNode newNode = new DefaultMutableTreeNode(nodeText ) ;
067               node.add( titleNode ) ;
068               titleNode.add( newNode ) ;
069               String attribute = schema.getSchemaFileName() ;
070               if ( attribute.length() > 0 )
071               {
072                   DefaultMutableTreeNode subNode = new  
073 DefaultMutableTreeNode( "SchemaFile" ) ;
074                   subNode.add( new DefaultMutableTreeNode( attribute ) ) ;
075                   newNode.add( subNode ) ;
076               }
077               attribute = schema.getWorkingXMLFileName() ;
078               if ( attribute.length() > 0 )
079               {
080                   DefaultMutableTreeNode subNode = new  
081 DefaultMutableTreeNode( "Working XML" ) ;
082                   subNode.add( new DefaultMutableTreeNode( attribute ) ) ;
083                   newNode.add( subNode ) ;
084               }
085               attribute = schema.getTemplateFileName() ;
086               if ( attribute.length() > 0 )
087               {
088                   DefaultMutableTreeNode subNode = new  
089 DefaultMutableTreeNode( "Template" ) ;
090                   subNode.add( new DefaultMutableTreeNode( attribute ) ) ;
091                   newNode.add( subNode ) ;
092               }
093               titleNode = new DefaultMutableTreeNode( "Parameters" ) ;
094               for (Parameter param : doc.getParameters())
095               {
096                   DefaultMutableTreeNode newNode = new DefaultMutableTreeNode( "$" +
097                   param.getName() ) ;
098                   node.add( titleNode ) ;
099                   titleNode.add( newNode ) ;
100                   DefaultMutableTreeNode subNode = new  
101 DefaultMutableTreeNode( "Value" ) ;
102       }}
subNode.add(new DefaultMutableTreeNode(param.getValue()));
newNode.add(subNode);
}

catch (Exception e)
{
e.printStackTrace();
}
}
Command Table for StyleVision

Tables in this section list the names and identifiers of all commands that are available within StyleVision. Every sub-section lists the commands from the corresponding top-level menu of StyleVision. The left-most column shows the command's menu text to make it easier for you to identify the functionality behind the command. The last sub-section is a collection of those commands that are not accessible via the main menu.

Depending on the edition of StyleVision you have installed, some of these commands might not be supported. See `Query StyleVision Commands` on how to query the current resource structure and command availability.

Use the command identifiers with `StyleVisionControl.QueryStatus` or `StyleVisionControlDocument.QueryStatus` to check the current status of a command. Use `StyleVisionControl.Exec` or `StyleVisionControlDocument.Exec` to execute a command.

File Menu
Edit Menu
Project Menu
View Menu
Insert Menu
Table Menu
Authentic Menu
Properties Menu
Tools Menu
Window Menu
Help Menu
Misc Menu
### File Menu

**File** menu commands (some are not available in Pro and Std editions):

<table>
<thead>
<tr>
<th>Command Description</th>
<th>Code</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>New/New from XML Schema/DTD/XML...</td>
<td>IDC_FILE_NEW_FROM_SCHEMA</td>
<td>37579</td>
</tr>
<tr>
<td>New/New from DB...</td>
<td>IDC_FILE_NEW_FROM_DB</td>
<td>37577</td>
</tr>
<tr>
<td>New/New from XML column in DB table...</td>
<td>IDC_FILE_NEW_FROM_DBCELL</td>
<td>37861</td>
</tr>
<tr>
<td>New/New from XBRL Taxonomy...</td>
<td>IDC_FILE_NEW_FROM_XBRL_TAXONOMY</td>
<td>37912</td>
</tr>
<tr>
<td>New/New from HTML file...</td>
<td>IDC_FILE_NEW_FROM_HTML</td>
<td>37578</td>
</tr>
<tr>
<td>New/New (empty)</td>
<td>IDC_FILE_NEW_EMPTY</td>
<td>37576</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>36002</td>
</tr>
<tr>
<td>Close</td>
<td>ID_FILE_CLOSE</td>
<td>57602</td>
</tr>
<tr>
<td>Close All</td>
<td>ID_FILE_CLOSE_ALL</td>
<td>37777</td>
</tr>
<tr>
<td>Save Design</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>37778</td>
</tr>
<tr>
<td>Save Authentic XML Data</td>
<td>ID_SAVE_AUTHENTIC_XML</td>
<td>32860</td>
</tr>
<tr>
<td>Save Authentic XML Data as ...</td>
<td>ID_SAVE_AUTHENTIC_XML_AS</td>
<td>32812</td>
</tr>
<tr>
<td>Save Generated Files/Save Generated XSLT-HTML File...</td>
<td>IDC_SAVE_GEN_XSLT_HTML</td>
<td>37642</td>
</tr>
<tr>
<td>Save Generated Files/Save Generated HTML File...</td>
<td>IDC_SAVE_GEN_HTML</td>
<td>37636</td>
</tr>
<tr>
<td>Save Generated Files/Save Generated XSLT-RTF File...</td>
<td>IDC_SAVE_GEN_XSLT_RTF</td>
<td>37643</td>
</tr>
<tr>
<td>Save Generated Files/Save Generated RTF File...</td>
<td>IDC_SAVE_GEN_RTF</td>
<td>37638</td>
</tr>
<tr>
<td>Save Generated Files/Save Generated XSLT-FO File...</td>
<td>IDC_SAVE_GEN_XSLT_FO</td>
<td>37641</td>
</tr>
<tr>
<td>Save Generated Files/Save Generated FO File...</td>
<td>IDC_SAVE_GEN_FO</td>
<td>37635</td>
</tr>
<tr>
<td>Save Generated Files/Save Generated PDF File...</td>
<td>IDC_SAVE_GEN_PDF</td>
<td>37637</td>
</tr>
<tr>
<td>Save Generated Files/Save Generated XSLT-Word 2007+ File...</td>
<td>IDC_SAVE_GEN_XSLT_WORDML</td>
<td>37523</td>
</tr>
<tr>
<td>Save Generated Files/Save Generated Word 2007+ File...</td>
<td>IDC_SAVE_GEN_WORDML</td>
<td>37529</td>
</tr>
<tr>
<td>Save Generated Files/Save Generated DB Schema...</td>
<td>IDC_SAVE_GEN_DB_SCHEMA</td>
<td>37633</td>
</tr>
<tr>
<td>Save Generated Files/Save Generated DB XML Data...</td>
<td>IDC_SAVE_GEN_DB_XML</td>
<td>37634</td>
</tr>
<tr>
<td>Save Generated Files/Save Generated User-Defined Schema...</td>
<td>IDC_SAVE_GEN_USERDEF_SCHEMA</td>
<td>37639</td>
</tr>
</tbody>
</table>

*contd...*
<table>
<thead>
<tr>
<th>Menu Item</th>
<th>ID</th>
<th>ID Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save Generated Files/Save Generated User-Defined XML Data...</td>
<td>IDC_SAVE_GEN_USERDEF_XML</td>
<td>37640</td>
</tr>
<tr>
<td>Assign Working XML File...</td>
<td>IDC_ASSIGN_WORKING_XML_FILE</td>
<td>37526</td>
</tr>
<tr>
<td>Unassign Working XML File</td>
<td>IDC_UNASSIGN_WORKING_XML_FILE</td>
<td>37683</td>
</tr>
<tr>
<td>Assign Template XML File...</td>
<td>IDC_ASSIGN_TEMPLATE_XML_FILE</td>
<td>37525</td>
</tr>
<tr>
<td>Unassign Template XML File</td>
<td>IDC_UNASSIGN_TEMPLATE_XML_FILE</td>
<td>37682</td>
</tr>
<tr>
<td>Properties...</td>
<td>IDC_FILE_PROPERTIES</td>
<td>36027</td>
</tr>
<tr>
<td>Print Preview...</td>
<td>ID_FILE_PRINT_PREVIEW</td>
<td>57609</td>
</tr>
<tr>
<td>Print...</td>
<td>ID_FILE_PRINT</td>
<td>57607</td>
</tr>
<tr>
<td>Recent File</td>
<td>ID_FILE_MRU_FILE1</td>
<td>57616</td>
</tr>
<tr>
<td>Exit</td>
<td>ID_APP_EXIT</td>
<td>57665</td>
</tr>
</tbody>
</table>
### Edit Menu

**Edit menu commands:**

<table>
<thead>
<tr>
<th>Command</th>
<th>Menu Command</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undo</td>
<td>ID_EDIT_UNDO</td>
<td>57643</td>
</tr>
<tr>
<td>Redo</td>
<td>ID_EDIT_REDO</td>
<td>57644</td>
</tr>
<tr>
<td>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>Find</td>
<td>ID_EDIT_FIND</td>
<td>57636</td>
</tr>
<tr>
<td>Find Next</td>
<td>ID_EDIT_FINDNEXT</td>
<td>36802</td>
</tr>
<tr>
<td>Replace...</td>
<td>ID_EDIT_REPLACE</td>
<td>57641</td>
</tr>
<tr>
<td>Stylesheet Parameters...</td>
<td>IDC_EDIT_PARAMETERS</td>
<td>37573</td>
</tr>
<tr>
<td>Collapse/Expand Markup</td>
<td>IDC_SPSGUI_COLLAPSE_EXPAND_MARKUP</td>
<td>36804</td>
</tr>
<tr>
<td>Select All</td>
<td>ID_EDIT_SELECT_ALL</td>
<td>57642</td>
</tr>
</tbody>
</table>
**Project Menu**

**Project menu commands:**

<table>
<thead>
<tr>
<th>Command</th>
<th>Control ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<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>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>Recent Project</td>
<td>IDC_ICPROJECTGUI_RECENT</td>
<td>37224</td>
</tr>
</tbody>
</table>
View Menu

View menu commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toolbars/Dummy entry</td>
<td>ID_VIEW_TOOLBARS</td>
<td>37807</td>
</tr>
<tr>
<td>Project</td>
<td>ID_VIEW_PROJECT</td>
<td>57682</td>
</tr>
<tr>
<td>Design Overview</td>
<td>ID_VIEW_DESIGNOVERVIEW</td>
<td>37883</td>
</tr>
<tr>
<td>Schema Tree</td>
<td>ID_VIEW_SCHEMA SOURCES</td>
<td>37805</td>
</tr>
<tr>
<td>Design Tree</td>
<td>ID_VIEW_DESIGN TREE</td>
<td>37804</td>
</tr>
<tr>
<td>Style Repository</td>
<td>ID_VIEW_STYLEREPOSITORY</td>
<td>37806</td>
</tr>
<tr>
<td>Context Properties</td>
<td>ID_VIEW_CONTEXTPROPERTY_COMMON</td>
<td>37802</td>
</tr>
<tr>
<td>Context Styles</td>
<td>ID_VIEW_CONTEXTPROPERTY_STYLES</td>
<td>37803</td>
</tr>
<tr>
<td>Design Filter/Show only one template at once</td>
<td>IDC_DESIGN_FILTER_ONE</td>
<td>37704</td>
</tr>
<tr>
<td>Design Filter/Show all template types</td>
<td>IDC_DESIGN_FILTER_ALL</td>
<td>37700</td>
</tr>
<tr>
<td>Design Filter/Show imported templates</td>
<td>IDC_DESIGN_FILTER_IMPORTED</td>
<td>37870</td>
</tr>
<tr>
<td>Design Filter/Show/Hide main template</td>
<td>IDC_DESIGN_FILTER_MAIN</td>
<td>37701</td>
</tr>
<tr>
<td>Design Filter/Show/Hide global templates</td>
<td>IDC_DESIGN_FILTER_MATCH</td>
<td>37702</td>
</tr>
<tr>
<td>Design Filter/Show/Hide design fragments</td>
<td>IDC_DESIGN_FILTER_NAMED</td>
<td>37703</td>
</tr>
<tr>
<td>Design Filter/Show/Hide page layout templates</td>
<td>IDC_DESIGN_FILTER_PAGELAYOUT</td>
<td>37705</td>
</tr>
<tr>
<td>Zoom/Zoom In</td>
<td>IDC_SPSGUI_ZOOM_IN</td>
<td>36805</td>
</tr>
<tr>
<td>Zoom/Zoom Out</td>
<td>IDC_SPSGUI_ZOOM_OUT</td>
<td>36806</td>
</tr>
<tr>
<td>Zoom/Zoom 500%</td>
<td>IDC_SPSGUI_ZOOM_500</td>
<td>36807</td>
</tr>
<tr>
<td>Zoom/Zoom 400%</td>
<td>IDC_SPSGUI_ZOOM_400</td>
<td>36808</td>
</tr>
<tr>
<td>Zoom/Zoom 200%</td>
<td>IDC_SPSGUI_ZOOM_200</td>
<td>36809</td>
</tr>
<tr>
<td>Zoom/Zoom 150%</td>
<td>IDC_SPSGUI_ZOOM_150</td>
<td>36810</td>
</tr>
<tr>
<td>Zoom/Zoom 100%</td>
<td>IDC_SPSGUI_ZOOM_100</td>
<td>36811</td>
</tr>
<tr>
<td>Zoom/Zoom 75%</td>
<td>IDC_SPSGUI_ZOOM_75</td>
<td>36812</td>
</tr>
<tr>
<td>Zoom/Zoom 50%</td>
<td>IDC_SPSGUI_ZOOM_50</td>
<td>36813</td>
</tr>
<tr>
<td>Status Bar</td>
<td>ID_VIEW_STATUS_BAR</td>
<td>59393</td>
</tr>
</tbody>
</table>
**Insert Menu**

Insert menu commands (some are not available in Pro and Std editions):

<table>
<thead>
<tr>
<th>Insert Command</th>
<th>ID</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert Contents</td>
<td>IDC_INSERT_CONTENTS</td>
<td>37846</td>
</tr>
<tr>
<td>Insert Rest of Contents</td>
<td>IDC_INSERT_REST_OF_CONTENTS</td>
<td>37617</td>
</tr>
<tr>
<td>Insert Form Controls/Input Field</td>
<td>IDC_INSERT_EDITFIELD</td>
<td>37854</td>
</tr>
<tr>
<td>Insert Form Controls/Multiline Input Field</td>
<td>IDC_INSERT_MULTILINEEDITFIELD</td>
<td>37855</td>
</tr>
<tr>
<td>Insert Form Controls/Check Box...</td>
<td>IDC_INSERT_CHECKBOX</td>
<td>37857</td>
</tr>
<tr>
<td>Insert Form Controls/Combo Box...</td>
<td>IDC_INSERT_COMBOBOX</td>
<td>37856</td>
</tr>
<tr>
<td>Insert Form Controls/Button...</td>
<td>IDC_INSERT_RADIobutton</td>
<td>37858</td>
</tr>
<tr>
<td>Insert Auto-Calculation/Value...</td>
<td>IDC_INSERT_AUTOCALC_VALUE</td>
<td>37596</td>
</tr>
<tr>
<td>Insert Auto-Calculation/Input Field...</td>
<td>IDC_INSERT_AUTOCALC_FIELD</td>
<td>37594</td>
</tr>
<tr>
<td>Insert Auto-Calculation/Multiline Input Field...</td>
<td>IDC_INSERT_AUTOCALC_MULTILINE_FIELD</td>
<td>37595</td>
</tr>
<tr>
<td>Insert Date Picker</td>
<td>IDC_INSERT_DATEPICKER</td>
<td>37602</td>
</tr>
<tr>
<td>Insert Paragraph</td>
<td>IDC_INSERT_PARAgraph</td>
<td>37847</td>
</tr>
<tr>
<td>Insert Special Paragraph/Address</td>
<td>IDC_INSERT_ADDRESS</td>
<td>37843</td>
</tr>
<tr>
<td>Insert Special Paragraph/Block (div)</td>
<td>IDC_INSERT_BLOCK</td>
<td>37528</td>
</tr>
<tr>
<td>Insert Special Paragraph/Blockquote</td>
<td>IDC_INSERT_BLOCKQUOTE</td>
<td>37527</td>
</tr>
<tr>
<td>Insert Special Paragraph/Center</td>
<td>IDC_INSERT_CENTER</td>
<td>37534</td>
</tr>
<tr>
<td>Insert Special Paragraph/Fieldset</td>
<td>IDC_INSERT_FIELDSET</td>
<td>37574</td>
</tr>
<tr>
<td>Insert Special Paragraph/Preformatted</td>
<td>IDC_INSERT_FORMATTED</td>
<td>37580</td>
</tr>
<tr>
<td>Insert Special Paragraph/Preformatted, wrapping</td>
<td>IDC_INSERT_FORMATTED_WRAP</td>
<td>37876</td>
</tr>
<tr>
<td>Insert Special Paragraph/Heading 1 (h1)</td>
<td>IDC_INSERT_HEADING1</td>
<td>37585</td>
</tr>
<tr>
<td>Insert Special Paragraph/Heading 2 (h2)</td>
<td>IDC_INSERT_HEADING2</td>
<td>37586</td>
</tr>
<tr>
<td>Insert Special Paragraph/Heading 3 (h3)</td>
<td>IDC_INSERT_HEADING3</td>
<td>37587</td>
</tr>
<tr>
<td>Insert Special Paragraph/Heading 4 (h4)</td>
<td>IDC_INSERT_HEADING4</td>
<td>37588</td>
</tr>
<tr>
<td>Insert Special Paragraph/Heading 5 (h5)</td>
<td>IDC_INSERT_HEADING5</td>
<td>37589</td>
</tr>
<tr>
<td>Insert Special Paragraph/Heading 6 (h6)</td>
<td>IDC_INSERT_HEADING6</td>
<td>37590</td>
</tr>
<tr>
<td>Insert Image...</td>
<td>IDC_INSERT_IMAGE</td>
<td>37593</td>
</tr>
<tr>
<td>Insert Horizontal Line</td>
<td>IDC_INSERT_HORIZONTAL_LINE</td>
<td>37606</td>
</tr>
<tr>
<td>Insert Table...</td>
<td>IDC_INSERT_TABLE</td>
<td>40212</td>
</tr>
<tr>
<td>Insert Bullets and Numbering...</td>
<td>IDC_INSERT_FORMAT_BULLETs</td>
<td>37848</td>
</tr>
</tbody>
</table>

contd...
<table>
<thead>
<tr>
<th>Function</th>
<th>ID Code</th>
<th>ID Code Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert Bookmark...</td>
<td>IDC_INSERT_BOOKMARK</td>
<td>37530</td>
</tr>
<tr>
<td>Insert Hyperlink...</td>
<td>IDC_INSERT_HYPERLINK</td>
<td>37849</td>
</tr>
<tr>
<td>Insert Condition...</td>
<td>IDC_INSERT_CONDITION</td>
<td>37850</td>
</tr>
<tr>
<td>Insert Output-based Condition</td>
<td>IDC_INSERT_CONDITION_PER_OUTPUT</td>
<td>37851</td>
</tr>
<tr>
<td>Insert Template</td>
<td>IDC_INSERT_TEMPLATE</td>
<td>40216</td>
</tr>
<tr>
<td>Insert User-Defined Template...</td>
<td>IDC_INSERT_USER_DEFINED_TEMPLATE</td>
<td>40184</td>
</tr>
<tr>
<td>Insert Variable Template...</td>
<td>IDC_INSERT_VARIABLE_TEMPLATE</td>
<td>37531</td>
</tr>
<tr>
<td>Insert Layout Container</td>
<td>IDC_INSERT_LAYOUT_CONTAINER</td>
<td>40213</td>
</tr>
<tr>
<td>Insert Layout Box</td>
<td>IDC_INSERT_LAYOUT_BOX</td>
<td>40214</td>
</tr>
<tr>
<td>Insert Line</td>
<td>IDC_INSERT_SHAPE_LINE</td>
<td>40215</td>
</tr>
<tr>
<td>Insert Table Of Contents/Table of Contents...</td>
<td>IDC_INSERT_TOC_WIZARD</td>
<td>37618</td>
</tr>
<tr>
<td>Insert Table Of Contents/TOC Bookmark</td>
<td>IDC_INSERT_MARKER</td>
<td>37608</td>
</tr>
<tr>
<td>Insert Table Of Contents/TOC Bookmark (Wizard)...</td>
<td>IDC_INSERT_MARKER_WIZARD</td>
<td>37609</td>
</tr>
<tr>
<td>Insert Table Of Contents/TOC Reference</td>
<td>IDC_INSERT_REF</td>
<td>37613</td>
</tr>
<tr>
<td>Insert Table Of Contents/TOC Reference/Entry Text</td>
<td>IDC_INSERT_REF_ENTRY_TEXT</td>
<td>37614</td>
</tr>
<tr>
<td>Insert Table Of Contents/TOC Reference/Leader</td>
<td>IDC_INSERT_REF_LEADER</td>
<td>37615</td>
</tr>
<tr>
<td>Insert Table Of Contents/TOC Reference/Page Reference</td>
<td>IDC_INSERT_REF_PAGE_REFERENCE</td>
<td>37616</td>
</tr>
<tr>
<td>Insert Table Of Contents/Hierarchical numbering</td>
<td>IDC_INSERT_AUTO_NUMBER_LEVEL</td>
<td>37598</td>
</tr>
<tr>
<td>Insert Table Of Contents/Sequential numbering</td>
<td>IDC_INSERT_AUTO_NUMBER_FLAT</td>
<td>37597</td>
</tr>
<tr>
<td>Insert Table Of Contents/TOC Level</td>
<td>IDC_INSERT_LEVEL</td>
<td>37607</td>
</tr>
<tr>
<td>Insert Table Of Contents/TOC Level Reference</td>
<td>IDC_INSERT_REFLEVEL</td>
<td>37860</td>
</tr>
<tr>
<td>Insert Table Of Contents/Template serves as Level</td>
<td>IDC_ASSIGN_LEVEL</td>
<td>37524</td>
</tr>
<tr>
<td>Insert Design Fragment/Dummy entry</td>
<td>IDC_INSERT_NAMED_TEMPLATE</td>
<td>39300</td>
</tr>
<tr>
<td>Insert Page/Column/Document Section/New Page</td>
<td>IDC_INSERT_PAGE_BREAK</td>
<td>37610</td>
</tr>
<tr>
<td>Insert Page/Column/Document Section/Page Number</td>
<td>IDC_INSERT_PAGE_NUMBER</td>
<td>37611</td>
</tr>
<tr>
<td>Insert Page/Column/Document Section/Page Total</td>
<td>IDC_INSERT_PAGE_TOTAL</td>
<td>37612</td>
</tr>
<tr>
<td>Insert Page/Column/Document Section/New Column</td>
<td>IDC_INSERT_COLUMN_BREAK</td>
<td>37864</td>
</tr>
<tr>
<td>Insert Page/Column/Document Section/New Document Section</td>
<td>IDC_INSERT_DOCUMENT_SECTION</td>
<td>37951</td>
</tr>
<tr>
<td>Insert DB Control/Navigation</td>
<td>IDC_INSERT_DBNAVIGATION</td>
<td>37603</td>
</tr>
<tr>
<td>Insert DB Control/Navigation + Goto</td>
<td>IDC_INSERT_DBNAVIGATIONGOTO</td>
<td>37604</td>
</tr>
<tr>
<td>Insert DB Control/Query Button</td>
<td>IDC_INSERT_DBQUERY</td>
<td>37605</td>
</tr>
<tr>
<td>Insert XBRL Element/Context</td>
<td>IDC_INSERT_XBRL_CONTEXT</td>
<td>37942</td>
</tr>
<tr>
<td>Insert XBRL Element/Period</td>
<td>IDC_INSERT_XBRL_PERIOD</td>
<td>37909</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-----------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>Insert XBRL Element/Period Start</td>
<td>IDC_INSERT_XBRL_PERIOD_START</td>
<td>37947</td>
</tr>
<tr>
<td>Insert XBRL Element/Period End</td>
<td>IDC_INSERT_XBRL_PERIOD_END</td>
<td>37948</td>
</tr>
<tr>
<td>Insert XBRL Element/Identifier</td>
<td>IDC_INSERT_XBRL_IDENTIFIER</td>
<td>37910</td>
</tr>
<tr>
<td>Insert XBRL Element/Unit</td>
<td>IDC_INSERT_XBRL_UNIT</td>
<td>37946</td>
</tr>
<tr>
<td>Insert XBRL Element/Footnote</td>
<td>IDC_INSERT_XBRL_FOOTNOTE</td>
<td>37911</td>
</tr>
<tr>
<td>Insert User-Defined Item/User-Defined Element</td>
<td>IDC_STYLEVISIONGUI_INSERT_USERXML_ELEM</td>
<td>37908</td>
</tr>
<tr>
<td>Insert User-Defined Item/User-Defined Block</td>
<td>IDC_STYLEVISIONGUI_INSERT_USERXML_TEXT</td>
<td>37920</td>
</tr>
</tbody>
</table>
### Enclose With Menu

**Enclose With** menu commands (some might not be available in Pro and Std editions):

<table>
<thead>
<tr>
<th>Command</th>
<th>Identifier</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template...</td>
<td>IDC_ENCLOSE_WITHTEMPLATES</td>
<td>40180</td>
</tr>
<tr>
<td>User-Defined Template...</td>
<td>IDC_ENCLOSE_WITHUSERDEFINEDTEMPLATES</td>
<td>40217</td>
</tr>
<tr>
<td>Variable Template...</td>
<td>IDC_ENCLOSE_WITHVARIABLETEMPLATES</td>
<td>37520</td>
</tr>
<tr>
<td>Paragraph</td>
<td>IDC_ENCLOSE_PARAGRAPH</td>
<td>37627</td>
</tr>
<tr>
<td>Special Paragraph/Address</td>
<td>IDC_ENCLOSE_ADDRESS</td>
<td>40194</td>
</tr>
<tr>
<td>Special Paragraph/Block (div)</td>
<td>IDC_ENCLOSE_BLOCK</td>
<td>40195</td>
</tr>
<tr>
<td>Special Paragraph/Blockquote</td>
<td>IDC_ENCLOSE_BLOCKQUOTE</td>
<td>40196</td>
</tr>
<tr>
<td>Special Paragraph/Center</td>
<td>IDC_ENCLOSE_CENTER</td>
<td>40197</td>
</tr>
<tr>
<td>Special Paragraph/Fieldset</td>
<td>IDC_ENCLOSE_FIELDSET</td>
<td>40198</td>
</tr>
<tr>
<td>Special Paragraph/Preformatted</td>
<td>IDC_ENCLOSE_FORMATTED</td>
<td>40199</td>
</tr>
<tr>
<td>Special Paragraph/Preformatted, wrapping</td>
<td>IDC_ENCLOSE_FORMATTED_WRAP</td>
<td>40200</td>
</tr>
<tr>
<td>Special Paragraph/Heading 1 (h1)</td>
<td>IDC_ENCLOSE_HEADING1</td>
<td>40201</td>
</tr>
<tr>
<td>Special Paragraph/Heading 2 (h2)</td>
<td>IDC_ENCLOSE_HEADING2</td>
<td>40202</td>
</tr>
<tr>
<td>Special Paragraph/Heading 3 (h3)</td>
<td>IDC_ENCLOSE_HEADING3</td>
<td>40203</td>
</tr>
<tr>
<td>Special Paragraph/Heading 4 (h4)</td>
<td>IDC_ENCLOSE_HEADING4</td>
<td>40204</td>
</tr>
<tr>
<td>Special Paragraph/Heading 5 (h5)</td>
<td>IDC_ENCLOSE_HEADING5</td>
<td>40205</td>
</tr>
<tr>
<td>Special Paragraph/Heading 6 (h6)</td>
<td>IDC_ENCLOSE_HEADING6</td>
<td>40206</td>
</tr>
<tr>
<td>Bullets and Numbering...</td>
<td>IDC_ENCLOSE_FORMATBULLETS</td>
<td>37581</td>
</tr>
<tr>
<td>Bookmark...</td>
<td>IDC_ENCLOSE_BOOKMARK</td>
<td>40207</td>
</tr>
<tr>
<td>Hyperlink...</td>
<td>IDC_ENCLOSE HYPERLINK</td>
<td>37620</td>
</tr>
<tr>
<td>Condition...</td>
<td>IDC_ENCLOSE_CONDITION</td>
<td>37599</td>
</tr>
<tr>
<td>Output-based Condition</td>
<td>IDC_ENCLOSE_CONDITIONPEROUTPUT</td>
<td>37600</td>
</tr>
<tr>
<td>TOC Bookmark</td>
<td>IDC_ENCLOSE_MARKER</td>
<td>40210</td>
</tr>
<tr>
<td>TOC Bookmark (Wizard)...</td>
<td>IDC_ENCLOSE_MARKERWIZARD</td>
<td>40211</td>
</tr>
<tr>
<td>TOC Level</td>
<td>IDC_ENCLOSE_LEVEL</td>
<td>40208</td>
</tr>
<tr>
<td>TOC Level Reference</td>
<td>IDC_ENCLOSE_REFLEVEL</td>
<td>40209</td>
</tr>
<tr>
<td>User-Defined Element...</td>
<td>IDC_ENCLOSE_WITHUSERDEFINEDXMLELEMENT</td>
<td>40222</td>
</tr>
</tbody>
</table>
### Table Menu

**Table** menu commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Menu Command</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert Table...</td>
<td>IDC_INSERT_TABLE</td>
<td>40212</td>
</tr>
<tr>
<td>Delete Table</td>
<td>IDC_TABLE_DELETE</td>
<td>37658</td>
</tr>
<tr>
<td>Add Table Header Column</td>
<td>IDC_TABLE_ADD_HEADERCOL</td>
<td>37888</td>
</tr>
<tr>
<td>Add Table Footer Column</td>
<td>IDC_TABLE_ADD_FOOTERCOL</td>
<td>37889</td>
</tr>
<tr>
<td>Add Table Header Row</td>
<td>IDC_TABLE_ADD_HEADERROW</td>
<td>37900</td>
</tr>
<tr>
<td>Add Table Footer Row</td>
<td>IDC_TABLE_ADD_FOOTERROW</td>
<td>37901</td>
</tr>
<tr>
<td>Append Row</td>
<td>IDC_TABLE_APPEND_ROW</td>
<td>37657</td>
</tr>
<tr>
<td>Append Column</td>
<td>IDC_TABLE_APPEND_COL</td>
<td>37656</td>
</tr>
<tr>
<td>Insert Row</td>
<td>IDC_TABLE_INSERT_ROW</td>
<td>37664</td>
</tr>
<tr>
<td>Insert Column</td>
<td>IDC_TABLE_INSERT_COL</td>
<td>37662</td>
</tr>
<tr>
<td>Delete Row</td>
<td>IDC_TABLE_DELETE_ROW</td>
<td>37660</td>
</tr>
<tr>
<td>Delete Column</td>
<td>IDC_TABLE_DELETE_COL</td>
<td>37659</td>
</tr>
<tr>
<td>Join Cell Left</td>
<td>IDC_TABLE_JOIN_LEFT</td>
<td>37666</td>
</tr>
<tr>
<td>Join Cell Right</td>
<td>IDC_TABLE_JOIN_RIGHT</td>
<td>37667</td>
</tr>
<tr>
<td>Join Cell Below</td>
<td>IDC_TABLE_JOIN_DOWN</td>
<td>37665</td>
</tr>
<tr>
<td>Join Cell Above</td>
<td>IDC_TABLE_JOIN_UP</td>
<td>37668</td>
</tr>
<tr>
<td>Split Cell Horizontally</td>
<td>IDC_TABLE_SPLIT_HORZ</td>
<td>37670</td>
</tr>
<tr>
<td>Split Cell Vertically</td>
<td>IDC_TABLE_SPLIT_VERT</td>
<td>37671</td>
</tr>
<tr>
<td>View Cell Bounds</td>
<td>IDC_TABLE_SHOW_ZERO_BORDER</td>
<td>37669</td>
</tr>
<tr>
<td>View Table Markup</td>
<td>IDC_TABLE_SHOW_UICELLS</td>
<td>37887</td>
</tr>
<tr>
<td>Table Properties...</td>
<td>IDC_TABLE_EDIT_PROPERTIES</td>
<td>37661</td>
</tr>
</tbody>
</table>
Authentic Menu

Authentic menu commands (some are not available in Pro and Std editions):

<table>
<thead>
<tr>
<th>Command</th>
<th>Value</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text State Icons...</td>
<td>IDC_TEXT_STATE_ICONS</td>
<td>37672</td>
</tr>
<tr>
<td>CALS / HTML Tables...</td>
<td>IDC_CALS_HTML_TABLES</td>
<td>37533</td>
</tr>
<tr>
<td>Auto-add Date Picker</td>
<td>IDC_TOGGLE_DATEPICKER_AUTOINSERT</td>
<td>37674</td>
</tr>
<tr>
<td>Auto-add DB Controls</td>
<td>IDC_TOGGLE_DBCONTROL_AUTOINSERT</td>
<td>37675</td>
</tr>
<tr>
<td>Reload Authentic View</td>
<td>IDC_AUTHENTICGUI_RELOAD</td>
<td>32800</td>
</tr>
<tr>
<td>Validate XML</td>
<td>IDC_VALIDATE</td>
<td>32954</td>
</tr>
<tr>
<td>Select New Row with XML Data for Editing...</td>
<td>IDC_CHANGE_WORKING_DB_XML_CELL</td>
<td>32861</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>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>
<tr>
<td>Move Row Down</td>
<td>IDC_ROW_MOVE_DOWN</td>
<td>32810</td>
</tr>
<tr>
<td>Delete Row</td>
<td>IDC_ROW_DELETE</td>
<td>32807</td>
</tr>
</tbody>
</table>
**Database Menu**

**Database** menu commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Database</td>
<td>ID_VIEW_DBQUERY</td>
<td>37818</td>
</tr>
<tr>
<td>Edit DB Filter...</td>
<td>IDC_EDIT_DBFILTERS</td>
<td>37571</td>
</tr>
<tr>
<td>Clear DB Filter</td>
<td>IDC_CLEAR_DBFILTERS</td>
<td>37547</td>
</tr>
</tbody>
</table>
Properties Menu

**Properties** menu commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>ID</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit Bullets and Numbering...</td>
<td>IDC_EDIT_BULLETS_AND_NUMBERING</td>
<td>37839</td>
</tr>
<tr>
<td>Predefined Input Formatting Strings...</td>
<td>IDC_FORMAT_PREDEFINES</td>
<td>37582</td>
</tr>
</tbody>
</table>
### Tools Menu

**Tools** menu commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Menu Command</th>
<th>ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spelling</td>
<td>IDC_SPSGUI_SPELL_CHECK</td>
<td>36800</td>
<td></td>
</tr>
<tr>
<td>Spelling Options</td>
<td>IDC_SPSGUI SPELL_OPTIONS</td>
<td>36801</td>
<td></td>
</tr>
<tr>
<td>Global Resources</td>
<td>IDC_GLOBALRESOURCES</td>
<td>37401</td>
<td></td>
</tr>
<tr>
<td>Active Configuration/&lt;plugin not loaded&gt;</td>
<td>IDC_GLOBALRESOURCES_SUBMENUENTRY1</td>
<td>37408</td>
<td></td>
</tr>
<tr>
<td>Customize</td>
<td>IDC_CUSTOMIZE_VIEW</td>
<td>37560</td>
<td></td>
</tr>
<tr>
<td>Restore Toolbars and Windows</td>
<td>IDC_APP_RESET_TOOLBARS_AND_WINDOWS</td>
<td>32956</td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td>IDC_TOOLS_OPTIONS</td>
<td>37676</td>
<td></td>
</tr>
</tbody>
</table>
### Window Menu

**Window** menu commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>ID/Description</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascade</td>
<td>ID_WINDOW_CASCADE</td>
<td>57650</td>
</tr>
<tr>
<td>Tile Horizontal</td>
<td>ID_WINDOW TILE_HORIZ</td>
<td>57651</td>
</tr>
<tr>
<td>Tile Vertical</td>
<td>ID_WINDOW TILE VERT</td>
<td>59405</td>
</tr>
<tr>
<td>Arrange Icons</td>
<td>ID_WINDOW ARRANGE</td>
<td>57649</td>
</tr>
</tbody>
</table>
### Help Menu

**Help menu commands:**

<table>
<thead>
<tr>
<th>Table of Contents</th>
<th>ID_HELP_FINDER</th>
<th>57667</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index...</td>
<td>ID_HELP_INDEX</td>
<td>57666</td>
</tr>
<tr>
<td>Search...</td>
<td>ID_HELP_SEARCH</td>
<td>36036</td>
</tr>
<tr>
<td>Software Activation...</td>
<td>IDC_ACTIVATION</td>
<td>36025</td>
</tr>
<tr>
<td>Order form...</td>
<td>ID_HELP_ORDERFORM</td>
<td>36034</td>
</tr>
<tr>
<td>Registration...</td>
<td>ID_HELP_REGISTRATION</td>
<td>36035</td>
</tr>
<tr>
<td>Check for Updates...</td>
<td>IDC_CHECK_FOR_UPDATES</td>
<td>36026</td>
</tr>
<tr>
<td>StyleVision Product Comparison...</td>
<td>IDC_PRODUCT_COMPARISON</td>
<td>32955</td>
</tr>
<tr>
<td>Support Center...</td>
<td>ID_HELP_SUPPORTCENTER</td>
<td>36037</td>
</tr>
<tr>
<td>FAQ on the Web...</td>
<td>ID_HELP_FAQ</td>
<td>36032</td>
</tr>
<tr>
<td>Components Download...</td>
<td>ID_HELP_COMPONENTS</td>
<td>36031</td>
</tr>
<tr>
<td>StyleVision on the Internet...</td>
<td>ID_HELP_INTERNET</td>
<td>36033</td>
</tr>
<tr>
<td>StyleVision Training...</td>
<td>ID_HELP_TRAINING</td>
<td>36038</td>
</tr>
<tr>
<td>About StyleVision...</td>
<td>ID_APP_ABOUT</td>
<td>57664</td>
</tr>
</tbody>
</table>
# Misc Menu

Miscellaneous menu and context menu commands (some are not available in Professional and Basic editions):

<table>
<thead>
<tr>
<th>Command</th>
<th>Active Command</th>
<th>Command ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>IDC_ACTIVATE_PART_TOGGLE</td>
<td>37532</td>
</tr>
<tr>
<td>Database</td>
<td>IDC_ADDRESOURCE_DATABASE</td>
<td>37405</td>
</tr>
<tr>
<td>File</td>
<td>IDC_ADDRESOURCE_FILE</td>
<td>37403</td>
</tr>
<tr>
<td>Folder</td>
<td>IDC_ADDRESOURCE_FOLDER</td>
<td>37404</td>
</tr>
<tr>
<td>Copy Branch</td>
<td>IDC_ADD_COPY_OPTION</td>
<td>37515</td>
</tr>
<tr>
<td>Add New Branch</td>
<td>IDC_ADD_NEW_OPTION</td>
<td>37516</td>
</tr>
<tr>
<td>Append element</td>
<td>IDC_ADD_NEXT_NODE</td>
<td>37517</td>
</tr>
<tr>
<td>Insert element</td>
<td>IDC_ADD_PREV_NODE</td>
<td>37518</td>
</tr>
<tr>
<td>Add Child element</td>
<td>IDC_ADD_SUB_NODE</td>
<td>37519</td>
</tr>
<tr>
<td></td>
<td>IDC_ALIGN_CENTER</td>
<td>37826</td>
</tr>
<tr>
<td></td>
<td>IDC_ALIGN_JUSTIFY</td>
<td>37828</td>
</tr>
<tr>
<td></td>
<td>IDC_ALIGN_LEFT</td>
<td>37825</td>
</tr>
<tr>
<td></td>
<td>IDC_ALIGN_RIGHT</td>
<td>37827</td>
</tr>
<tr>
<td>Assign XML Schema/DTD File...</td>
<td>IDC_ASSIGN_SCHEMA_FILE</td>
<td>37886</td>
</tr>
<tr>
<td>Assign Template XBRL File...</td>
<td>IDC_ASSIGN TEMPLATE XBRL_FILE</td>
<td>37938</td>
</tr>
<tr>
<td>Assign Working XBRL File...</td>
<td>IDC_ASSIGN WORKING XBRL_FILE</td>
<td>37939</td>
</tr>
<tr>
<td>Assign XBRL Taxonomy File...</td>
<td>IDC_ASSIGN XBRL TAXONOMY FILE</td>
<td>37935</td>
</tr>
<tr>
<td>Authentic</td>
<td>IDC_AUTHENTIC</td>
<td>37692</td>
</tr>
<tr>
<td></td>
<td>IDC_BOLD</td>
<td>37822</td>
</tr>
<tr>
<td></td>
<td>IDC_BULLET_ORDERED</td>
<td>37829</td>
</tr>
<tr>
<td></td>
<td>IDC_BULLET_UNORDERED</td>
<td>37830</td>
</tr>
<tr>
<td>Change page margins...</td>
<td>IDC_CHANGE_PAGE_MARGINS</td>
<td>37535</td>
</tr>
<tr>
<td>Dummy entry</td>
<td>IDC_CHANGE_TEMPLATE_MATCH</td>
<td>39320</td>
</tr>
<tr>
<td>Contents</td>
<td>IDC_CHANGE_TO_ALL_CONTENTS</td>
<td>37536</td>
</tr>
<tr>
<td>Button</td>
<td>IDC_CHANGE_TO_BUTTON</td>
<td>37537</td>
</tr>
<tr>
<td>Check Box...</td>
<td>IDC_CHANGE_TO_CHECKBOX</td>
<td>37538</td>
</tr>
<tr>
<td>Input Field</td>
<td>IDC_CHANGE_TO_FIELD</td>
<td>37539</td>
</tr>
<tr>
<td>Image...</td>
<td>IDC_CHANGE_TO_IMAGE</td>
<td>37540</td>
</tr>
<tr>
<td>Multiline Input Field</td>
<td>IDC_CHANGE_TO_MULTILINE_FIELD</td>
<td>37542</td>
</tr>
<tr>
<td>Paragraph</td>
<td>IDC_CHANGE_TO_PARAGRAPH</td>
<td>37543</td>
</tr>
</tbody>
</table>

contd...

<table>
<thead>
<tr>
<th>Command</th>
<th>Active Command</th>
<th>Command ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radio Button...</td>
<td>IDC_CHANGE_TO_RADIO_BUTTON</td>
<td>37544</td>
</tr>
<tr>
<td>Action</td>
<td>Resource ID</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------</td>
<td>------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>Combo Box...</td>
<td>IDC_CHANGE_TO_SELECTION</td>
<td>40220</td>
</tr>
<tr>
<td>Table...</td>
<td>IDC_CHANGE_TO_TABLE</td>
<td>37545</td>
</tr>
<tr>
<td>Clear XPath Filter</td>
<td>IDC_CLEAR_FILTER</td>
<td>37925</td>
</tr>
<tr>
<td>Add to Data Comparison Document</td>
<td>IDC_COMMAND_REFRESH_DATASOURCE</td>
<td>36674</td>
</tr>
<tr>
<td>Add to Schema Comparison Document</td>
<td>IDC_COMMAND_SHOW_IN_EXISTING_SCHEMA_COMPARISON</td>
<td>36690</td>
</tr>
<tr>
<td>Show in new Data Comparison Document</td>
<td>IDC_COMMAND_SHOW_IN_NEW_COMPARISON</td>
<td>36541</td>
</tr>
<tr>
<td>Show in new Schema Comparison Document</td>
<td>IDC_COMMAND_SHOW_IN_NEW_SCHEMA_COMPARISON</td>
<td>36691</td>
</tr>
<tr>
<td>Construct Entry Text using XPath</td>
<td>IDC_CONSTRUCT_ENTRY_TEXT_USING_XPATH</td>
<td>37831</td>
</tr>
<tr>
<td>Convert all to Global Resources</td>
<td>IDC_CONVERT_ALL_DATASOURCES_TO_GLOBAL_RESOURCES</td>
<td>36687</td>
</tr>
<tr>
<td>Convert to Global Resource</td>
<td>IDC_CONVERT_DATASOURCE_TO_GLOBAL_RESOURCES</td>
<td>36684</td>
</tr>
<tr>
<td>Copy Global Template Locally</td>
<td>IDC_CONVERT_FROM_GLOBAL_TEMPLATE</td>
<td>37548</td>
</tr>
<tr>
<td>Convert to attribute</td>
<td>IDC_CONVERT_TO_ATTRIBUTE</td>
<td>37549</td>
</tr>
<tr>
<td>Convert to element</td>
<td>IDC_CONVERT_TO_ELEMENT</td>
<td>37550</td>
</tr>
<tr>
<td>Make Global Template</td>
<td>IDC_CONVERT_TO_GLOBAL_TEMPLATE</td>
<td>37551</td>
</tr>
<tr>
<td>Use Global Template</td>
<td>IDC_CONVERT_TO_GLOBAL_TEMPLATE_WITH_EXISTS</td>
<td>37552</td>
</tr>
<tr>
<td>Copy Global Resource into Project</td>
<td>IDC_COPY_GLOBAL_RESOURCES_TO_PROJECT</td>
<td>36685</td>
</tr>
<tr>
<td>Create Button</td>
<td>IDC_CREATE_BUTTON</td>
<td>37553</td>
</tr>
<tr>
<td>Create Check Box...</td>
<td>IDC_CREATE_CHECKBOX</td>
<td>37554</td>
</tr>
<tr>
<td>Create Image...</td>
<td>IDC_CREATE_DYNAMIC_IMAGE</td>
<td>37555</td>
</tr>
<tr>
<td>Create Combo Box...</td>
<td>IDC_CREATE_DYNAMIC_SELECTION</td>
<td>37556</td>
</tr>
<tr>
<td>Create Input Field</td>
<td>IDC_CREATE_FIELD</td>
<td>37557</td>
</tr>
<tr>
<td>Create Hyperlink</td>
<td>IDC_CREATE_HYPERLINK</td>
<td>37842</td>
</tr>
<tr>
<td>Insert as Hyperlink</td>
<td>IDC_CREATE_HYPERLINK_FROM_FILE</td>
<td>37881</td>
</tr>
<tr>
<td>Insert as Image</td>
<td>IDC_CREATE_IMAGE_FROM_FILE</td>
<td>37882</td>
</tr>
<tr>
<td>Create Multiline Input Field</td>
<td>IDC_CREATE_MULTILINE_FIELD</td>
<td>37558</td>
</tr>
<tr>
<td>Create new Design...</td>
<td>IDC_CREATE_NEWDESIGN_FROM_FILE</td>
<td>37878</td>
</tr>
<tr>
<td>Create Radio Button...</td>
<td>IDC_CREATE_RADIOBUTTON</td>
<td>37559</td>
</tr>
</tbody>
</table>

contd...

<table>
<thead>
<tr>
<th>Action</th>
<th>Resource ID</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autocommit all changes on saving Database XML Document</td>
<td>IDC_DBQUERY_AUTOCOMMIT</td>
<td>38012</td>
</tr>
<tr>
<td>Autohide Database Query window</td>
<td>IDC_DBQUERY_AUTOHIDE</td>
<td>38011</td>
</tr>
<tr>
<td></td>
<td>IDC_DBQUERY_OPTIONS</td>
<td>38005</td>
</tr>
<tr>
<td></td>
<td>IDC_DBQUERY_SQLVIEW_EXECUTE</td>
<td>38006</td>
</tr>
</tbody>
</table>
Open in DatabaseSpy

IDC_DBQUERY_SQLVIEW_EXECUTE_EDIT 38007
IDC_DBQUERY_SQLVIEW_OPEN 38004

IDC_DBQUERY_SQLVIEW_EXECUTE_EDIT 38007
IDC_DBQUERY_SQLVIEW_SAVE 38003
IDC_DBQUERY_SQLVIEW_STOPEXECUTION 38009
IDC_DBQUERY_TOGGLEBROWSER 38001
IDC_DBQUERY_TOGGLERESULT 38000

Open in DatabaseSpy

IDC_DBQUERY_SQLVIEW_OPENINDBSPY 38002
IDC_DBQUERY_SQLVIEW.Save 38003
IDC_DBQUERY_SQLVIEW_STOPEXECUTION 38009
IDC_DBQUERY_TOGGLEBROWSER 38001
IDC_DBQUERY_TOGGLERESULT 38000

Edit Parameters...

IDC_DEFINE_NAMED_TEMPLATE_ACTUAL_PARAMETERS 39715
Define Parameters...

IDC_DEFINE_NAMED_TEMPLATE_FORMAL_PARAMETERS 39716
Define Variables...

IDC_DEFINE_VARIABLES 40185
Delete

IDC_DELETE_NODE 37561
Delete Branch

IDC_DELETE_OPTION 37562
Add Footer

IDC_DESIGNTREE_ADD_FOOTER_ALL 37564
Add Footer Even

IDC_DESIGNTREE_ADD_FOOTER_EVEN 37565
Add Footer First

IDC_DESIGNTREE_ADD_FOOTER_FIRST 40119
Add Footer Last

IDC_DESIGNTREE_ADD_FOOTER_LAST 40123
Add Footer Odd

IDC_DESIGNTREE_ADD_FOOTER_ODD 37566
Add Header

IDC_DESIGNTREE_ADD_HEADER_ALL 37567
Add Header Even

IDC_DESIGNTREE_ADD_HEADER_EVEN 37568
Add Header First

IDC_DESIGNTREE_ADD_HEADER_FIRST 39995
Add Header Last

IDC_DESIGNTREE_ADD_HEADER_LAST 40075
Add Header Odd

IDC_DESIGNTREE_ADD_HEADER_ODD 37569
Add Main Template

IDC_DESIGNTREE_ADD_MAIN_TEMPLATE 37866
Add Module...

IDC_DESIGNTREE_ADD_MODULE 37865
Add Design Fragment

IDC_DESIGNTREE_ADD_NAMED_TEMPLATE 37570
Create Design Fragment

IDC_DESIGNTREE_CREATE_NAMED_TEMPLATE 40181

contd...

Goto Definition

IDC_DESIGNTREE_GOTO_DEF 37869
Move Down

IDC_DESIGNTREE_MOVE_DOWN 37868
Move Up

IDC_DESIGNTREE_MOVE_UP 37867
Bring Forward

IDC_DESIGN_BRING_FORWARD 37961
Bring To Front

IDC_DESIGN_BRING_TO_FRONT 37959
Edit Blueprint Image Properties...

IDC_DESIGN_EDIT_BLUEPRINT_IMAGE_PROPERTIES 40187
Edit Line Properties...

IDC_DESIGN_EDIT_LINE_PROPERTIES 40188
Enable Snap to Grid

IDC_DESIGN_ENABLE_SNAP_TO_GRID 37963
Design Filter

IDC_DESIGN_FILTER 37699
IDC_DESIGN_FILTER_TEMPLATE_COMBO 37706
<table>
<thead>
<tr>
<th>Command</th>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto-fit to Paper size</td>
<td>IDC_DESIGN_LAYOUTCONTAINER_AUTOFIT_TO_PAPER_SIZE</td>
<td>40190</td>
</tr>
<tr>
<td>Add CSS File...</td>
<td>IDC_DESIGN_OVERVIEW_ADD_CSS</td>
<td>37890</td>
</tr>
<tr>
<td>Add Parameter...</td>
<td>IDC_DESIGN_OVERVIEW_ADD_PARAMETER</td>
<td>37793</td>
</tr>
<tr>
<td>Add XSLT File...</td>
<td>IDC_DESIGN_OVERVIEW_ADD_XSLT</td>
<td>37871</td>
</tr>
<tr>
<td>Change DB Connection...</td>
<td>IDC_DESIGN_OVERVIEW_CHANGE_DB_CONNECTION</td>
<td>37891</td>
</tr>
<tr>
<td>Open Module</td>
<td>IDC_DESIGN_OVERVIEW_OPEN_MODULE</td>
<td>37931</td>
</tr>
<tr>
<td>Active</td>
<td>IDC_DESIGN_OVERVIEW_PAGE_LAYOUT_ACTIVE</td>
<td>37922</td>
</tr>
<tr>
<td>Edit Page Properties...</td>
<td>IDC_DESIGN_OVERVIEW_PAGE_LAYOUT_PROPERTIES</td>
<td>37921</td>
</tr>
<tr>
<td>Select DB Tables and Views...</td>
<td>IDC_DESIGN_OVERVIEW_SELECT_DB_TABLES_AND_VIEWS</td>
<td>37892</td>
</tr>
<tr>
<td>Select XML Schema from DB...</td>
<td>IDC_DESIGN_OVERVIEW_SELECT_SCHEMA_FROM_DB</td>
<td>37621</td>
</tr>
<tr>
<td>Select Working XML DB Field...</td>
<td>IDC_DESIGN_OVERVIEW_SELECT_WORKING_XML_DB_FIELD</td>
<td>37622</td>
</tr>
<tr>
<td>Edit XBRL Taxonomy Source</td>
<td>IDC_DESIGN_OVERVIEW_XBRL_TAXONOMY_PROPERTIES</td>
<td>37945</td>
</tr>
<tr>
<td>Properties...</td>
<td>IDC_DESIGN_SEND_BACKWARD</td>
<td>37960</td>
</tr>
<tr>
<td>Send Backward</td>
<td>IDC_DESIGN_SEND_TO_BACK</td>
<td>37958</td>
</tr>
<tr>
<td>Show Grid</td>
<td>IDC_DESIGN_SHOW_GRID</td>
<td>37962</td>
</tr>
<tr>
<td>Show Large Design Markups</td>
<td>IDC_DESIGN_SHOW_LARGE_MARKUPS</td>
<td>37954</td>
</tr>
<tr>
<td>Show Small Design Markups</td>
<td>IDC_DESIGN_SHOW_SMALL_MARKUPS</td>
<td>37953</td>
</tr>
<tr>
<td>Auto-resize Layout Box</td>
<td>IDC_DESIGN_TEXTBOX_AUTORESIZE</td>
<td>40189</td>
</tr>
<tr>
<td>IDC_DEVHELPER_ADDERRORTESTCASE</td>
<td>36014</td>
<td></td>
</tr>
<tr>
<td>IDC_DEVHELPER_ADDTESTCASE</td>
<td>36005</td>
<td></td>
</tr>
<tr>
<td>IDC_DEVHELPER_HTMLVIEW_SRC</td>
<td>36803</td>
<td></td>
</tr>
<tr>
<td>contd...</td>
<td>IDC_EDIT_ALT</td>
<td>37841</td>
</tr>
<tr>
<td>Edit Format...</td>
<td>IDC_EDIT_AUTO_NUMBER_FORMAT</td>
<td>37834</td>
</tr>
<tr>
<td>Edit Bookmark Name...</td>
<td>IDC_EDIT_BOOKMARK_NAME</td>
<td>37838</td>
</tr>
<tr>
<td>Edit Checked Values...</td>
<td>IDC_EDIT_CHECKED_VALUES</td>
<td>37835</td>
</tr>
<tr>
<td>Edit Combo Box entry values...</td>
<td>IDC_EDIT_COMBOBOX_ENTRY_VALUES</td>
<td>37836</td>
</tr>
<tr>
<td>Edit XPath Filter...</td>
<td>IDC_EDIT_FILTER</td>
<td>37924</td>
</tr>
<tr>
<td>Edit Global Resource...</td>
<td>IDC_EDIT_GLOBAL_RESOURCES</td>
<td>36686</td>
</tr>
<tr>
<td>Edit Input Formatting...</td>
<td>IDC_EDIT_INPUT_FORMATTING</td>
<td>37833</td>
</tr>
<tr>
<td>Edit File in XMLSpy</td>
<td>IDC_EDIT_IN_XMLSPY</td>
<td>37575</td>
</tr>
<tr>
<td>Edit Name...</td>
<td>IDC_EDIT_NAME</td>
<td>37840</td>
</tr>
<tr>
<td>Edit Condition</td>
<td>IDC_EDIT_OPTION_CONDITION</td>
<td>37572</td>
</tr>
<tr>
<td>Edit Paragraph Type...</td>
<td>IDC_EDIT_PARAGRAPH_TYPE</td>
<td>37845</td>
</tr>
<tr>
<td>Edit Scope...</td>
<td>IDC_EDIT_SCOPE</td>
<td>37844</td>
</tr>
<tr>
<td>Edit Template Match...</td>
<td>IDC_EDIT_TEMPLATE_MATCH</td>
<td>40182</td>
</tr>
<tr>
<td>Edit Update XML Node...</td>
<td>IDC_EDIT_UPDATE_XML_NODE</td>
<td>37853</td>
</tr>
</tbody>
</table>
contd...

<table>
<thead>
<tr>
<th>Function</th>
<th>Resource ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>IDC_GLOBALRESOURCES_MAINDLG_DELETESOURCE</td>
<td>37422</td>
</tr>
<tr>
<td>&lt;placeholder&gt;</td>
<td>IDC_GLOBALRESOURCES_SUBMENUENTRY2</td>
<td>37409</td>
</tr>
<tr>
<td>&lt;placeholder&gt;</td>
<td>IDC_GLOBALRESOURCES_SUBMENUENTRY3</td>
<td>37410</td>
</tr>
<tr>
<td>&lt;placeholder&gt;</td>
<td>IDC_GLOBALRESOURCES_SUBMENUENTRY4</td>
<td>37411</td>
</tr>
<tr>
<td>&lt;placeholder&gt;</td>
<td>IDC_GLOBALRESOURCES_SUBMENUENTRY5</td>
<td>37412</td>
</tr>
<tr>
<td>&lt;placeholder&gt;</td>
<td>IDC_GLOBALRESOURCES_SUBMENUENTRY6</td>
<td>37413</td>
</tr>
<tr>
<td>&lt;placeholder&gt;</td>
<td>IDC_GLOBALRESOURCES_SUBMENUENTRY7</td>
<td>37414</td>
</tr>
<tr>
<td>&lt;placeholder&gt;</td>
<td>IDC_GLOBALRESOURCES_SUBMENUENTRY8</td>
<td>37415</td>
</tr>
<tr>
<td>&lt;placeholder&gt;</td>
<td>IDC_GLOBALRESOURCES_SUBMENUENTRY9</td>
<td>37416</td>
</tr>
<tr>
<td>&lt;placeholder&gt;</td>
<td>IDC_GLOBALRESOURCES_SUBMENUENTRY_LAST</td>
<td>37417</td>
</tr>
<tr>
<td>Goto Next Bookmark</td>
<td>IDC_GOTONEXTBOOKMARK</td>
<td>37583</td>
</tr>
<tr>
<td>Goto Previous Bookmark</td>
<td>IDC_GOTOPREVBOOKMARK</td>
<td>37584</td>
</tr>
<tr>
<td>Edit Authentic Properties</td>
<td>IDC_GOTO_AUTHENTIC_PROPERTIES</td>
<td>37852</td>
</tr>
<tr>
<td>Group by...</td>
<td>IDC_GROUP</td>
<td>37874</td>
</tr>
<tr>
<td></td>
<td>IDC_ICDBWND_ADD_SELECT_STATEMENT</td>
<td>36663</td>
</tr>
<tr>
<td></td>
<td>IDC_ICDBWND_DELETE_DSN</td>
<td>36666</td>
</tr>
<tr>
<td>Edit a SELECT Statement...</td>
<td>IDC_ICDBWND_EDIT_LOCALVIEW</td>
<td>36676</td>
</tr>
<tr>
<td></td>
<td>IDC_ICDBWND_MANAGE_LOCAL_RELATIONS</td>
<td>36688</td>
</tr>
<tr>
<td>Function</td>
<td>ID</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Locate File...</td>
<td>IDC_ICDBWNDS_NEW_DSN</td>
<td>36661</td>
</tr>
<tr>
<td>IDC_ICDBWNDS_OPENINEXPLORER</td>
<td>36540</td>
<td></td>
</tr>
<tr>
<td>IDC_ICDBWNDS_REFRESH_DSN</td>
<td>36667</td>
<td></td>
</tr>
<tr>
<td>Remove SELECT statement</td>
<td>IDC_ICDBWNDS_REMOVE_LOCALVIEW</td>
<td>36680</td>
</tr>
<tr>
<td>IDC_ICDBWNDS_BACK_COLOR</td>
<td>36543</td>
<td></td>
</tr>
<tr>
<td>IDC_ICDBWND_BOLD</td>
<td>36544</td>
<td></td>
</tr>
<tr>
<td>Add a SELECT Statement...</td>
<td>IDC_ICDBWND_CREATE_LOCALVIEW</td>
<td>36677</td>
</tr>
<tr>
<td>IDC_ICDBWND_EDIT_DSN</td>
<td>36662</td>
<td></td>
</tr>
<tr>
<td>Generate and add a SELECT Statement.</td>
<td>IDC_ICDBWND_GENERATE_SQL_FOR_LOCALVIEW</td>
<td>36668</td>
</tr>
<tr>
<td>IDC_ICDBWND_ITALIC</td>
<td>36545</td>
<td></td>
</tr>
<tr>
<td>Remove from Online Browser</td>
<td>IDC_ICDBWND_REMOVE_FROM_BROWSEVIEW</td>
<td>36683</td>
</tr>
<tr>
<td>IDC_ICDBWND_TEXT_COLOR</td>
<td>36546</td>
<td></td>
</tr>
<tr>
<td>IDC_ICDBWND_TOOLBAR_COLOR</td>
<td>36547</td>
<td></td>
</tr>
<tr>
<td>IDC_ICDBWND_UNDERLINE</td>
<td>36548</td>
<td></td>
</tr>
</tbody>
</table>

contd...

<table>
<thead>
<tr>
<th>Function</th>
<th>ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Active and Related Files</td>
<td>IDC_ICPROJECTGUI_ADD_ACTIVE_ANDRELATED_FILES</td>
<td>37217</td>
</tr>
<tr>
<td>IDC_ICPROJECTGUI_ADD_ACTIVE_FILE</td>
<td>37216</td>
<td></td>
</tr>
<tr>
<td>IDC_ICPROJECTGUI_ADD_EXT_FOLDER</td>
<td>37219</td>
<td></td>
</tr>
<tr>
<td>IDC_ICPROJECTGUI_ADD_EXT_URL_FOLDER</td>
<td>37220</td>
<td></td>
</tr>
<tr>
<td>IDC_ICPROJECTGUI_ADD_FILES</td>
<td>37213</td>
<td></td>
</tr>
<tr>
<td>IDC_ICPROJECTGUI_ADD_FOLDER</td>
<td>37218</td>
<td></td>
</tr>
<tr>
<td>IDC_ICPROJECTGUI_ADD_GLOBAL_RESOURCE</td>
<td>37238</td>
<td></td>
</tr>
<tr>
<td>IDC_ICPROJECTGUI_ADD_URL</td>
<td>37214</td>
<td></td>
</tr>
<tr>
<td>IDC_ICPROJECTGUI_OPEN_CONTAINING_FOLDER</td>
<td>37237</td>
<td></td>
</tr>
<tr>
<td>IDC_ICPROJECTGUI_OPEN_IN_EXTERNAL_APP</td>
<td>37236</td>
<td></td>
</tr>
<tr>
<td>IDC_ICPROJECTGUI_OPEN_IN_MAPFORCE</td>
<td>37234</td>
<td></td>
</tr>
<tr>
<td>IDC_ICPROJECTGUI_OPEN_IN_XMLSPY</td>
<td>37233</td>
<td></td>
</tr>
<tr>
<td>IDC_ICPROJECTGUI_PROPERTIES</td>
<td>37222</td>
<td></td>
</tr>
<tr>
<td>IDC_ICPROJECTGUI_REFRESH_EXT_FOLDER</td>
<td>37221</td>
<td></td>
</tr>
<tr>
<td>IDC_IMPORT_MODULE_FROM_FILE</td>
<td>37879</td>
<td></td>
</tr>
<tr>
<td>IDC_IMPORT_STYLESHEET_FROM_FILE</td>
<td>37880</td>
<td></td>
</tr>
<tr>
<td>IDC_INSERT_DESIGN_ELEMENT</td>
<td>40192</td>
<td></td>
</tr>
<tr>
<td>IDC_INSERT_DESIGN_ELEM_INTERACTIVE_AUTOCALC</td>
<td>40164</td>
<td></td>
</tr>
<tr>
<td>IDC_INSERT_DESIGN_ELEM_INTERACTIVE_AUTO_CREATE_TEMPLATES</td>
<td>40193</td>
<td></td>
</tr>
<tr>
<td>IDC_INSERT_DESIGN_ELEM_INTERACTIVE_CHECKBOX</td>
<td>40177</td>
<td></td>
</tr>
<tr>
<td>IDC_INSERT_DESIGN_ELEM_INTERACTIVE_COMBOBOX</td>
<td>40176</td>
<td></td>
</tr>
<tr>
<td>IDC_INSERT_DESIGN_ELEM_INTERACTIVE_CONTENTS</td>
<td>40165</td>
<td></td>
</tr>
<tr>
<td>Insert Input Field</td>
<td>IDC_INSERT_DESIGN_ELEM_INTERACTIVE_EDITFIELD</td>
<td>40174</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>Insert Multiline Input Field</td>
<td>IDC_INSERT_DESIGN_ELEM_INTERACTIVE_EDITFIELD_MULTILINE</td>
<td>40175</td>
</tr>
<tr>
<td>Insert Image...</td>
<td>IDC_INSERT_DESIGN_ELEM_INTERACTIVE_IMAGE</td>
<td>40173</td>
</tr>
<tr>
<td>Insert Layout Box</td>
<td>IDC_INSERT_DESIGN_ELEM_INTERACTIVE_LAYOUT_BOX</td>
<td>40157</td>
</tr>
<tr>
<td>Insert Layout Container</td>
<td>IDC_INSERT_DESIGN_ELEM_INTERACTIVE_LAYOUT_CONTAINER</td>
<td>40156</td>
</tr>
<tr>
<td>Insert Bullets and Numbering...</td>
<td>IDC_INSERT_DESIGN_ELEM_INTERACTIVE_LIST</td>
<td>40171</td>
</tr>
<tr>
<td>Insert Paragraph</td>
<td>IDC_INSERT_DESIGN_ELEM_INTERACTIVE_PARAGRAPH</td>
<td>40167</td>
</tr>
<tr>
<td>Insert Radio Button...</td>
<td>IDC_INSERT_DESIGN_ELEM_INTERACTIVE_RADIOBUTTON</td>
<td>40178</td>
</tr>
<tr>
<td>Insert Line</td>
<td>IDC_INSERT_DESIGN_ELEM_INTERACTIVE_SHAPE_LINE</td>
<td>40186</td>
</tr>
<tr>
<td>Insert Table...</td>
<td>IDC_INSERT_DESIGN_ELEM_INTERACTIVE_TABLE</td>
<td>40169</td>
</tr>
<tr>
<td>Insert Template</td>
<td>IDC_INSERT_DESIGN_ELEM_INTERACTIVE_TEMPLATE</td>
<td>40166</td>
</tr>
<tr>
<td>Insert User-Defined Template</td>
<td>IDC_INSERT_DESIGN_ELEM_INTERACTIVE_USER_DEFINED_TEMPLATE</td>
<td>40183</td>
</tr>
</tbody>
</table>

contd...

<p>| Insert List                | IDC_LIST_INSERT_FIX                                           | 36028 |
| Move Branch Down           | IDC_MOVE_OPTION_DOWN                                          | 37625 |
| Move Branch Up             | IDC_MOVE_OPTION_UP                                           | 37626 |
| Open Design                | IDC_OPEN_DESIGN_FROM_FILE                                    | 37877 |
| Apply/Change Paragraph     | IDC_PARAGRAPH_PLACEHOLDER                                    | 37821 |
| Add as new Schema Source... | IDC_PROJECT_ADD_AS_XBRL_SCHEMASOURCE                        | 36133 |
| Add as new Schema Source... | IDC_PROJECT_ADD_AS_XSD_SCHEMASOURCE                          | 36045 |
| Assign as Schema File      | IDC_PROJECT_ASSIGN_AS_SCHEMA                                  | 36048 |
| Assign as Template XBRL File | IDC_PROJECT_ASSIGN_AS_TEMPLATE_XBRL                      | 36135 |
| Assign as Template XML File | IDC_PROJECT_ASSIGN_AS_TEMPLATE_XML                         | 36047 |
| Assign as Working XBRL File | IDC_PROJECT_ASSIGN_AS_WORKING_XBRL                        | 36134 |
| Assign as Working XML File | IDC_PROJECT_ASSIGN_AS_WORKING_XML                           | 36046 |
| Create new Design          | IDC_PROJECT_CREATE_NEW_FROM_HTML                             | 36044 |
| Create new Design...       | IDC_PROJECT_CREATE_NEW_FROM_XBRL                             | 36132 |
| Create new Design...       | IDC_PROJECT_CREATE_NEW_FROM_XSD                              | 36043 |
| Import as Module           | IDC_PROJECT_IMPORT_AS_MODULE                                 | 36042 |
| Import into Style Repository | IDC_PROJECT_IMPORT_AS_STYLESHEET                             | 36049 |
| Insert Image to Design     | IDC_PROJECT_INSERT_IMAGE                                     | 36051 |
| Open Design                | IDC_PROJECT_OPEN_IN_STYLEVISION                              | 36041 |
| Remove Tag Only            | IDC_REMOVE                                                   | 37521 |
| Remove All Bookmarks       | IDC_REMOVEALLBOOKMARKS                                       | 37629 |
| Remove                     | IDC_REMOVE_PAGE_LAYOUT                                       | 37630 |</p>
<table>
<thead>
<tr>
<th>Description</th>
<th>IDC Constant</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove Global Template</td>
<td>IDC_REMOVE_TEMPLATE</td>
<td>37631</td>
</tr>
<tr>
<td>Rename</td>
<td>IDC_RENAME_NODE</td>
<td>37632</td>
</tr>
<tr>
<td>Auto-collapse on Synchronize</td>
<td>IDC_SCHEMASOURCES_AUTOCOLLAPSE</td>
<td>37873</td>
</tr>
<tr>
<td>Convert Selection to Attributes</td>
<td>IDC_SCHEMASOURCES_CONVERT_TO_ATTRIBUTES</td>
<td>37643</td>
</tr>
<tr>
<td>Convert Selected Table / List to Attributes</td>
<td>IDC_SCHEMASOURCES_CONVERT_TO_ATTRIBUTES_TABLE</td>
<td>37644</td>
</tr>
<tr>
<td>Convert Selection to Elements</td>
<td>IDC_SCHEMASOURCES_CONVERT_TO_ELEMENTS</td>
<td>37646</td>
</tr>
<tr>
<td>Convert Selected Table / List to Elements</td>
<td>IDC_SCHEMASOURCES_CONVERT_TO_ELEMENTS_TABLE</td>
<td>37647</td>
</tr>
<tr>
<td>Surround Selection with Element</td>
<td>IDC_SCHEMASOURCES_SURROUND_TEMPLATE</td>
<td>37648</td>
</tr>
<tr>
<td>Add DB Schema...</td>
<td>IDC_SCHEMASOURCE_ADD_FROM_DB</td>
<td>37649</td>
</tr>
<tr>
<td>Add Schema for XML Column in DB table...</td>
<td>IDC_SCHEMASOURCE_ADD_FROM_DBCELL</td>
<td>37862</td>
</tr>
<tr>
<td>Add XML Schema/DTD/XML...</td>
<td>IDC_SCHEMASOURCE_ADD_FROM_FILE</td>
<td>37650</td>
</tr>
<tr>
<td>Add XBRL Taxonomy...</td>
<td>IDC_SCHEMASOURCE_ADD_FROM_XBRL_TAXONOMY</td>
<td>37917</td>
</tr>
<tr>
<td>Add User-Defined Schema</td>
<td>IDC_SCHEMASOURCE_ADD_USER</td>
<td>37651</td>
</tr>
<tr>
<td>Show: %s</td>
<td>IDC_SHOW_OPTION_FIRST</td>
<td>37652</td>
</tr>
<tr>
<td>Sort by...</td>
<td>IDC_SORT</td>
<td>37653</td>
</tr>
<tr>
<td>Collapse All</td>
<td>IDC_STYLES_COLLAPSE</td>
<td>37654</td>
</tr>
<tr>
<td>Expand All</td>
<td>IDC_STYLES_EXPANDALL</td>
<td>37655</td>
</tr>
<tr>
<td>Display All Branches</td>
<td>IDC_STYLEVISIONGUI_CONDITION_SHOW_ALL_BRANCHES</td>
<td>37794</td>
</tr>
<tr>
<td>Table</td>
<td>IDC_TABLE</td>
<td>37734</td>
</tr>
<tr>
<td>Insert Table...</td>
<td>IDC_TABLE_INSERT_FIX</td>
<td>37663</td>
</tr>
<tr>
<td>Insert / Remove Bookmark</td>
<td>IDC_TOGGLE_BOOKMARK</td>
<td>37673</td>
</tr>
<tr>
<td>Global Resources</td>
<td>IDC_TOOLBAR_ALTOVA_GLOBAL_RESOURCES</td>
<td>36029</td>
</tr>
<tr>
<td>Standard Toolbar</td>
<td>IDC_TOOLBAR_STANDARD</td>
<td>36030</td>
</tr>
<tr>
<td>Collapse All</td>
<td>IDC_TREE_COLLAPSE</td>
<td>37677</td>
</tr>
<tr>
<td>Collapse To this Point</td>
<td>IDC_TREE_COLLAPSE_KNOT</td>
<td>37678</td>
</tr>
<tr>
<td>Expand All</td>
<td>IDC_TREE_EXPANDALL</td>
<td>37679</td>
</tr>
<tr>
<td>Expand From this Point</td>
<td>IDC_TREE_EXPAND_KNOT</td>
<td>37680</td>
</tr>
<tr>
<td>Expand / Collapse All to this Level</td>
<td>IDC_TREESAME_LEVEL</td>
<td>37681</td>
</tr>
<tr>
<td>Unassign Template XBRL File</td>
<td>IDC_UNASSIGN_TEMPLATE_XBRL_FILE</td>
<td>37940</td>
</tr>
<tr>
<td>Unassign Working XBRL File</td>
<td>IDC_UNASSIGN_WORKING_XBRL_FILE</td>
<td>37941</td>
</tr>
<tr>
<td>Use Global Template</td>
<td>IDC_USE_GLOBAL_TEMPLATE</td>
<td>37685</td>
</tr>
<tr>
<td>valign bottom</td>
<td>IDC_VALIGN_BOTTOM</td>
<td>37687</td>
</tr>
<tr>
<td>valign center</td>
<td>IDC_VALIGN_CENTER</td>
<td>37688</td>
</tr>
</tbody>
</table>
Add Schema...
Commit Changes
Decomposition
Drop Schema
Remove Drop Flag

contd...

View Schema
Set/ remove important flag
Reset
List All
List Non-Empty
Grouped
XPath
Append Attribute
Append Element
Add Child Attribute
Add Child Element
Convert to Attribute / Element
Make / Remove Global Template
Insert Attribute
Insert Element
All Templates Serve as Level
Remove Item
Rename
Set as Main Schema Source
Synchronize tree
No Template Serves as Level
Add
Set/ remove important flag
Insert
Move Down
Move Up
contd...

<table>
<thead>
<tr>
<th>Button</th>
<th>IDM/IDH Name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reload all external CSS files</td>
<td>IDR_STYLES_RELOAD</td>
<td>37728</td>
</tr>
<tr>
<td>Remove / Reset</td>
<td>IDR_STYLES_REMOVE</td>
<td>37729</td>
</tr>
<tr>
<td>Reset</td>
<td>IDR_STYLES_RESET</td>
<td>37730</td>
</tr>
<tr>
<td>List All</td>
<td>IDR_STYLES_VIEWMODE_ALPHA</td>
<td>37731</td>
</tr>
<tr>
<td>List Non-Empty</td>
<td>IDR_STYLES_VIEWMODE_ALPHA_SET</td>
<td>37732</td>
</tr>
<tr>
<td>Grouped</td>
<td>IDR_STYLES_VIEWMODE_GROUPED</td>
<td>37733</td>
</tr>
<tr>
<td>Add Active File to Project</td>
<td>ID_ADDACTIVEFILETOPROJECT</td>
<td>36549</td>
</tr>
<tr>
<td>Add Files to Project ...</td>
<td>ID_ADDFILESTOPROJECT</td>
<td>36550</td>
</tr>
<tr>
<td></td>
<td>ID_BUTTON_OPTIONS</td>
<td>36551</td>
</tr>
<tr>
<td></td>
<td>ID_BUTTON_SOURCE</td>
<td>36552</td>
</tr>
<tr>
<td>Connect</td>
<td>ID_CONNECT</td>
<td>36553</td>
</tr>
<tr>
<td>Connect to all Data Sources</td>
<td>ID_CONNECTTOALLDATASOURCES</td>
<td>36554</td>
</tr>
<tr>
<td>-14:00</td>
<td>ID_CONTENT_TIMEZONES_M14_0000</td>
<td>18356</td>
</tr>
<tr>
<td>-13:00</td>
<td>ID_CONTENT_TIMEZONES_M14_0100</td>
<td>18357</td>
</tr>
<tr>
<td>-12:00</td>
<td>ID_CONTENT_TIMEZONES_M14_0200</td>
<td>18358</td>
</tr>
<tr>
<td>-11:00</td>
<td>ID_CONTENT_TIMEZONES_M14_0300</td>
<td>18359</td>
</tr>
<tr>
<td>-10:00</td>
<td>ID_CONTENT_TIMEZONES_M14_0400</td>
<td>18360</td>
</tr>
<tr>
<td>-09:00</td>
<td>ID_CONTENT_TIMEZONES_M14_0500</td>
<td>18361</td>
</tr>
<tr>
<td>-08:00</td>
<td>ID_CONTENT_TIMEZONES_M14_0600</td>
<td>18362</td>
</tr>
<tr>
<td>-07:00</td>
<td>ID_CONTENT_TIMEZONES_M14_0700</td>
<td>18363</td>
</tr>
<tr>
<td>-06:00</td>
<td>ID_CONTENT_TIMEZONES_M14_0800</td>
<td>18364</td>
</tr>
<tr>
<td>-05:00</td>
<td>ID_CONTENT_TIMEZONES_M14_0900</td>
<td>18365</td>
</tr>
<tr>
<td>-04:00</td>
<td>ID_CONTENT_TIMEZONES_M14_1000</td>
<td>18366</td>
</tr>
<tr>
<td>-03:00</td>
<td>ID_CONTENT_TIMEZONES_M14_1100</td>
<td>18367</td>
</tr>
<tr>
<td>-02:00</td>
<td>ID_CONTENT_TIMEZONES_M14_1200</td>
<td>18368</td>
</tr>
<tr>
<td>-01:00</td>
<td>ID_CONTENT_TIMEZONES_M14_1300</td>
<td>18369</td>
</tr>
<tr>
<td>00:00</td>
<td>ID_CONTENT_TIMEZONES_M14_1400</td>
<td>18370</td>
</tr>
<tr>
<td>+01:00</td>
<td>ID_CONTENT_TIMEZONES_M14_1500</td>
<td>18371</td>
</tr>
<tr>
<td>+02:00</td>
<td>ID_CONTENT_TIMEZONES_M14_1600</td>
<td>18372</td>
</tr>
<tr>
<td>+03:00</td>
<td>ID_CONTENT_TIMEZONES_M14_1700</td>
<td>18373</td>
</tr>
<tr>
<td>+04:00</td>
<td>ID_CONTENT_TIMEZONES_M14_1800</td>
<td>18374</td>
</tr>
<tr>
<td>+05:00</td>
<td>ID_CONTENT_TIMEZONES_M14_1900</td>
<td>18375</td>
</tr>
<tr>
<td>+06:00</td>
<td>ID_CONTENT_TIMEZONES_M14_2000</td>
<td>18376</td>
</tr>
<tr>
<td>+07:00</td>
<td>ID_CONTENT_TIMEZONES_M14_2100</td>
<td>18377</td>
</tr>
<tr>
<td>+08:00</td>
<td>ID_CONTENT_TIMEZONES_M14_2200</td>
<td>18378</td>
</tr>
<tr>
<td>+09:00</td>
<td>ID_CONTENT_TIMEZONES_M14_2300</td>
<td>18379</td>
</tr>
<tr>
<td>Time</td>
<td>Category</td>
<td>ID</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>+10:00</td>
<td>ID_CONTENT_TIMEZONES_M14_2400</td>
<td>18380</td>
</tr>
<tr>
<td>+11:00</td>
<td>ID_CONTENT_TIMEZONES_M14_2500</td>
<td>18381</td>
</tr>
<tr>
<td>+12:00</td>
<td>ID_CONTENT_TIMEZONES_M14_2600</td>
<td>18382</td>
</tr>
<tr>
<td>+13:00</td>
<td>ID_CONTENT_TIMEZONES_M14_2700</td>
<td>18383</td>
</tr>
<tr>
<td>+14:00</td>
<td>ID_CONTENT_TIMEZONES_M14_2800</td>
<td>18384</td>
</tr>
</tbody>
</table>

contd...

<table>
<thead>
<tr>
<th>Time</th>
<th>Category</th>
<th>ID</th>
<th>Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>14:59</td>
<td>ID_CONTENT_TIMEZONES_MIN00</td>
<td>18341</td>
<td></td>
</tr>
<tr>
<td>29:59:00</td>
<td>ID_CONTENT_TIMEZONES_MIN30</td>
<td>18343</td>
<td></td>
</tr>
<tr>
<td>44:59:00</td>
<td>ID_CONTENT_TIMEZONES_MIN45</td>
<td>18344</td>
<td></td>
</tr>
<tr>
<td>No TZ</td>
<td>ID_CONTENT_TIMEZONES_NOTIMEZONE</td>
<td>18354</td>
<td></td>
</tr>
<tr>
<td>UTC</td>
<td>ID_CONTENT_TIMEZONES_UTCTIMEZONE</td>
<td>18355</td>
<td></td>
</tr>
<tr>
<td>Create Folder</td>
<td>ID_CREATEFOLDER</td>
<td>36555</td>
<td></td>
</tr>
<tr>
<td>Create Contents</td>
<td>ID_CREATE_ALL_CONTENTS</td>
<td>37735</td>
<td></td>
</tr>
<tr>
<td>Create Bullets and Numbering</td>
<td>ID_CREATE_LIST</td>
<td>37736</td>
<td></td>
</tr>
<tr>
<td>Create Paragraph</td>
<td>ID_CREATE_PARAGRAPH</td>
<td>37737</td>
<td></td>
</tr>
<tr>
<td>Create Table...</td>
<td>ID_CREATE_TABLE</td>
<td>37738</td>
<td></td>
</tr>
<tr>
<td>Create Templates</td>
<td>ID_CREATE_TEMPLATES</td>
<td>37739</td>
<td></td>
</tr>
<tr>
<td>Create XBRL Label</td>
<td>ID_CREATE_XBRL_LABEL</td>
<td>37915</td>
<td></td>
</tr>
<tr>
<td>Create XBRL Label as Text</td>
<td>ID_CREATE_XBRL_LABEL_AS_TEXT</td>
<td>37916</td>
<td></td>
</tr>
<tr>
<td>Create XBRL Table...</td>
<td>ID_CREATE_XBRL_TABLE</td>
<td>37918</td>
<td></td>
</tr>
<tr>
<td>Create XBRL Template</td>
<td>ID_CREATE_XBRL_TEMPLATE</td>
<td>37914</td>
<td></td>
</tr>
<tr>
<td>Auto-collapse on Synchronize</td>
<td>ID_DESIGNTREE_AUTOCOLLAPSE</td>
<td>37740</td>
<td></td>
</tr>
<tr>
<td>Remove</td>
<td>ID_DESIGNTREE_REMOVAL</td>
<td>37741</td>
<td></td>
</tr>
<tr>
<td>Rename</td>
<td>ID_DESIGNTREE_RENAME</td>
<td>37872</td>
<td></td>
</tr>
<tr>
<td>Synchronize Tree</td>
<td>ID_DESIGNTREE_SYNC</td>
<td>37742</td>
<td></td>
</tr>
<tr>
<td>Disconnect</td>
<td>ID_DISCONNECT</td>
<td>38014</td>
<td></td>
</tr>
<tr>
<td>Disconnect from all Data Sources</td>
<td>ID_DISCONNECTFROMALLDATASOURCES</td>
<td>36560</td>
<td></td>
</tr>
<tr>
<td>Edit Data</td>
<td>ID_EDITRESULTDATA</td>
<td>38022</td>
<td></td>
</tr>
<tr>
<td>Delete</td>
<td>ID_EDIT_DROP_DBOBJECT</td>
<td>38019</td>
<td></td>
</tr>
<tr>
<td>Delete</td>
<td>ID_EDIT_DROP_FAV_OBJECT</td>
<td>36562</td>
<td></td>
</tr>
<tr>
<td>Rename</td>
<td>ID_EDIT_RENAME</td>
<td>38020</td>
<td></td>
</tr>
<tr>
<td>Execute All SQL Files</td>
<td>ID_EXECUTEALLSQLFILES</td>
<td>36564</td>
<td></td>
</tr>
<tr>
<td>ID_FIELDTYPES</td>
<td>Description</td>
<td>Code</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>AD' / 'BC</td>
<td></td>
<td>37744</td>
<td></td>
</tr>
<tr>
<td>AM' / 'PM</td>
<td></td>
<td>37745</td>
<td></td>
</tr>
<tr>
<td>CE' / 'BCE</td>
<td></td>
<td>37746</td>
<td></td>
</tr>
<tr>
<td>Century ('2004' -&gt; '20')</td>
<td></td>
<td>37747</td>
<td></td>
</tr>
<tr>
<td>Century ('2004' -&gt; '21')</td>
<td></td>
<td>37748</td>
<td></td>
</tr>
<tr>
<td>Comma</td>
<td></td>
<td>37749</td>
<td></td>
</tr>
<tr>
<td>contd...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Day as number</td>
<td>ID_FIELDTYPES_DAYASNUMBER</td>
<td>37750</td>
<td></td>
</tr>
<tr>
<td>Day-in-year as number</td>
<td>ID_FIELDTYPES_DAYINYEARASNUMBER</td>
<td>37751</td>
<td></td>
</tr>
<tr>
<td>Digit or leading zero</td>
<td>ID_FIELDTYPES_DIGITORLEADINGZERO</td>
<td>37752</td>
<td></td>
</tr>
<tr>
<td>Digit or space</td>
<td>ID_FIELDTYPES_DIGITSPACE</td>
<td>37753</td>
<td></td>
</tr>
<tr>
<td>Digit separator</td>
<td>ID_FIELDTYPES_DIGITSEPARATOR</td>
<td>37754</td>
<td></td>
</tr>
<tr>
<td>Fraction (2 digits)</td>
<td>ID_FIELDTYPES_FRACTION2</td>
<td>37755</td>
<td></td>
</tr>
<tr>
<td>Fraction (2 or more digits)</td>
<td>ID_FIELDTYPES_FRACTION2PLUS</td>
<td>37756</td>
<td></td>
</tr>
<tr>
<td>Hour as number (1 - 12)</td>
<td>ID_FIELDTYPES_HOURASNUMBER12</td>
<td>37757</td>
<td></td>
</tr>
<tr>
<td>Hour as number</td>
<td>ID_FIELDTYPES_HOURASNUMBER24</td>
<td>37758</td>
<td></td>
</tr>
<tr>
<td>Minutes as number</td>
<td>ID_FIELDTYPES_MINUTESASNUMBER</td>
<td>37759</td>
<td></td>
</tr>
<tr>
<td>Month as number</td>
<td>ID_FIELDTYPES_MONTHASNUMBER</td>
<td>37760</td>
<td></td>
</tr>
<tr>
<td>Month as Text, abbrev.</td>
<td>ID_FIELDTYPES_MONTHASTEXTABBREV</td>
<td>37761</td>
<td></td>
</tr>
<tr>
<td>Month as Text, long format</td>
<td>ID_FIELDTYPES_MONTHASTEXTLONGFORMAT</td>
<td>37762</td>
<td></td>
</tr>
<tr>
<td>Number (3 digits)</td>
<td>ID_FIELDTYPES_NUMBER3</td>
<td>37763</td>
<td></td>
</tr>
<tr>
<td>Number (4 digits with leading 0's)</td>
<td>ID_FIELDTYPES_NUMBER4_0</td>
<td>37764</td>
<td></td>
</tr>
<tr>
<td>Number (6 digits with separator)</td>
<td>ID_FIELDTYPES_NUMBER6</td>
<td>37765</td>
<td></td>
</tr>
<tr>
<td>Quoted text</td>
<td>ID_FIELDTYPES_QUOTEDTEXT</td>
<td>37766</td>
<td></td>
</tr>
<tr>
<td>Seconds as decimal number</td>
<td>ID_FIELDTYPES_SECONDSASDECIMALNUMBER</td>
<td>37767</td>
<td></td>
</tr>
<tr>
<td>Seconds as number</td>
<td>ID_FIELDTYPES_SECONDSASNUMBER</td>
<td>37768</td>
<td></td>
</tr>
<tr>
<td>Timezone hours</td>
<td>ID_FIELDTYPES_TIMEZONEHOURS</td>
<td>37769</td>
<td></td>
</tr>
<tr>
<td>Timezone minutes</td>
<td>ID_FIELDTYPES_TIMEZONEMINUTES</td>
<td>37770</td>
<td></td>
</tr>
<tr>
<td>Week as number</td>
<td>ID_FIELDTYPES_WEEKASNUMBER</td>
<td>37771</td>
<td></td>
</tr>
<tr>
<td>Weekday as number</td>
<td>ID_FIELDTYPES_WEEKDAYASNUMBER</td>
<td>37772</td>
<td></td>
</tr>
<tr>
<td>Weekday as Text, abbrev.</td>
<td>ID_FIELDTYPES_WEEKDAYASTEXTABBREV</td>
<td>37773</td>
<td></td>
</tr>
<tr>
<td>Weekday as text, long format</td>
<td>ID_FIELDTYPES_WEEKDAYASTEXTLONGFORMAT</td>
<td>37774</td>
<td></td>
</tr>
<tr>
<td>Year as number</td>
<td>ID_FIELDTYPES_YEARASNUMBER</td>
<td>37775</td>
<td></td>
</tr>
<tr>
<td>Year as short number</td>
<td>ID_FIELDTYPES_YEARASSHORTNUMBER</td>
<td>37776</td>
<td></td>
</tr>
<tr>
<td>Open Project</td>
<td>ID_FILE_LOAD_PROJECT</td>
<td>36565</td>
<td></td>
</tr>
<tr>
<td>New...</td>
<td>ID_FILE_NEW</td>
<td>57600</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ID_FILTER</td>
<td>36566</td>
<td></td>
</tr>
</tbody>
</table>
Set base year (needs input) | ID_FORMATOPTIONS_BASE | 37779
Set decimal point character (needs input) | ID_FORMATOPTIONS_DEC | 37780
(Empty option) | ID_FORMATOPTIONS_EMPTY | 37781
Set fill character (needs input) | ID_FORMATOPTIONS_FILL | 37782
Hours from 1-12 | ID_FORMATOPTIONS_H12 | 37783
Hours from 0-23 | ID_FORMATOPTIONS_H24 | 37784
Initial character only | ID_FORMATOPTIONS_INIT | 37785
Long format | ID_FORMATOPTIONS_LONG | 37786
Lower case | ID_FORMATOPTIONS_LOWERCASE | 37787
Show as positive number | ID_FORMATOPTIONS_POS | 37788
Read-only | ID_FORMATOPTIONS_READONLY | 37789
Set digit separator character (needs input) | ID_FORMATOPTIONS_SEP | 37790
Short format | ID_FORMATOPTIONS_SHORT | 37791
UPPER CASE | ID_FORMATOPTIONS_UPPERCASE | 37792
Adds a configuration | ID_GLOBALRESOURCES_ADDCONFIG | 37429
Adds a configuration as copy of the currently selected configuration | ID_GLOBALRESOURCES_ADDCONFIGCOPY | 37430
Deletes a configuration | ID_GLOBALRESOURCES_DELCONFIG | 37431
Assign XML schema... | ID_ICDBWNS_ASSIGN_SCHEMA | 36656
Show referenced table | ID_ICDBWNS_FGNKEY_GOTO_REFERENCE | 36567
View in XMLSpy | ID_ICDBWNS_VIEW_XMLSCHEMA_IN_XMLSPY | 36500
Add a New Data Source... | ID_ICDBWNS_ADDNEWDATASOURCE | 36568
Add to Design Editor | ID_ICDBWNS_BROWSER_ADD_TO_DESIGNVIEW | 38021
Add to/Remove from Favorites | ID_ICDBWNS_BROWSER_ADD_TO_FAVOURITES | 38018
Refresh | ID_ICDBWNS_BROWSER_REFRESH_ROOT | 36571
| ID_ICDBWNS_BROWSER_SEARCH | 36572
| ID_ICDBWNS_BROWSER_SEARCH_COMBO | 36573
| ID_ICDBWNS_BROWSER_SEARCH_MODE | 36574
All | ID_ICDBWNS_BROWSER_SEARCH_MODE_ALL | 36575
From current DataSource | ID_ICDBWNS_BROWSER_SEARCH_MODE_DATASOURCE | 36576
From focused item | ID_ICDBWNS_BROWSER_SEARCH_MODE_FOCUSED_ITEM | 36577
Show in new Design Editor | ID_ICDBWNS_BROWSER_SHOW_IN_DESIGNVIEW | 38016

contd...
<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_ICDBWND_CHECK</td>
<td>Check</td>
</tr>
<tr>
<td>ID_ICDBWND_CHECK_ALL</td>
<td>Check Children</td>
</tr>
<tr>
<td>ID_ICDBWND_CLEAR_ROWCOUNT</td>
<td>Clear</td>
</tr>
<tr>
<td>ID_ICDBWND_COLLAPSE_CHILDREN</td>
<td>Children</td>
</tr>
<tr>
<td>ID_ICDBWND_COLLAPSE_SIBLING</td>
<td>Siblings</td>
</tr>
<tr>
<td>ID_ICDBWND_EXECUTE</td>
<td>Execute SQL</td>
</tr>
<tr>
<td>ID_ICDBWND_EXPAND_CHILDREN</td>
<td>Children</td>
</tr>
<tr>
<td>ID_ICDBWND_EXPAND_SIBLINGS</td>
<td>Siblings</td>
</tr>
<tr>
<td>ID_ICDBWND_EXPORT</td>
<td>Export database data...</td>
</tr>
<tr>
<td>ID_ICDBWND_EXPORT_PREVIEW</td>
<td>ID_ICDBWND_EXPORT</td>
</tr>
<tr>
<td>ID_ICDBWND_FILEDSN_ADD</td>
<td>Add</td>
</tr>
<tr>
<td>ID_ICDBWND_FILEDSN_DELETE</td>
<td>Delete</td>
</tr>
<tr>
<td>ID_ICDBWND_FILTER_CHECKED</td>
<td>Contains</td>
</tr>
<tr>
<td>ID_ICDBWND_FILTER_CONTAINS</td>
<td>Does not contain</td>
</tr>
<tr>
<td>ID_ICDBWND_FILTER_ENDS_WITH</td>
<td>Ends with</td>
</tr>
<tr>
<td>ID_ICDBWND_FILTER_EQUALS</td>
<td>Equals</td>
</tr>
<tr>
<td>ID_ICDBWND_FILTER_FAVORITES</td>
<td>No Filter</td>
</tr>
<tr>
<td>ID_ICDBWND_FILTER_INACTIVE</td>
<td>Starts with</td>
</tr>
<tr>
<td>ID_ICDBWND_FLAG_BLUE</td>
<td>ID_ICDBWND_FLAG_BLUE</td>
</tr>
<tr>
<td>ID_ICDBWND_FLAG_GREEN</td>
<td>ID_ICDBWND_FLAG_GREEN</td>
</tr>
<tr>
<td>ID_ICDBWND_FLAG_MARK</td>
<td>ID_ICDBWND_FLAG_MARK</td>
</tr>
<tr>
<td>ID_ICDBWND_FLAG_ORANGE</td>
<td>ID_ICDBWND_FLAG_ORANGE</td>
</tr>
<tr>
<td>ID_ICDBWND_FLAG_PURPLE</td>
<td>ID_ICDBWND_FLAG_PURPLE</td>
</tr>
<tr>
<td>ID_ICDBWND_FLAG_RED</td>
<td>ID_ICDBWND_FLAG_RED</td>
</tr>
<tr>
<td>ID_ICDBWND_FLAG_YELLOW</td>
<td>ID_ICDBWND_FLAG_YELLOW</td>
</tr>
<tr>
<td>ID_ICDBWND_MENU_DATASOURCE_CONNECT</td>
<td>Connect</td>
</tr>
<tr>
<td>ID_ICDBWND_MENU_DATASOURCE_DISCONNECT</td>
<td>Disconnect</td>
</tr>
<tr>
<td>ID_ICDBWND_MENU_DATASOURCE_GETTABLES</td>
<td>Get Tables</td>
</tr>
<tr>
<td>ID_ICDBWND_UNCHECK</td>
<td>Uncheck</td>
</tr>
</tbody>
</table>

**contd...**
<table>
<thead>
<tr>
<th>Command</th>
<th>ID</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncheck Children</td>
<td>ID_ICDBWND_UNCHECK_ALL</td>
<td>36612</td>
<td></td>
</tr>
<tr>
<td>Show/Update</td>
<td>ID_ICDBWND_UPDATE_ROWCOUNT</td>
<td>36537</td>
<td></td>
</tr>
<tr>
<td>Flat</td>
<td>ID_LAYOUT_FLAT</td>
<td>36615</td>
<td></td>
</tr>
<tr>
<td>Folders</td>
<td>ID_LAYOUT_FOLDERS</td>
<td>36616</td>
<td></td>
</tr>
<tr>
<td>No Schemas</td>
<td>ID_LAYOUT_FOLDERS_NOSCHEMAS</td>
<td>36617</td>
<td></td>
</tr>
<tr>
<td>No Folders</td>
<td>ID_LAYOUT_NOFOLDERS</td>
<td>36618</td>
<td></td>
</tr>
<tr>
<td>Open</td>
<td>ID_OPEN</td>
<td>36619</td>
<td></td>
</tr>
<tr>
<td>Manage XML Schemas...</td>
<td>ID_OPEN_MANAGE_XMLSCHEMA_DIALOG</td>
<td>36658</td>
<td></td>
</tr>
<tr>
<td>New Project</td>
<td>ID_PROJECT_NEWPROJECT</td>
<td>36620</td>
<td></td>
</tr>
<tr>
<td>Remove all Data Sources</td>
<td>IDREMOVEALLDATASOURCES</td>
<td>36621</td>
<td></td>
</tr>
<tr>
<td>Remove</td>
<td>ID_REMOVE_DATASOURCE</td>
<td>36622</td>
<td></td>
</tr>
<tr>
<td>Remove from Favorites</td>
<td>ID_REMOVE_FAVORITE_ITEM</td>
<td>36623</td>
<td></td>
</tr>
<tr>
<td>Remove</td>
<td>ID_REMOVE_FROM_PROJECT</td>
<td>36624</td>
<td></td>
</tr>
<tr>
<td>All rows</td>
<td>ID_RETRIEVE_ALLROWS</td>
<td>36625</td>
<td></td>
</tr>
<tr>
<td>First n rows</td>
<td>ID_RETRIEVE_FIRSTROWS</td>
<td>36626</td>
<td></td>
</tr>
<tr>
<td>Save Project</td>
<td>ID_SAVEPROJECT</td>
<td>36627</td>
<td></td>
</tr>
<tr>
<td>Save Project As...</td>
<td>ID_SAVEPROJECTAS</td>
<td>36628</td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>ID_SHOW_SQL_ADD</td>
<td>36629</td>
<td></td>
</tr>
<tr>
<td>Alter</td>
<td>ID_SHOW_SQL_ALTER</td>
<td>36630</td>
<td></td>
</tr>
<tr>
<td>Create</td>
<td>ID_SHOW_SQL_CREATE</td>
<td>36631</td>
<td></td>
</tr>
<tr>
<td>contd...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete data</td>
<td>ID_SHOW_SQL_DELETE</td>
<td>36632</td>
<td></td>
</tr>
<tr>
<td>Drop</td>
<td>ID_SHOW_SQL_DROP</td>
<td>36633</td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td>ID_SHOW_SQL_EXECUTE</td>
<td>36634</td>
<td></td>
</tr>
<tr>
<td>Insert</td>
<td>ID_SHOW_SQL_INSERT</td>
<td>36635</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>ID_SHOW_SQL_NAME</td>
<td>36636</td>
<td></td>
</tr>
<tr>
<td>Path</td>
<td>ID_SHOW_SQL_PATH</td>
<td>36637</td>
<td></td>
</tr>
<tr>
<td>Rename</td>
<td>ID_SHOW_SQL_RENAME</td>
<td>36638</td>
<td></td>
</tr>
<tr>
<td>Select</td>
<td>ID_SHOW_SQL_SELECT</td>
<td>36639</td>
<td></td>
</tr>
<tr>
<td>Update</td>
<td>ID_SHOW_SQL_UPDATE</td>
<td>36640</td>
<td></td>
</tr>
<tr>
<td>Sort into User and System Tables</td>
<td>ID_SORT_BY_TABLE_TYPE</td>
<td>36640</td>
<td></td>
</tr>
<tr>
<td>Dummy entry</td>
<td>ID_STYLES_ADD_SELECTOR_CLASS_RANGE_FIRST</td>
<td>19661</td>
<td></td>
</tr>
<tr>
<td>Dummy entry</td>
<td>ID_STYLES_ADD_SELECTOR_HTML_RANGE_FIRST</td>
<td>21031</td>
<td></td>
</tr>
<tr>
<td>Dummy entry</td>
<td>ID_STYLES_ADD_SELECTOR_ID_RANGE_FIRST</td>
<td>19681</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ID_STYLEVISION</td>
<td>36039</td>
<td></td>
</tr>
<tr>
<td>Office 2000</td>
<td>ID_VIEW_APPLOOK_2000</td>
<td>37797</td>
<td></td>
</tr>
<tr>
<td>Office 2003</td>
<td>ID_VIEW_APPLOOK_2003</td>
<td>37798</td>
<td></td>
</tr>
<tr>
<td>Visual Studio.NET 2005</td>
<td>ID_VIEW_APPLOOK_VS2005</td>
<td>37799</td>
<td></td>
</tr>
<tr>
<td>Windows XP</td>
<td>ID_VIEW_APPLOOK_WIN_XP</td>
<td>37800</td>
<td></td>
</tr>
<tr>
<td>Office XP</td>
<td>ID_VIEW_APPLOOK_XP</td>
<td>37801</td>
<td></td>
</tr>
<tr>
<td>&lt;User Toolbar&gt;</td>
<td>ID_VIEW_USER_TOOLBAR1</td>
<td>37808</td>
<td></td>
</tr>
<tr>
<td>&lt;User Toolbar&gt;</td>
<td>ID_VIEW_USER_TOOLBAR10</td>
<td>37817</td>
<td></td>
</tr>
<tr>
<td>&lt;User Toolbar&gt;</td>
<td>ID_VIEW_USER_TOOLBAR2</td>
<td>37809</td>
<td></td>
</tr>
<tr>
<td>&lt;User Toolbar&gt;</td>
<td>ID_VIEW_USER_TOOLBAR3</td>
<td>37810</td>
<td></td>
</tr>
<tr>
<td>&lt;User Toolbar&gt;</td>
<td>ID_VIEW_USER_TOOLBAR4</td>
<td>37811</td>
<td></td>
</tr>
<tr>
<td>&lt;User Toolbar&gt;</td>
<td>ID_VIEW_USER_TOOLBAR5</td>
<td>37812</td>
<td></td>
</tr>
<tr>
<td>&lt;User Toolbar&gt;</td>
<td>ID_VIEW_USER_TOOLBAR6</td>
<td>37813</td>
<td></td>
</tr>
<tr>
<td>&lt;User Toolbar&gt;</td>
<td>ID_VIEW_USER_TOOLBAR7</td>
<td>37814</td>
<td></td>
</tr>
<tr>
<td>&lt;User Toolbar&gt;</td>
<td>ID_VIEW_USER_TOOLBAR8</td>
<td>37815</td>
<td></td>
</tr>
<tr>
<td>&lt;User Toolbar&gt;</td>
<td>ID_VIEW_USER_TOOLBAR9</td>
<td>37816</td>
<td></td>
</tr>
</tbody>
</table>
Accessing StyleVisionAPI

The focus of this documentation is the ActiveX controls and interfaces required to integrate the StyleVision user interface into your application. To allow you to automate or control the functionality of the integrated components, the following properties give you access to the StyleVision automation interface (StyleVisionAPI):

- `StyleVisionControl.Application`
- `StyleVisionControlDocument.Document`
- `StyleVisionControlPlaceholder.Project`

Some restrictions apply to the usage of the StyleVision automation interface when integrating StyleVisionControl at document-level. See [Integration at document level](#) for details.
Object Reference

Objects:
- StyleVisionCommand
- StyleVisionCommands
- StyleVisionControl
- StyleVisionControlDocument
- StyleVisionControlPlaceHolder

To give access to standard StyleVision functionality, objects of the StyleVision automation interface can be accessed as well. See StyleVisionControl.Application, StyleVisionControlDocument.Document and StyleVisionControlPlaceHolder.Project for more information.
**StyleVisionCommand**

**Properties:**
- **ID**
- **Label**
- **IsSeparator**
- **ToolTip**
- **StatusText**
- **Accelerator**
- **SubCommands**

**Description:**
Each Command object can be one of three possible types:

- **Command:** ID is set to a value greater than 0 and Label is set to the command name. IsSeparator is false and the SubCommands collection is empty.
- **Separator:** IsSeparator is true. ID is 0 and Label is not set. The SubCommands collection is empty.
- **(Sub) Menu:** The SubCommands collection contains Command objects and Label is the name of the menu. ID is set to 0 and IsSeparator is false.

**Accelerator**

**Property:** Label as string

**Description:**
For command objects that are children of the ALL_COMMANDS collection, this is 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:**
ID is 0 for separators and menus.
For commands, this is the ID which can be used with **Exec** and **QueryStatus**.

**IsSeparator**

**Property:** IsSeparator as boolean

**Description:**
True if the command is a separator.
Label

**Property:** Label as string

**Description:**
Label is empty for separators.
For command objects that are children of the ALL_COMMANDS collection, this is a unique name.
Command icons are stored in files with this name. See Query Commands for more information.

For command objects that are children of menus, the label property holds the command's menu text.
For sub-menus, this property holds the menu text.

StatusText

**Property:** Label as string

**Description:**
For command objects that are children of the ALL_COMMANDS collection, this is the text shown in the status bar when the command is selected.

SubCommands

**Property:** SubCommands as Commands

**Description:**
The SubCommands collection holds any sub-commands if this command is actually a menu or submenu.

ToolTip

**Property:** ToolTip as string

**Description:**
For command objects that are children of the ALL_COMMANDS collection, this is the text shown as tool-tip.
StyleVisionCommands

Properties:

- Count
- Item

Description:
Collection of Command objects to get access to command labels and IDs of the StyleVisionControl. 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.
StyleVisionControl

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 StyleVision library is used in the Application Level mode.

Properties

The following properties are defined:
IntegrationLevel
EnableUserPrompts
Appearance
BorderStyle

Command related properties:
CommandsList
MainMenu
Toolbars
CommandsStructure (deprecated)

Access to StyleVisionAPI:
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 StyleVision 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 StyleVisionControl.

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 ICActiveXIntegrationLevel

Dispatch Id: 1000

Description:
The IntegrationLevel property determines the operation mode of the control. See also Integration at the application level and Integration at document level for more information.

Note: It is important to set this property immediately after the creation of the StyleVisionControl object.
MainMenu

Property: MainMenu = \c \C (read-only)

Dispatch Id: 1003

Description:
This property gives access to the description of the StyleVisionControl main menu.

Toolbars

Property: Toolbars as \c \C (read-only)

Dispatch Id: 1005

Description:
This property returns a list of all toolbar descriptions that describe all toolbars available with StyleVisionControl.

Methods

The following methods are defined:

Open
Exec
QueryStatus

Exec

Method: Exec (nCmdID as long) as boolean

Dispatch Id: 6

Description:
Exec calls the StyleVision command with the ID nCmdID. If the command can be executed, the method returns true. See also CommandsStructure to get a list of all available commands and QueryStatus to retrieve the status of any command.

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 StyleVisionControlDocument.Open and StyleVisionControlPlaceHolder.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 StyleVision command. If QueryStatus returns a value of 1 or 5, the command is disabled.

**Events**

The StyleVisionControl ActiveX control provides the following connection point events:

- OnUpdateCmdUI
- OnOpenedOrFocused
- OnCloseEditingWindow
- OnFileChangedAlert
- OnContextChanged
- OnDocumentOpened
- OnValidationWindowUpdated

**OnCloseEditingWindow**

**Event:** OnCloseEditingWindow (i_strFilePath as String) as boolean

**Dispatch Id:** 1002

**Description:**
This event is triggered when StyleVision 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 StyleVisionControl 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 not used in StyleVision
OnDocumentOpened

*Event:* OnDocumentOpened (objDocument as Document)

*Dispatch Id:* 1

*Description:*
This event is triggered whenever a document is opened. The argument objDocument is a Document object from the StyleVision 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 `StyleVisionControlDocument.OnDocumentOpened` instead.

OnFileChangedAlert

*Event:* OnFileChangedAlert (i_strFilePath as String) as bool

*Dispatch Id:* 1001

*Description:*
This event is triggered when a file loaded with StyleVisionControl, is changed on the harddisk by another application. Clients should return true, if they handled the event, or false, if StyleVision 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 StyleVisionControl 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 StyleVision.

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 StyleVisionControl, 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 StyleVision commands using `StyleVisionControl.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.
StyleVisionControlDocument

Properties:
Appearance
BorderStyle
Document
IsModified
Path
ReadOnly

Methods:
Exec
New
Open
QueryStatus
Reload
Save
SaveAs

Events:
OnDocumentOpened
OnDocumentClosed
OnModifiedFlagChanged
OnContextChanged
OnFileChangedAlert
OnActivate

If the StyleVisionControl is integrated in the Document Level mode each document is displayed in an own object of type StyleVisionControlDocument. The StyleVisionControlDocument 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.

Properties

The following properties are defined:
ReadOnly
IsModified
Path
Appearance
BorderStyle

Access to StyleVisionAPI:
Document
Appearance

Property: Appearance as short

Dispatch Id: -520

Description:
A value not equal to 0 displays a client edge around the document control. Default value is 0.
BorderStyle

**Property:** BorderStyle as short

**Dispatch Id:** -504

**Description:**
A value of 1 displays the control with a thin border. Default value is 0.

Document

**Property:** Document as Document

**Dispatch Id:** 1

**Description:**
The Document property gives access to the Document object of the StyleVision automation server API. This interface provides additional functionalities 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 StyleVision command with the ID nCmdID. If the command can be executed, the method returns true. The client should call the Exec method of the document control if there is currently an active document available in the application.

See also CommandsStructure to get a list of all available commands and QueryStatus to retrieve the status of any command.

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 Value</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
</table>
0 1  Supported  Set if the command is supported.
1 2  Enabled   Set if the command is enabled (can be executed).
2 4  Checked  Set if the command is checked.

This means that if QueryStatus returns 0 the command ID is not recognized as a valid StyleVision 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 StyleVisionControlDocument ActiveX control provides following connection point events:

- OnDocumentOpened
- OnDocumentClosed
- OnModifiedFlagChanged
- OnContextChanged
- OnFileChangedAlert
- OnActivate
- OnSetEditorTitle

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: None

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 StyleVision 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 StyleVision 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 harddisk by another application. Clients should return true, if they handled the event, or false, if StyleVision 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.
StyleVisionControlPlaceHolder

Properties available for all kinds of placeholder windows:

**PlaceholderWindowID**

Properties for project placeholder window:

**Project**

Methods for project placeholder window:

*OpenProject*

*CloseProject*

The `StyleVisionControlPlaceHolder` control is used to show the additional StyleVision windows like Overview, Library or Project window. It is used like any other ActiveX control and can be placed anywhere in the client application.

Properties

The following properties are defined:

**PlaceholderWindowID**

Access to StyleVisionAPI:

**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

**Dispatch Id:** 1

**Description:**
Using this property the object knows which StyleVision 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 enumeration. The control changes its state immediately and shows the new StyleVision window.

**Project**

**Property:** Project as Project (read-only)

**Dispatch Id:** 2

**Description:**
The Project property gives access to the Project object of the StyleVision automation server API. This interface provides additional functionalities 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 \texttt{PlaceholderWindowID} with a value of \texttt{StyleVisionXProjectWindow (=3)}. The property is read-only.

**Methods**

The following method is defined:

- \texttt{OpenProject}
- \texttt{CloseProject}

\texttt{OpenProject}

**Method:** \texttt{OpenProject (strFileName as string) as boolean}

**Dispatch Id:** 3

**Description:**

\texttt{OpenProject} loads the file \texttt{strFileName} as the new project into the control. The method will fail if the placeholder window has a \texttt{PlaceholderWindowID} different to \texttt{XMLSpyXProjectWindow (=3)}.

\texttt{CloseProject}

**Method:** \texttt{CloseProject ()}

**Dispatch Id:** 4

**Description:**

\texttt{CloseProject} closes the project loaded the control. The method will fail if the placeholder window has a \texttt{PlaceholderWindowID} different to \texttt{StyleVisionXProjectWindow (=3)}.

**Events**

The \texttt{StyleVisionControlPlaceholder} ActiveX control provides following connection point events:

- \texttt{OnModifiedFlagChanged}

\texttt{OnModifiedFlagChanged}

**Event:** \texttt{OnModifiedFlagChanged (i\_bIsModified as boolean)}

**Dispatch Id:** 1

**Description:**

This event gets triggered only for placeholder controls with a \texttt{PlaceholderWindowID} of \texttt{StyleVisionXProjectWindow (=3)}. The event is fired whenever the project content changes between modified and unmodified state. The parameter \texttt{i\_bIsModified} is \texttt{true} if the project contents differs from the original content, and \texttt{false}, otherwise.
OnSetLabel

*Event:* OnSetLabel(i_strNewLabel as string)

*Dispatch Id:* 1000

*Description:* Raised when the title of the placeholder window is changed.
Enumerations

The following enumerations are defined:

ICActiveXIntegrationLevel
StyleVisionControlPlaceholderWindow

ICActiveXIntegrationLevel

Possible values for the IntegrationLevel property of the StyleVisionControl.

ICActiveXIntegrationOnApplicationLevel    = 0
ICActiveXIntegrationOnDocumentLevel       = 1

StyleVisionControlPlaceholderWindow

This enumeration contains the list of the supported additional StyleVision windows.

StyleVisionControlNoToolWnd              = -1
StyleVisionControlProjectWnd             =  0
StyleVisionControlDesignOverviewWnd      =  1
StyleVisionControlSchemaSourcesWnd       =  2
StyleVisionControlDesignTreeWnd          =  3
StyleVisionControlStyleRepositoryWnd     =  4
StyleVisionControlContextPropertiesWnd   =  5
StyleVisionControlContextStylesWnd       =  6
StyleVisionControlMessageWnd             =  7
Chapter 22

Appendices
22 Appendices

These appendices contain (i) information about the XSLT Engines used in StyleVision; (ii) information about the conversion of DB datatypes to XML Schema datatypes; (iii) technical information about StyleVision; and (iv) licensing information for StyleVision. Each appendix contains the sub-sections listed below:

XSLT Engine Information
Provides implementation-specific information about the Altova XSLT Engines, which are used by StyleVision to generate output.

- Altova XSLT 1.0 Engine
- Altova XSLT 2.0 Engine
- Altova XSLT 3.0 Engine
- XSLT and XPath/XQuery Functions

DatatypesDB2XSD
When DB fields are converted to XML nodes, the DB datatypes are converted to XML Schema datatypes. This appendix lists the mappings for the following source DBs.

- MS Access
- MS SQL Server
- MySQL
- Oracle
- ODBC
- ADO
- Sybase

Technical Data
Provides technical information about StyleVision.

- OS and memory requirements
- Altova XML Parser
- Altova XSLT and XQuery Engines
- Unicode support
- Internet usage

License Information
Contains information about the way StyleVision is distributed and about its licensing.

- Electronic software distribution
- License metering
- Copyright
- End User License Agreement
22.1 XSLT and XQuery Engine Information

The XSLT and XQuery engines of StyleVision 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 StyleVision.

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
**XSLT 1.0**

The XSLT 1.0 Engine of StyleVision 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 `&nbsp;` (the decimal character reference for a non-breaking space) is inserted as `&nbsp;` in the HTML code.
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:

```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:stylesheet>
```

The following points should be noted:
The XSLT 2.0 engine uses the XPath 2.0 and XQuery 1.0 Functions namespace (listed in the table above) as its **default functions namespace**. So you can use XPath 2.0 and XSLT 2.0 functions in your stylesheet without any prefix. If you declare the XPath 2.0 Functions namespace in your stylesheet with a prefix, then you can additionally use the prefix assigned in the declaration.

- When using type constructors and types from the XML Schema namespace, the prefix used in the namespace declaration must be used when calling the type constructor (for example, `xs:date`).
- Some XPath 2.0 functions have the same name as XML Schema datatypes. For example, for the XPath functions `fn:string` and `fn:boolean` there exist XML Schema datatypes with the same local names: `xs:string` and `xs:boolean`. So if you were to use the XPath expression `string('Hello')`, the expression evaluates as `fn:string('Hello')`—not as `xs:string('Hello')`.

### Schema-awareness
The XSLT 2.0 engine is schema-aware. So you can use user-defined schema types and the `xsl:validate` instruction.

### Implementation-specific behavior
Given below is a description of how the XSLT 2.0 engine handles implementation-specific aspects of the behavior of certain XSLT 2.0 functions.

- **xsl:result-document**
  Additionally supported encodings are (the Altova-specific): `x-base16tobinary` and `x-base64tobinary`.

- **function-available**
  The function tests for the availability of in-scope functions (XSLT 2.0, XPath 2.0, 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`.
**XSLT 3.0**

The XSLT 3.0 Engine of StyleVision conforms to the World Wide Web Consortium's (W3C's) [XSLT 3.0 Last Call Working Draft of 2 October 2014](https://www.w3.org/TR/xslt30-lcd/) and [XPath 3.1 Candidate Recommendation of 18 December 2014](https://www.w3.org/TR/xpath31-cd/).

The XSLT 3.0 engine has the same implementation-specific characteristics as the XSLT 2.0 engine. Additionally, it includes support for the following XSLT 3.0 features: `xsl:evaluate`, `xsl:try`, `xsl:catch`, `xsl:map`, `xsl:map-entry`, text value templates, XPath/XQuery 3.1 functions and operators, and the [XPath 3.1 specification](https://www.w3.org/TR/xpath31-cd/).

The following XSLT 3.0 instructions are currently unsupported:

- `xsl:accept`
- `xsl:accumulator`
- `xsl:accumulator-rule`
- `xsl:assert`
- `xsl:break`
- `xsl:context-item`
- `xsl:expose`
- `xsl:fork`
- `xsl:iterate`
- `xsl:merge`
- `xsl:merge-action`
- `xsl:merge-key`
- `xsl:merge-source`
- `xsl:mode`
- `xsl:next-iteration`
- `xsl:next-match`
- `xsl:on-completion`
- `xsl:override`
- `xsl:package`
- `xsl:stream`
- `xsl:use-package`
XQuery 1.0

This section:

- Engine conformance
- Schema awareness
- Encoding
- Namespaces
- XML source and validation
- Static and dynamic type checking
- Library modules
- External modules
- Collations
- Precision of numeric data
- XQuery instructions support

---

Conformance

The XQuery 1.0 Engine of StyleVision 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:

```xml
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:

```xml
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:

```xml
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.
XQuery 3.1

The XQuery 3.1 Engine of StyleVision conforms to the World Wide Web Consortium's (W3C's) XQuery 3.1 Candidate Recommendation of 18 December 2014 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.
22.2 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. Other supported collations are the ICU collations listed below. To use a specific collation, supply its URI as given in the list of supported collations (table below). Any string comparisons, including for the max and min functions, will be made according to the specified collation. If the collation option is not specified, the default Unicode-codepoint collation is used.

<table>
<thead>
<tr>
<th>Language</th>
<th>URIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>da: Danish</td>
<td>da_DK</td>
</tr>
<tr>
<td>de: German</td>
<td>de_AT, de_BE, de_CH, de_DE, de_LI, de_LU</td>
</tr>
<tr>
<td>en: English</td>
<td>en_AS, en_AU, en_BB, en_BE, en_BM, en_BW, en_BZ, en_CA</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.

<table>
<thead>
<tr>
<th>Language</th>
<th>Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>en_GB, en_GU, en_HK, en_IE, en_JM, en_MH, en_MP, en_MT, en_MU, en_NA, en_NZ, en_PH, en_PK, en_SG, en_TT, en_Um, en_US, en_VI, en_ZA, en_ZW</td>
</tr>
<tr>
<td>Spanish</td>
<td>es_419, es_AR, es_BO, es_CL, es_CO, es_CR, es_DO, es_EC, es_ES, es_GQ, es_GT, es_HN, es_MX, es_NI, es_PA, es_PE, es_PR, es_PY, es_SV, es_US, es_UY, es_VE</td>
</tr>
<tr>
<td>Italian</td>
<td>it_CH, it_IT</td>
</tr>
<tr>
<td>Japanese</td>
<td>ja_JP</td>
</tr>
<tr>
<td>Norwegian Bokmal</td>
<td>nb_NO</td>
</tr>
<tr>
<td>Dutch</td>
<td>nl_AW, nl_BE, nl_NL</td>
</tr>
<tr>
<td>Nynorsk</td>
<td>nn_NO</td>
</tr>
<tr>
<td>Portuguese</td>
<td>pt_AO, pt_BR, pt_GW, pt_MZ, pt_PT, pt_ST</td>
</tr>
<tr>
<td>Russian</td>
<td>ru_MD, ru_RU, ru_UA</td>
</tr>
<tr>
<td>Swedish</td>
<td>sv_FI, sv_SE</td>
</tr>
</tbody>
</table>
Altova Extension Functions

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the Altova extension functions namespace, http://www.altova.com/xslt-extensions, and are indicated in this section with the prefix altova:, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

Functions defined in the W3C's XPath/XQuery Functions specifications can be used in: (i) XPath expressions in an XSLT context, and (ii) in XQuery expressions in an XQuery document. In this documentation we indicate the functions that can be used in the former context (XPath in XSLT) with an xP symbol and call them XPath functions; those functions that can be used in the latter (XQuery) context are indicated with an xQ symbol; they work as XQuery functions. The W3C's XSLT specifications—not XPath/XQuery Functions specifications—also define functions that can be used in XPath expressions in XSLT documents. These functions are marked with an xSLT symbol and are called XSLT functions. The XPath/XQuery and XSLT versions in which a function can be used are indicated in the description of the function (see symbols below). Functions from the XPath/XQuery and XSLT function libraries are listed without a prefix. Extension functions from other libraries, such as Altova extension functions, are listed with a prefix.

<table>
<thead>
<tr>
<th>XPath functions (used in XPath expressions in XSLT):</th>
<th>XP1</th>
<th>XP2</th>
<th>XP3</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLT functions (used in XPath expressions in XSLT):</td>
<td>XSLT1</td>
<td>XSLT2</td>
<td>XSLT3</td>
</tr>
<tr>
<td>XQuery functions (used in XQuery expressions in XQuery):</td>
<td>XQ1</td>
<td>XQ3</td>
<td></td>
</tr>
</tbody>
</table>

**XSLT functions**

XSLT functions can only be used in XPath expressions in an XSLT context (similarly to XSLT 2.0's current-group() or key() functions). These functions are not intended for, and will not work in, a non-XSLT context (for instance, in an XQuery context). Note that XSLT functions for XBRL can be used only with editions of Altova products that have XBRL support.

**XPath/XQuery functions**

XPath/XQuery functions can be used both in XPath expressions in XSLT contexts as well as in XQuery expressions:

- Date/Time
- Geolocation
- Image-related
- Numeric
- Sequence
- String
- Miscellaneous
**Chart functions (Enterprise and Server Editions only)**
Altova extension functions for charts are supported only in the Enterprise and Server Editions of Altova products and enable charts to be generated from XML data.

**Barcode functions**
Altova's barcode extension functions enable barcodes to be generated and placed in output generated via XSLT stylesheets.
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](http://www.altova.com/xslt-extensions), and are indicated in this section with the prefix `altova:` which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

| XPath functions (used in XPath expressions in XSLT): | XP1 XP2 XP3 |
| XSLT functions (used in XPath expressions in XSLT): | XSLT1 XSLT2 XSLT3 |
| XQuery functions (used in XQuery expressions in XQuery): | XQ1 XQ3 |

**Standard functions**

**distinct-nodes [altova:]**

`altova:distinct-nodes(node()) as node()*`  
XSLT1 XSLT2 XSLT3

Takes a set of one or more nodes as its input and returns the same set minus nodes with duplicate values. The comparison is done using the XPath/XQuery function `fn:deep-equal`.

**Examples**

- `altova:distinct-nodes(country)` returns all child country nodes less those having duplicate values.

**evaluate [altova:]**

`altova:evaluate(XPathExpression as xs:string[, ValueOf$p1, ... ValueOf$pN])`  
XSLT1 XSLT2 XSLT3

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: `pl` to `pN`: The second argument will be the value of the variable `pl`, the third argument that of the variable `p2`, and so on; (iii) The variable values must be of type `item*`. 
Example

```xml
<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
<xsl:value-of select="altova:evaluate($xpath, 10, 20, 'hi')" />
```

outputs "hi 20 10"

In the listing above, notice the following:

- The second argument of the `altova:evaluate` expression is the value assigned to the variable `$p1`, the third argument that assigned to the variable `$p2`, and so on.
- Notice that the fourth argument of the function is a string value, indicated by its being enclosed in quotes.
- The `select` attribute of the `xs:variable` element supplies the XPath expression. Since this expression must be of type `xs:string`, it is enclosed in single quotes.

Examples to further illustrate the use of variables

- `<xsl:variable name="xpath" select="'$p1'" />
  <xsl:value-of select="altova:evaluate($xpath, //Name[1])" />

  Outputs value of the first Name element.

- `<xsl:variable name="xpath" select="'$p1'" />
  <xsl:value-of select="altova:evaluate($xpath, '//Name[1]')" />

  Outputs "//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:
  
  ```
  <xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
  <xsl:value-of select="altova:evaluate($xpath, 10, 20, 30)" />
  Outputs "30 20 10"
  ```

• Dynamic XPath expression with no dynamic variable:
  
  ```
  <xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
  <xsl:value-of select="altova:evaluate($xpath)" />
  Outputs error: No variable defined for $p3.
  ```

### encode-for-rtf [altova:]

| altova:encode-for-rtf(input as xs:string, preserveallwhitespace as xs:boolean, preservenewlines as xs:boolean) as xs:string |
| XSLT2 XSLT3 |

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:]

| altova:xbrl-footnotes(node()) as node(*) |
| XSLT2 XSLT3 |

Takes a node as its input argument and returns the set of XBRL footnote nodes referenced by the input node.

#### xbrl-labels [altova:]

| altova:xbrl-labels(xs:QName, xs:string) as node()* |
| XSLT2 XSLT3 |

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.
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 X Path, X Query, 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.

### Grouped by functionality

- Add 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 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:age
- altova:age-details
- altova:build-date
- altova:build-duration
- altova:build-time
- 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
Add a duration to xs:dateTime

These functions add a duration to xs:dateTime and return xs:dateTime. The xs:dateTime type has a format of CCYY-MM-DDThh:mm:ss.sss. This is a concatenation of the xs:date and xs:time formats separated by the letter T. A timezone suffix +01:00 (for example) is optional.

#### add-years-to-dateTime [altova:]

altova:add-years-to-dateTime(DateTime as xs:dateTime, Years as xs:integer) as xs:dateTime

Adds a duration in years to an xs:dateTime (see examples below). The second argument is the number of years to be added to the xs:dateTime supplied as the first argument. The result is of type xs:dateTime.

**Examples**

- altova:add-years-to-dateTime(xs:dateTime("2014-01-15T14:00:00"), 10)
  returns 2024-01-15T14:00:00
- altova:add-years-to-dateTime(xs:dateTime("2014-01-15T14:00:00"), -4)
  returns 2010-01-15T14:00:00

#### add-months-to-dateTime [altova:]

altova:add-months-to-dateTime(DateTime as xs:dateTime, Months as xs:integer) as xs:dateTime

Adds a duration in months to an xs:dateTime (see examples below). The second argument is the number of months to be added to the xs:dateTime supplied as the first argument. The result is of type xs:dateTime.

**Examples**

- altova:add-months-to-dateTime(xs:dateTime("2014-01-15T14:00:00"), 10)
  returns 2014-11-15T14:00:00
- altova:add-months-to-dateTime(xs:dateTime("2014-01-15T14:00:00"), -2)
  returns 2013-11-15T14:00:00

#### add-days-to-dateTime [altova:]

altova:add-days-to-dateTime(DateTime as xs:dateTime, Days as xs:integer) as xs:dateTime
**xs:dateTime**  

Adds a duration in days to an `xs:dateTime` *(see examples below)*. The second argument is the number of days to be added to the `xs:dateTime` supplied as the first argument. The result is of type `xs:dateTime`.

**Examples**

- `altova:add-days-to-dateTime(xs:dateTime("2014-01-15T14:00:00"), 10)` returns `2014-01-25T14:00:00`
- `altova:add-days-to-dateTime(xs:dateTime("2014-01-15T14:00:00"), -8)` returns `2014-01-07T14:00:00`

**add-hours-to-dateTime**  

`altova: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: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:add-seconds-to-dateTime(DateTime as xs:dateTime, Seconds as xs:integer)` as `xs:dateTime`  

Adds a duration in seconds to an `xs:dateTime` *(see examples below)*. The second argument is the number of seconds to be added to the `xs:dateTime` supplied as the first argument. The result is of type `xs:dateTime`.

**Examples**

- `altova:add-seconds-to-dateTime(xs:dateTime("2014-01-15T14:00:10"), 20)` returns `2014-01-15T14:00:30`
- `altova:add-seconds-to-dateTime(xs:dateTime("2014-01-15T14:00:10"), -5)
returns 2014-01-15T14:00:05

---

### Add a duration to xs:date

These functions add a duration to `xs: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
```

Adds a duration in years to a date. The second argument is the number of years to be added to the `xs:date` supplied as the first argument. The result is of type `xs:date`.

**Examples**

- `altova:add-years-to-date(xs:date("2014-01-15"), 10)` returns `2024-01-15`
- `altova:add-years-to-date(xs:date("2014-01-15"), -4)` returns `2010-01-15`

**add-months-to-date** [altova:]

```
altova:add-months-to-date(Date as xs:date, Months as xs:integer) as xs:date
```

Adds a duration in months to a date. The second argument is the number of months to be added to the `xs:date` supplied as the first argument. The result is of type `xs:date`.

**Examples**


**add-days-to-date** [altova:]

```
altova:add-days-to-date(Date as xs:date, Days as xs:integer) as xs:date
```

Adds a duration in days to a date. The second argument is the number of days to be added to the `xs:date` supplied as the first argument. The result is of type `xs:date`.

**Examples**


---

### Format and retrieve durations

These functions add a duration to `xs:date` and return `xs:date`. The `xs:date` type has a format of `CCYY-MM-DD`. 
Appendices SLT and XPath/XQuery Functions

**format-duration [altova:]**

```
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`.

**Examples**
- `altova:add-minutes-to-time(xs:time("14:10:00"), 45)` returns 14:55:00
- `altova:add-minutes-to-time(xs:time("14:10:00"), -5)` returns 14:05:00

### add-seconds-to-time [altova:]

**altova:add-seconds-to-time** *(Time as xs:time, Minutes as xs:integer) as xs:time*

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-dateTime-no-TZ [altova:]

**altova:current-dateTime-no-TZ() as xs:dateTime**

This function takes no argument. It removes the timezone part of `current-dateTime()` (which is the current date-and-time according to the system clock) and returns an `xs:dateTime` value.

**Examples**
- If the current `dateTime` is **2014-01-15T14:00:00+01:00**:
  - `altova:current-dateTime-no-TZ()` returns **2014-01-15T14:00:00**
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-time-no-TZ [altova:]**

altova:current-time-no-TZ() as xs:time XP3 XQ3

This function takes no argument. It removes the timezone part of current-time() (which is the current time according to the system clock) and returns an xs:time value.

**Examples**

If the current time is 14:00:00+01:00:
- altova:current-time-no-TZ() returns 14:00:00

---

**Return the weekday from xs:dateTime or xs:date**

These functions return the weekday (as an integer) from xs:dateTime or xs:date. The days of the week are numbered (using the American format) from 1 to 7, with Sunday=1. In the European format, the week starts with Monday (=1). The American format, where Sunday=1, can be set by using the integer 0 where an integer is accepted to indicate the format.

**weekday-from-dateTime [altova:]**

altova:weekday-from-dateTime(DateTime as xs:dateTime) as xs:integer XP3 XQ3

Takes a date-with-time as its single argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with Sunday=1. If the European format is required (where Monday=1), use the other signature of this function (see next signature below).

**Examples**

- altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00")) returns 2, which would indicate a Monday.

altova:weekday-from-dateTime(DateTime as xs:dateTime, Format as xs:integer) as xs:integer XP3 XQ3

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.

\subsection{weekday-from-date [altova:]}\\
\begin{verbatim}
altova:weekday-from-date(Date as xs:date) as xs:integer  
\end{verbatim}

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).

\paragraph{Examples}

\begin{itemize}
\item altova:weekday-from-date(xs:date("2014-02-03+01:00")) returns 2, which
would indicate a Monday.
\end{itemize}

altova:weekday-from-date(Date as xs:date, Format as xs:integer) as xs:integer

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).

\paragraph{Examples}

\begin{itemize}
\item altova:weekday-from-date(xs:date("2014-02-03"), 1) returns 1, which would
indicate a Monday
\item altova:weekday-from-date(xs:date("2014-02-03"), 4) returns 1, which would
indicate a Monday
\item altova:weekday-from-date(xs:date("2014-02-03"), 0) returns 2, which would
indicate a Monday.
\end{itemize}

\section{Return the week number from xs:dateTime or xs:date}  
These functions return the week number (as an integer) from xs:dateTime or xs:date. Week-
numbering is available in the US, ISO-European, and Islamic calendar formats. Week-numbering
is different in these calendar formats because the week is considered to start on different days (on
Sunday in the US format, Monday in the ISO/European format, and Saturday in the Islamic
format).

\subsection{weeknumber-from-date [altova:]}\\
\begin{verbatim}
altova:weeknumber-from-date(Date as xs:date, Calendar as xs:integer) as xs:integer
\end{verbatim}

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:
\begin{itemize}
\item 0 = US calendar (week starts Sunday)
\item 1 = ISO standard, European calendar (week starts Monday)
\end{itemize}
• 2 = Islamic calendar (*week starts Saturday*)

Default is 0.

**Examples**

```xml
• 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**

```xml
altova:weeknumber-from-datetime(DateTimex 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**

```xml
• 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**

```xml
```
altova:build-date(Year as xs:integer, Month as xs:integer, Date as xs:integer) as xs:date

The first, second, and third arguments are, respectively, the year, month, and date. They are combined to build a value of xs:date type. The values of the integers must be within the correct range of that particular date part. For example, the second argument (for the month part) should not be greater than 12.

Examples

- altova:build-date(2014, 2, 03) returns 2014-02-03

build-time [altova:]

altova:build-time(Hours as xs:integer, Minutes as xs:integer, Seconds as xs:integer) as xs:time

The first, second, and third arguments are, respectively, the hour (0 to 23), minutes (0 to 59), and seconds (0 to 59) values. They are combined to build a value of xs:time type. The values of the integers must be within the correct range of that particular time part. For example, the second (Minutes) argument should not be greater than 59. To add a timezone part to the value, use the other signature of this function (see next signature).

Examples

- altova:build-time(23, 4, 57) returns 23:04:57

altova:build-time(Hours as xs:integer, Minutes as xs:integer, Seconds as xs:integer, TimeZone as xs:string) as xs:time

The first, second, and third arguments are, respectively, the hour (0 to 23), minutes (0 to 59), and seconds (0 to 59) values. The fourth argument is a string that provides the timezone part of the value. The four arguments are combined to build a value of xs:time type. The values of the integers must be within the correct range of that particular time part. For example, the second (Minutes) argument should not be greater than 59.

Examples

- altova:build-time(23, 4, 57, '+1') returns 23:04:57+01:00

build-duration [altova:]

altova:build-duration(Years as xs:integer, Months as xs:integer) as xs:yearMonthDuration

Takes two arguments to build a value of type xs:yearMonthDuration. The first arguments provides the Years part of the duration value, while the second argument provides the Months part. If the second (Months) argument is greater than or equal to 12, then the integer is divided by 12; the quotient is added to the first argument to provide the Years part of the duration value while the remainder (of the division) provides the Months part. To build a duration of type xs:dayTimeDuration, see the next signature.

Examples

- altova:build-duration(2, 10) returns P2Y10M
- altova:build-duration(14, 27) returns P16Y3M
- altova:build-duration(2, 24) returns P4Y

altova:build-duration(Days as xs:integer, Hours as xs:integer, Minutes as
xs:integer, Seconds as xs:integer) as xs:dayTimeDuration

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

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

Returns the input string Date as an xs:date value. The second argument DatePattern specifies the pattern (sequence of components) of the input string. DatePattern is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

D     Date
M     Month
Y     Year

The pattern in DatePattern must match the pattern in Date. Since the output is of type xs:date, the output will always have the lexical format YYYY-MM-DD.

Examples

- `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`
altova:parse-dateTime(Date-Time | as xs:string, Date-Time-Pattern | as xs:string) as xs:dateTime

XSLT and XPath/XQuery Functions 1457

Returns the input string Date-Time as an xs:dateTime value. The second argument
Date-Time-Pattern specifies the pattern (sequence of components) of the input string.
Date-Time-Pattern is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

d  Date

m  minutes

s  seconds

The pattern in Date-Time-Pattern must match the pattern in Date-Time. 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 2014-12-09T13:56:24

parse-time [altova:]

altova:parse-time(Time | as xs:string, Time-Pattern | as xs:string) as xs:time

XSLT and XPath/XQuery Functions 1457

Returns the input string Time as an xs:time value. The second argument Time-Pattern specifies the pattern (sequence of components) of the input string. Time-Pattern 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 Time-Pattern must match the pattern in Time. Since the output is of type xs:time, the output will always have the lexical format HH:mm:ss.

Examples

- `altova:parse-time("13-56-24", "[H]-[m]")` returns 13:56:00
- `altova:parse-time("time=13h56m24s", "time=[H]h[m]m[s]s")` returns
  13:56:24
- `altova:parse-time("time=24s56m13h", "time=[s]s[m]m[H]h")` returns
  13:56:24

[ Top ]
Age-related functions  XP3  XQ3
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.

\[ \text{age} \] [altova:]

\[
\text{altova:age}([\text{StartDate} \ as \ \text{xs:date}] \ as \ \text{xs:integer}) \ as \ \text{xs:integer} \quad \text{XP3} \ XQ3
\]

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:
- \( \text{altova:age}([\text{xs:date}("2013-01-15")]) \) returns 1
- \( \text{altova:age}([\text{xs:date}("2013-01-16")]) \) returns 0
- \( \text{altova:age}([\text{xs:date}("2015-01-15")]) \) returns -1
- \( \text{altova:age}([\text{xs:date}("2015-01-14")]) \) returns 0

\[
\text{altova:age}([\text{StartDate} \ as \ \text{xs:date}, \ \text{EndDate} \ as \ \text{xs:date}] \ as \ \text{xs:integer}) \quad \text{XP3} \ XQ3
\]

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:
- \( \text{altova:age}([\text{xs:date}("2000-01-15"), \ \text{xs:date}("2010-01-15")]) \) returns 10
- \( \text{altova:age}([\text{xs:date}("2000-01-15"), \ \text{current-date()}]) \) returns 14 if the current date is 2014-01-15
- \( \text{altova:age}([\text{xs:date}("2014-01-15"), \ \text{xs:date}("2010-01-15")]) \) returns -4

\[ \text{age-details} \] [altova:]

\[
\text{altova:age-details}([\text{InputDate} \ as \ \text{xs:date}] \ as (\text{xs:integer})* \quad \text{XP3} \ XQ3
\]

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:
- \( \text{altova:age-details}([\text{xs:date}("2014-01-16")]) \) returns (0 0 1)
- \( \text{altova:age-details}([\text{xs:date}("2014-01-14")]) \) returns (0 0 1)
- \( \text{altova:age-details}([\text{xs:date}("2013-01-16")]) \) returns (1 0 1)
• `altova:age-details`\([\text{current-date()}]\) returns \((0 \ 0 \ 0)\)

\[
\text{altova:age-details}(\text{Date-1 as xs:date, Date-2 as xs:date}) \text{ as } (xs:integer)\quad \text{XP3 XQ3}
\]

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 \((0 \ 0 \ 1)\)
XPath/XQuery Functions: Geolocation

The following geolocation XPath/XQuery extension functions are supported in the current version of StyleVision 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.

<table>
<thead>
<tr>
<th>XPath functions (used in XPath expressions in XSLT):</th>
<th>XP1 XP2 XP3</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</td>
</tr>
</tbody>
</table>

```
altova: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(148°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)**
  \[
  D^\circ M'.S'' N/S \quad D^\circ M'.S'' W/E
  \]
  **Example:** \(33^\circ 55'.11''N\) \(22^\circ 44'.66''W\)

- **Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for
  (N/W) is optional**
  \[
  +/-D^\circ M'.S'' +/-D^\circ M'.S''
  \]
  **Example:** \(33^\circ 55'.11''\) \(-22^\circ 44'.66''\)

- **Degrees, decimal minutes, with suffixed orientation (N/S, W/E)**
  \[
  D^\circ M.M'M/" N/S \quad D^\circ M.M'M/" W/E
  \]
  **Example:** \(33^\circ 55.55'N\) \(22^\circ 44.44'W\)

- **Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (N/W) is
  optional**
  \[
  +/-D^\circ M.M'M/" +/-D^\circ M.M'M/"
  \]
  **Example:** \(+33^\circ 55.55'\) \(-22^\circ 44.44'\)

- **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**
  \[
  +/-D.DD +/-D.DD
  \]
  **Example:** \(+33.33\) \(-22.22\)

**Examples of format-combinations:**
- \(33^\circ 55'.11''N\) \(22^\circ 44'.66''W\)
- \(33.33\) \(22^\circ 44'.6666''W\)
- \(33.33\) \(22.c\)

**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</td>
<td>E</td>
<td>(33^\circ 51'.21.91''S) (151^\circ 13'.11.73''E)</td>
</tr>
</tbody>
</table>

**geolocation-distance-km** [altova:]
altova:geolocation-distance-km(GeolocationInputString-1 as xs:string, GeolocationInputString-2 as xs:string) as xs:decimal

Calculates the distance between two geolocations in kilometers. The formats in which the geolocation input string can be supplied are listed below. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The image-exif-data function and the Exif metadata's @Geolocation attribute can be used to supply geolocation input strings.

Examples
- altova:geolocation-distance-km("33.33 -22.22", "48°51'29.6"N 24°17'40.2"W") returns the xs:decimal 4183.08132372392

Geolocation input string formats:
The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (" ") while unit indicators that are escaped are highlighted in blue (" ").

- Degrees, minutes, decimal seconds, with suffixed orientation (N/S, W/E)
  D°M'S.SS"N/S  D°M'S.SS"W/E
  Example: 33°55.111"N 22°44.666"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°55.111"  -22°44.666"W

- Degrees, decimal minutes, with suffixed orientation (N/S, W/E)
  D°M.MM'N/S  D°M.MM'W/E
  Example: 33°55.55°N 22°44.44°W

- Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (N/W) is optional
  +/-D°M.MM"  +/-D°M.MM"
  Example: +33°55.55°  -22°44.44°

- Decimal degrees, with suffixed orientation (N/S, W/E)
  D.DDN/S  D.DDW/E
  Example: 33.33N 22.22W

- Decimal degrees, with prefixed sign (+/-); the plus sign for (N/W) is optional
XSLT and XPath/XQuery Functions

+/-D.DD  +/−D.DD

Example:  33.33  -22.22

Examples of format-combinations:
33.33N  −22°44'66.66"
33.33  22°44'66.66"W
33.33  22.3

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute Geolocation from standard Exif metadata tags. Geolocation is a concatenation of four Exif tags: GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (see table below).

<table>
<thead>
<tr>
<th>GPSLatitude</th>
<th>GPSLatitudeRef</th>
<th>GPSLongitude</th>
<th>GPSLongitudeRef</th>
<th>Geolocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>33 51</td>
<td>S</td>
<td>151 13</td>
<td>E</td>
<td>33°51'21.91&quot;S 151°13'11.73&quot;E</td>
</tr>
</tbody>
</table>

geolocation-distance-mi [altova:]

altova:geolocation-distance-mi(GeolocationInputString-1 as xs:string, GeolocationInputString-2 as xs:string) as xs:decimal

Calculates the distance between two geolocations in miles. The formats in which a geolocation input string can be supplied are listed below. Latitude values range from +90 to −90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The image-exif-data function and the Exif metadata's @Geolocation attribute can be used to supply geolocation input strings.

Examples
• altova:geolocation-distance-mi("33.33  −22.22", "48°51'29.6""N 24°17'40.2""W") returns the xs:decimal 2599.40652340653

Geolocation input string formats:
The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from +90 to −90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow ("”) while unit indicators that are escaped are highlighted in blue.
Degrees, minutes, decimal seconds, with suffixed orientation (N/S, W/E)

\[
D^\circ M'.S''.SS''N/S \quad D^\circ M'.S''.SS''W/E
\]

Example: \(33^\circ55'11.11''N \quad 22^\circ44'66.66''W\)

Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (N/W) is optional

\[
+/-D^\circ M'.S''.SS'' \quad +/-D^\circ M'.S''.SS''
\]

Example: \(33^\circ55'11.11'' \quad -22^\circ44'66.66''\)

Degrees, decimal minutes, with suffixed orientation (N/S, W/E)

\[
D^\circ M.MM'N/S \quad D^\circ M.MM'W/E
\]

Example: \(33^\circ55.55'N \quad 22^\circ44.44'W\)

Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (N/W) is optional

\[
+/-D^\circ M.MM' \quad +/-D^\circ M.MM'
\]

Example: \(+33^\circ55.55' \quad -22^\circ44.44'\)

Decimal degrees, with suffixed orientation (N/S, W/E)

\[
D.DD/N/S \quad D.DD/W/E
\]

Example: \(33.33N \quad 22.22W\)

Decimal degrees, with prefixed sign (+/-); the plus sign for (N/W) is optional

\[
+/-D.DD \quad +/-D.DD
\]

Example: \(33.33 \quad -22.22\)

Examples of format-combinations:

\[
33 51 21.91 \quad 22^\circ44'66.66''W
33 33 22^\circ44'66.66''W
33.33 22.22
\]

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute Geolocation from standard Exif metadata tags. Geolocation is a concatenation of four Exif tags: GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (see table below).

<table>
<thead>
<tr>
<th>GPSLatitude</th>
<th>GPSLatitude Ref</th>
<th>GPSLongitude</th>
<th>GPSLongitude Ref</th>
<th>Geolocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>33 51 21.91</td>
<td>S</td>
<td>151 13 11.73</td>
<td>E</td>
<td>33°51'21.91&quot;S 151°13'11.73&quot;E</td>
</tr>
</tbody>
</table>

geolocation-within-polygon [altova:]

\[
altova:geolocation-within-polygon(Geolocation as xs:string, ((PolygonPoint as xs:string)+)) as xs:boolean
\]

XP3 XQ3
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") 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)
  \[ D°M'.SS"N/S  D°M'.SS"W/E \]
  Example: 33°55'11.11"N  22°44'66.66"W

- Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (N/W) is optional
  \[ +/-D°M'.SS"  +/-D°M'.SS" \]
  Example: 33°55'11.11"  -22°44'66.66"W

- Degrees, decimal minutes, with suffixed orientation (N/S, W/E)
  \[ D°.MM"N/S  D°.MM"W/E \]
  Example: 33°55.55"N  22°44.44"W

- Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (N/W) is optional
  \[ +/-D°.MM"  +/-D°.MM" \]
Example: +33°55.55' -22°44.44'

- Decimal degrees, with suffixed orientation (N/S, W/E)
  \[ \text{D.DD}/\text{S} \quad \text{D.DD}/\text{W} \]
  Example: 33.33N 22.22W

- Decimal degrees, with prefixed sign (+/-); the plus sign for (N/W) is optional
  \[ \text{+/D.DD} \quad \text{+/-D.DD} \]
  Example: 33.33  -22.22

<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&quot;E</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).

\[ \text{geolocation-within-rectangle \{altova\}} \]

\[ \text{altova:geolocation-within-rectangle(}\text{Geolocation \text{as \text{xs:string}}, \text{RectCorner-1 \text{as \text{xs:string}}, RectCorner-2 \text{as \text{xs:string}}} \text{\text{as \text{xs:boolean}} \text{\text{XP3 XQ3}}} \text{returns \text{true()}; otherwise it returns \text{false()}}) \]

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 \( \text{formats listed below} \). If the Geolocation argument is within the rectangle, then the function returns \text{true()}; otherwise it returns \text{false()}. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The \text{image-exif-data} function and the Exif metadata's \text{@Geolocation} attribute can be used to supply geolocation input strings.

Examples

- \text{altova:geolocation-within-rectangle(}"33 -22", "58 -32", "-48 24") returns \text{true()}
- \text{altova:geolocation-within-rectangle(}"33 -22", "58 -32", "48 24") returns \text{false()}
- \text{altova:geolocation-within-rectangle(}"33 -22", "58 -32", "48°51'29.6"S"
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)
  \[D^\circ M^\prime S^\prime.\prime\prime N/S\quad D^\circ M^\prime S^\prime.\prime\prime W/E\]
  \[Example: \quad 33^\circ 55^\prime 11.11^\prime\prime N\quad 22^\circ 44^\prime 66.66^\prime\prime W\]

- Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (N/W) is optional
  \[+/-D^\circ M^\prime S^\prime.\prime\prime\quad +/-D^\circ M^\prime S^\prime.\prime\prime\]
  \[Example: \quad 33^\circ 55^\prime 11.11^\prime\prime\quad -22^\circ 44^\prime 66.66^\prime\prime\]

- Degrees, decimal minutes, with suffixed orientation (N/S; W/E)
  \[D^\circ M.\prime M\prime N/S\quad D^\circ M.\prime M\prime W/E\]
  \[Example: \quad 33^\circ 55.55^\prime N\quad 22^\circ 44.44^\prime W\]

- Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (N/W) is optional
  \[+/-D^\circ M.\prime M\prime\quad +/-D^\circ M.\prime M\prime\]
  \[Example: \quad +33^\circ 55.55^\prime\quad -22^\circ 44.44^\prime\]

- Decimal degrees, with suffixed orientation (N/S; W/E)
  \[D.DDN/S\quad D.DDW/E\]
  \[Example: \quad 33.33N\quad 22.22W\]

- Decimal degrees, with prefixed sign (+/-); the plus sign for (N/W) is optional
  \[+/-D.DD\quad +/-D.DD\]
  \[Example: \quad 33.33\quad -22.22\]

Examples of format-combinations:
\[33.33N\quad -22^\circ 44^\prime 66.66^\prime\prime\]
\[33.33\quad 22^\circ 44^\prime 66.66^\prime\prime W\]
\[33.33\quad 22.\]
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>

[ Top ]
XPath/XQuery Functions: Image-Related

The following image-related XPath/XQuery extension functions are supported in the current version of StyleVision and can be used in (i) XPath expressions in an XSLT context, or (ii) XQuery expressions in an XQuery document.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the Altova extension functions namespace, http://www.altova.com/xslt-extensions, and are indicated in this section with the prefix altova:, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

| XPath functions (used in XPath expressions in XSLT):          |
|----------------------|------------------|
| XP1                  |
| XP2                  |
| XP3                  |

| XSLT functions (used in XPath expressions in XSLT):         |
|----------------------|------------------|
| XSLT1                |
| XSLT2                |
| XSLT3                |

| XQuery functions (used in XQuery expressions in XQuery):     |
|----------------------|------------------|
| XQ1                  |
| XQ3                  |

**suggested-image-file-extension [altova:]**

altova:suggested-image-file-extension(Base64String as string) as string?  XP3  XQ3

Takes the Base64 encoding of an image file as its argument and returns the file extension of the image as recorded in the Base64-encoding of the image. The returned value is a suggestion based on the image type information available in the encoding. If this information is not available, then an empty string is returned. This function is useful if you wish to save a Base64 image as a file and wish to dynamically retrieve an appropriate file extension.

**Examples**

- altova:suggested-image-file-extension(/MyImages/MobilePhone/Image20141130.01) returns 'jpg'
- altova:suggested-image-file-extension($XML1/Staff/Person/@photo) returns ''

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.

**image-exif-data [altova:]**

altova:image-exif-data(Base64BinaryString as string) as element?  XP3  XQ3

Takes a Base64-encoded image as its argument and returns an element called Exif that contains the Exif metadata of the image. The Exif metadata is created as attribute-value pairs of the Exif element. The attribute names are the Exif data tags found in the Base64 encoding. The list of Exif-specification tags is given below. If a vendor-specific tag is present in the Exif data, this tag and its value will also be returned as an attribute-value pair. Additional to the standard Exif metadata tags (see list below), Altova-specific attribute-value
pairs are also generated. These Altvā Exif attributes are listed below.

**Examples**

- To access any one attribute, use the function like this:
  
  ```xml
  image-exif-data(//MyImages/Image20141130.01)/@GPSLatitude
  ```

- 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</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>

**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, and 7. (These orientations are obtained by flipping image 1 across its vertical center axis to get the image with a value of 2, and then rotating this image in 90-degree jumps clockwise to get the values of 7, 4, and 5, respectively).
Listing of standard Exif meta tags

- ImageWidth
- ImageLength
- BitsPerSample
- Compression
- PhotometricInterpretation
- Orientation
- SamplesPerPixel
- PlanarConfiguration
- YCbCrSubSampling
- YCbCrPositioning
- XResolution
- YResolution
- ResolutionUnit
- StripOffsets
- RowsPerStrip
- StripByteCounts
- JPEGInterchangeFormat
- JPEGInterchangeFormatLength
- TransferFunction
- WhitePoint
- PrimaryChromaticities
- YCbCrCoefficients
- ReferenceBlackWhite
- DateTime
- ImageDescription
- Make
- Model
- Software
• Artist
• Copyright

-----------------------------------
• ExifVersion
• FlashpixVersion
• ColorSpace
• ComponentsConfiguration
• CompressedBitsPerPixel
• PixelXDimension
• PixelYDimension
• MakerNote
• UserComment
• RelatedSoundFile
• DateTimeOriginal
• DateTimeDigitized
• SubSecTime
• SubSecTimeOriginal
• SubSecTimeDigitized
• ExposureTime
• FNumber
• ExposureProgram
• SpectralSensitivity
• ISOSpeedRatings
• OECF
• ShutterSpeedValue
• ApertureValue
• BrightnessValue
• ExposureBiasValue
• MaxApertureValue
• SubjectDistance
• MeteringMode
• LightSource
• Flash
• FocalLength
• SubjectArea
• FlashEnergy
• SpatialFrequencyResponse
• FocalPlaneXResolution
• FocalPlaneYResolution
• FocalPlaneResolutionUnit
• SubjectLocation
• ExposureIndex
• SensingMethod
• FileSource
• SceneType
• CFAPattern
• CustomRendered
• ExposureMode
• WhiteBalance
• DigitalZoomRatio
• FocalLengthIn35mmFilm
• SceneCaptureType
• GainControl
• Contrast
• Saturation
- Sharpness
- DeviceSettingDescription
- SubjectDistanceRange
- ImageUniqueID

-------------------------------
- GPSVersionID
- GPSLatitudeRef
- GPSLatitude
- GPSLongitudeRef
- GPSLongitude
- GPSAltitudeRef
- GPSAltitude
- GPSTimeStamp
- GPSSatellites
- GPSstatus
- GPSMeasureMode
- GPSDOP
- GPSSpeedRef
- GPSSpeed
- GPSTrackRef
- GPSTrack
- GPSImgDirectionRef
- GPSImgDirection
- GPSMapDatum
- GPSDestLatitudeRef
- GPSDestLatitude
- GPSDestLongitudeRef
- GPSDestLongitude
- GPSDestBearingRef
- GPSDestBearing
- GPSDestDistanceRef
- GPSDestDistance
- GPSProcessingMethod
- GPSAreaInformation
- GPSDateStamp
- GPSDifferential

[ Top ]
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.

| XPath functions (used in XPath expressions in XSLT): | XP1 XP2 XP3 |
| XSLT functions (used in XPath expressions in XSLT): | XSLT1 XSLT2 XSLT3 |
| XQuery functions (used in XQuery expressions in XQuery): | XQ1 XQ3 |

**Auto-numbering 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")`.

- **reset-auto-number [altova:]**
  
  `altova:reset-auto-number(ID as xs:string)`  

  Resets the auto-numbering counter.
This function resets the number of the auto-numbering counter named in the ID argument. The number is reset to the number specified by the StartsWith argument of the altova:generate-auto-number function that created the counter named in the ID argument.

Examples

- `altova:reset-auto-number("ChapterNumber")` resets the number of the auto-numbering counter named ChapterNumber that was created by the altova:generate-auto-number function. The number is reset to the value of the StartsWith argument of the altova:generate-auto-number function that created ChapterNumber.

### Numeric functions

#### hex-string-to-integer [altova:]

`altova:hex-string-to-integer(HexString as xs:string) as xs:integer`  
Takes a string argument that is the Base-16 equivalent of an integer in the decimal system (Base-10), and returns the decimal integer.

Examples

- `altova:hex-string-to-integer('1')` returns 1
- `altova:hex-string-to-integer('9')` returns 9
- `altova:hex-string-to-integer('A')` returns 10
- `altova:hex-string-to-integer('B')` returns 11
- `altova:hex-string-to-integer('F')` returns 15
- `altova:hex-string-to-integer('G')` returns an error
- `altova:hex-string-to-integer('10')` returns 16
- `altova:hex-string-to-integer('01')` returns 1
- `altova:hex-string-to-integer('20')` returns 32
- `altova:hex-string-to-integer('21')` returns 33
- `altova:hex-string-to-integer('SA')` returns 90
- `altova:hex-string-to-integer('USA')` returns an error

#### integer-to-hex-string [altova:]

`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'
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").
**XPath/XQuery Functions: Sequence**

Altova's sequence extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the Altova extension functions namespace, [http://www.altova.com/xslt-extensions](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</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</td>
</tr>
</tbody>
</table>

**attributes** [altova:]

`altova:attributes(AttributeName  as xs:string)  as attribute()*`  

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()*`  

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`; `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  XQ3

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 element()*`  
XP3  XQ3

Returns all elements that have a local name which is the same as the name supplied in the input argument, `ElementName`. The search is case-sensitive and conducted along the `child::` axis. The context node must be the parent node of the element/s being searched for. The second argument is a string containing option flags. Available flags are:
- `r` = switches to a regular-expression search; `ElementName` must then be a regular-expression search string;
- `f` = If this option is specified, then `ElementName` provides a full match; otherwise `ElementName` need only partially match an element name to return that element. For example: if `f` is not specified, then `MyEelm` will return `MyElement`;
- `i` = switches to a case-insensitive search;
- `p` = includes the namespace prefix in the search; `ElementName` should then contain the namespace prefix, for example: `altova:MyElement`.

The flags can be written in any order. Invalid flags will generate errors. One or more flags can be omitted. The empty string is allowed, and will produce the same effect as the function having only one argument (previous signature). However, an empty sequence is not allowed.

**Examples**
- `altova:elements("MyElement", "rip")` returns `MyElement()`*
- `altova:elements("MyElement", "pri")` returns `MyElement()`*
- `altova:elements("MyElement", ")` returns `MyElement()`*
- `altova:attributes("MyEelm", "rip")` returns `MyElement()`*
- `altova:attributes("MyElements", "rfip")` returns no match
- `altova:elements("MyElement", "Rip")` returns an unrecognized-flag error.

## find-first [altova:]

### `altova:find-first(Sequence as item()*) (Condition( Sequence-Item as xs:boolean)) as item()?`  
XP3  XQ3

Returns the first element in the sequence that matches the condition.
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() is returned as the result of altova:find-first, and the iteration stops.

**Examples**

- **altova:find-first(5 to 10, function($a) {$a mod 2 = 0})** returns xs:integer 6
  
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 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 altova:find-first (in this case 6).

- **altova:find-first((1 to 10), (function($a) {$a+3=7}))** returns xs:integer 4

**Further examples**

If the file C:\Temp\Customers.xml exists:

- **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:

- **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 doc-available function can be used for Condition, the second argument of altova:find-first, 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()` evaluates to `true()` and that string is returned as the result of the `altova:find-first` function. Note about the `doc-available()` function: Relative paths are resolved relative to the the current base URI, which is by default the URI of the XML document from which the function is loaded.

### find-first-combination [altova:]

```
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 XQ3`
```

This function takes three arguments:

- The first two arguments, `Seq-01` and `Seq-02`, are sequences of one or more items of any datatype.
- The third argument, `Condition`, is a reference to an XPath function that takes two arguments (has an arity of 2) and returns a boolean.

The items of `Seq-01` and `Seq-02` are passed in ordered pairs (one item from each sequence making up a pair) as the arguments of the function in `Condition`. The pairs are ordered as follows.

If `Seq-01 = X1, X2, X3 ... Xn`
And `Seq-02 = Y1, Y2, Y3 ... Yn`
Then `(X1 Y1), (X1 Y2), (X1 Y3) ... (X1 Yn), (X2 Y1), (X2 Y2) ... (Xn Yn)`

The first ordered pair that causes the `Condition` function to evaluate to `true()` is returned as the result of `altova:find-first-combination`. Note that: (i) If the `Condition` function iterates through the submitted argument pairs and does not once evaluate to `true()`, then `altova:find-first-combination` returns `No results`; (ii) The result of `altova:find-first-combination` will always be a pair of items (of any datatype) or no item at all.

#### Examples

- `altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a +$b = 32})` returns the sequence of `xs:integers` `{11, 21}`
- `altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a +$b = 33})` returns the sequence of `xs:integers` `{11, 22}`
- `altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a +$b = 34})` returns the sequence of `xs:integers` `{11, 23}`

### find-first-pair [altova:]

```
altova:find-first-pair((Seq-01 as item(*)*), (Seq-02 as item(*)*), [Condition( Seq-01-Item, Seq-02-Item as xs:boolean)] as item()*) `XP3 XQ3`
```

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 XQ3`  

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 \texttt{Condition} function to evaluate to \texttt{true()}, and so its index position in the sequence, 1, is returned. The second example returns \textit{No results} because no pair gives a sum of 33.

\subsection*{\texttt{find-first-pos [altova:]} [XSLT and XPath/XQuery Functions]}

\begin{verbatim}
altova:find-first-pos((Sequence as item()), (Condition( Sequence-Item as xs:boolean)) as xs:integer) as xs:integer

\end{verbatim}

This function takes two arguments. The first argument is a sequence of one or more items of any datatype. The second argument, \texttt{Condition}, is a reference to an XPath function that takes one argument (has an arity of 1) and returns a boolean. Each item of \texttt{Sequence} is submitted, in turn, to the function referenced in \texttt{Condition}. (\textit{Remember:} This function takes a single argument.) The first \texttt{Sequence} item that causes the function in \texttt{Condition} to evaluate to \texttt{true()} has its index position in \texttt{Sequence} returned as the result of \texttt{altova:find-first-pos}, and the iteration stops.

\begin{itemize}
\item \texttt{altova:find-first-pos(5 to 10, function($a) {$a mod 2 = 0})} returns \texttt{xs:integer 2}
\end{itemize}

The \texttt{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 \texttt{Sequence} argument of \texttt{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 \texttt{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).

\begin{itemize}
\item \texttt{altova:find-first-pos((2 to 10), (function($a) {$a+3=7}))} returns \texttt{xs:integer 3}
\end{itemize}

\begin{itemize}
\item \texttt{altova:find-first-pos(C:\Temp\Customers.xml, "http://www.altova.com/index.html"), (doc-available#1)} returns 1
\end{itemize}

If the file \texttt{C:\Temp\Customers.xml} exists:

If the file \texttt{C:\Temp\Customers.xml} does not exist, and \texttt{http://www.altova.com/index.html} exists:

If the file \texttt{C:\Temp\Customers.xml} does not exist, and \texttt{http://www.altova.com/index.html} also does not exist:

\begin{itemize}
\item \texttt{altova:find-first-pos(C:\Temp\Customers.xml, "http://www.altova.com/index.html"), (doc-available#1)} returns no result
\end{itemize}

\textit{Notes about the examples given above}
• The XPath 3.0 function, doc-available, takes a single string argument, which is used as a URI, and returns true if a document node is found at the submitted URI. (The document at the submitted URI must therefore be an XML document.)
• The doc-available function can be used for condition, the second argument of altova:find-first-pos, because it takes only one argument (arity=1), because it takes an item() as input (a string which is used as a URI), and returns a boolean value.
• Notice that the doc-available function is only referenced, not called. The #1 suffix that is attached to it indicates a function with an arity of 1. In its entirety doc-available#1 simply means: Use the doc-available() function that has arity=1, passing to it as its single argument, in turn, each of the items in the first sequence. As a result, each of the two strings will be passed to doc-available(), which uses the string as a URI and tests whether a document node exists at the URI. If one does, the doc-available() function evaluates to true() and the index position of that string in the sequence is returned as the result of the altova:find-first-pos function. Note about the doc-available() function: Relative paths are resolved relative to the the current base URI, which is by default the URI of the XML document from which the function is loaded.

substitute-empty [altova:]
altova:substitute-empty(FirstSequence as item()* , SecondSequence as item()) as item() XP3 XP3
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)
**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</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</td>
</tr>
</tbody>
</table>

### camel-case [altova:]

*`altova:camel-case(InputString as xs:string) as xs:string`*  
XP3 XQ3

Returns the input string `InputString` in CamelCase. The string is analyzed using the regular expression `\s` (which is a shortcut for the whitespace character). The first non-whitespace character after a whitespace or sequence of consecutive whitespaces is capitalized. The first character in the output string is capitalized.

**Examples**

- `altova:camel-case("max")` returns `Max`  
- `altova:camel-case("max max")` returns `Max Max`  
- `altova:camel-case("file01.xml")` returns `File01.xml`  
- `altova:camel-case("file01.xml file02.xml")` returns `File01.xml File02.xml`  
- `altova:camel-case("file01.xml file02.xml")` returns `File01.xml File02.xml`  
- `altova:camel-case("file01.xml -file02.xml")` returns `File01.xml -file02.xml`  

*`altova:camel-case(InputString as xs:string, SplitChars as xs:string, IsRegex as xs:boolean) as xs:string`*  
XP3 XQ3

Converts the input string `InputString` to camel case by using `SplitChars` to determine the character/s that trigger the next capitalization. `SplitChars` is used as a regular expression when `IsRegex = true()`, or as plain characters when `IsRegex = false()`. The first character in the output string is capitalized.

**Examples**

- `altova:camel-case("setname getname", "set|get", true())` returns `setName getName`  
- `altova:camel-case("altova\documents\testcases", ",", false())` returns `Altova\Documents\Testcases`
XSLT and XPath/XQuery Functions

char [altova:]

\texttt{altova:char(Position as xs:integer) as xs:string \ XP3 XQ3}

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:
  - \texttt{altova:char(2)} returns 2
  - \texttt{altova:char(5)} returns A
  - \texttt{altova:char(9)} returns the empty string.
  - \texttt{altova:char(-2)} returns the empty string.

first-chars [altova:]

\texttt{altova:first-chars(X-Number as xs:integer) as xs:string \ XP3 XQ3}

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

last-chars [altova:]

\texttt{altova:last-chars(InputString as xs:string, X-Number as xs:integer) as xs:string \ XP3 XQ3}

Returns a string containing the first \texttt{X-Number} of characters of the string submitted as the \texttt{InputString} argument.

\textbf{Examples}

- \texttt{altova:last-chars("2014-01-15", 5)} returns 2014-
- \texttt{altova:last-chars("USA", 1)} returns U
**altova:last-chars**

**altova:last-chars(X-Number as xs:integer) as xs:string**  
**XP3 XQ3**

Returns a string containing the last X-Number of characters of the string obtained by converting the value of the context item to `xs:string`.

**Examples**

- If the context item is `1234ABCD`:
  - `altova:last-chars(2)` returns `CD`
  - `altova:last-chars(5)` returns `4ABCD`
  - `altova:last-chars(9)` returns `1234ABCD`

**altova:last-chars(InputString as xs:string, X-Number as xs:integer) as xs:string**  
**XP3 XQ3**

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 XQ3**

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 `'ZAP'`
- `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**  
**XP3 XQ3**

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:]

```xquery
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.

- `altova:repeat-string("Altova ", 3)` returns "Altova #Altova #Altova #"

### Substring-After-Last [altova:]

```xquery
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.

- `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', '')` returns 'ABCDEFGH'
- `altova:substring-after-last('ABCD-ABCD-ABCD', 'BCD')` returns ''

### Substring-Before-Last [altova:]

```xquery
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.

- `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` XP3 XQ3

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')` returns 3

`altova:substring-pos(StringToCheck as xs:string, StringToFind as xs:string, Integer as xs:integer)` as `xs:integer` XP3 XQ3

Returns the character position of `StringToFind` in the string `StringToCheck`. The search for `StringToFind` starts from the character position given by the `Integer` argument; the character substring before this position is not searched. The returned integer, however, is the position of the found string within the entire string `StringToCheck`. This signature is useful for finding the second or a later position of a string that occurs multiple times with the `StringToCheck`. If `StringToFind` does not occur within `StringToCheck`, the integer 0 is returned.

**Examples**

• `altova:substring-pos('Altova', 'to', 1)` returns 3
• `altova:substring-pos('Altova', 'to', 3)` returns 3
• `altova:substring-pos('Altova', 'to', 4)` returns 0
• `altova:substring-pos('Altova-Altova', 'to', 0)` returns 3
• `altova:substring-pos('Altova-Altova', 'to', 4)` returns 10

**trim-string [altova:]**

`altova:trim-string(InputString as xs:string)` as `xs:string` XP3 XQ3

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(" HelloWorld")` returns "HelloWorld"
• `altova:trim-string("Hello World")` returns "Hello World"
• `altova:trim-string("Hello   World")` returns "Hello   World"

### trim-string-left [altova:]

```
altova:trim-string-left(InputString as xs:string) as xs:string  XP3 XQ3
```

This function takes an `xs:string` argument, removes any leading whitespace, and returns a left-trimmed `xs:string`.

#### Examples

- `altova:trim-string-left("   Hello World   ")` returns "Hello World   
- `altova:trim-string-left("Hello World   ")` returns "Hello World   
- `altova:trim-string-left("   Hello World")` returns "Hello World"
- `altova:trim-string-left("Hello World")` returns "Hello World"
- `altova:trim-string-left("Hello   World")` returns "Hello   World"

### trim-string-right [altova:]

```
altova:trim-string-right(InputString as xs:string) as xs:string  XP3 XQ3
```

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"
XPath/XQuery Functions: Miscellaneous

The following general purpose XPath/XQuery extension functions are supported in the current version of StyleVision 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.

<table>
<thead>
<tr>
<th>XPath functions (used in XPath expressions in XSLT):</th>
<th>XP1</th>
<th>XP2</th>
<th>XP3</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLT functions (used in XPath expressions in XSLT):</td>
<td>XSLT1</td>
<td>XSLT2</td>
<td>XSLT3</td>
</tr>
<tr>
<td>XQuery functions (used in XQuery expressions in XQuery):</td>
<td>XQ1</td>
<td>XQ3</td>
<td></td>
</tr>
</tbody>
</table>

**URI functions**

- **get-temp-folder [altova:]**

  altova:get-temp-folder() as xs:string XP2 XQ1 XP3 XQ3

  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`.

  [ Top ]
Chart Functions

The chart functions listed below enable you to create, generate, and save charts as images. They are supported in the current version of your Altova product in the manner described below. However, note that in future versions of your product, support for one or more of these functions might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

The chart functions are XPath functions (not XSLT functions), and organized into two groups:

- Functions for generating and saving charts
- Functions for creating charts

Note: Chart functions are supported only in Altova’s Server products and the Enterprise Editions of Altova products.

Note: Supported image formats for charts in server editions are jpg, png, and bmp. The best option is png because it is lossless and compressed. In Enterprise editions, the supported formats are jpg, png, bmp, and gif.

Functions for generating and saving charts

These functions take the chart object (obtained with the chart creation functions) and either generate an image or save an image to file.

\texttt{altova:generate-chart-image} ($\text{chart}$, $\text{width}$, $\text{height}$, $\text{encoding}$) as atomic

where

- $\text{chart}$ is the chart extension item obtained with the \texttt{altova:create-chart} function
- $\text{width}$ and $\text{height}$ must be specified with a length unit
- $\text{encoding}$ may be \texttt{binarytobase64} or \texttt{binarytobase16}

The function returns the chart image in the specified encoding.

\texttt{altova:generate-chart-image} ($\text{chart}$, $\text{width}$, $\text{height}$, $\text{encoding}$, $\text{imagetype}$) as atomic

where

- $\text{chart}$ is the chart extension item obtained with the \texttt{altova:create-chart} function
- $\text{width}$ and $\text{height}$ must be specified with a length unit
- $\text{encoding}$ may be \texttt{base64Binary} or \texttt{hexBinary}
- $\text{imagetype}$ may be one of the following image formats: png, gif, bmp, jpg, jpeg. Note that gif is not supported on server products. \textit{Also see note at top of page.}

The function returns the chart image in the specified encoding and image format.
altova:save-chart-image ($chart, $filename, $width, $height) as empty()  *(Windows only)*

where

- $chart is the chart extension item obtained with the altova:create-chart function
- $filename is the path to and name of the file to which the chart image is to be saved
- $width and $height must be specified with a length unit

The function saves the chart image to the file specified in $filename.

altova:save-chart-image ($chart, $filename, $width, $height, $imagetype) as empty()  *(Windows only)*

where

- $chart is the chart extension item obtained with the altova:create-chart function
- $filename is the path to and name of the file to which the chart image is to be saved
- $width and $height must be specified with a length unit
- $imagetype may be one of the following image formats: png, gif, bmp, jpg, jpeg. Note that gif is not supported on server products. Also see note at top of page.

The function saves the chart image to the file specified in $filename in the image format specified.

**Functions for creating charts**

The following functions are used to create charts.

altova:create-chart ($chart-config, $chart-data-series*) as chart extension item

where

- $chart-config is the chart-config extension item obtained with the altova:create-chart-config function or via the altova:create-chart-config-from-xml function
- $chart-data-series is the chart-data-series extension item obtained with the altova:create-chart-data-series function or altova:create-chart-data-series-from-rows function

The function returns a chart extension item, which is created from the data supplied via the arguments.
**altova:create-chart-config**($type-name, $title) as chart-config extension item

where

- $type-name specifies the type of chart to be created: Pie, Pie3d, BarChart, BarChart3d, BarChart3dGrouped, LineChart, ValueLineChart, RoundGauge, BarGauge
- $title is the name of the chart

The function returns a chart-config extension item containing the configuration information of the chart.

**altova:create-chart-config-from-xml**($xml-struct) as chart-config extension item

where

- $xml-struct is the XML structure containing the configuration information of the chart

The function returns a chart-config extension item containing the configuration information of the chart. This information is supplied in an XML data fragment.

**altova:create-chart-data-series**($series-name?, $x-values*, $y-values*) as chart-data-series extension item

where

- $series-name specifies the name of the series
- $x-values gives the list of X-Axis values
- $y-values gives the list of Y-Axis values

The function returns a chart-data-series extension item containing the data for building the chart: that is, the names of the series and the Axes data.

**altova:create-chart-data-row**(x, y1, y2, y3, ...) as chart-data-x-Ny-row extension item

where

- x is the value of the X-Axis column of the chart data row
- yN are the values of the Y-Axis columns

The function returns a chart-data-x-Ny-row extension item, which contains the data for the X-Axis column and Y-Axis columns of a single series.
altova:create-chart-data-series-from-rows($series-names as xs:string*, $row*) as chart-data-series extension item

where

- $series-name is the name of the series to be created
- $row is the chart-data-x-Ny-row extension item that is to be created as a series

The function returns a chart-data-series extension item, which contains the data for the X-Axis and Y-Axes of the series.

altova:create-chart-layer($chart-config, $chart-data-series*) as chart-layer extension item

where

- $chart-config is the chart-config extension item obtained with the altova:create-chart-config function or or via the altova:create-chart-config-from-xml function
- $chart-data-series is the chart-data-series extension item obtained with the altova:create-chart-data-series function or altova:create-chart-data-series-from-rows function

The function returns a chart-layer extension item, which contains chart-layer data.

altova:create-multi-layer-chart($chart-config, $chart-data-series*, $chart-layer*)

where

- $chart-config is the chart-config extension item obtained with the altova:create-chart-config function or or via the altova:create-chart-config-from-xml function
- $chart-data-series is the chart-data-series extension item obtained with the altova:create-chart-data-series function or altova:create-chart-data-series-from-rows function
- $chart-layer is the chart-layer extension item obtained with the altova:create-chart-layer function

The function returns a multi-layer-chart item.
The function returns a multi-layer-chart item.

**Chart Data XML Structure**

Given below is the XML structure of chart data, how it might appear for the Altova extension functions for charts. This affects the appearance of the specific chart. Not all elements are used for all chart kinds, e.g. the `<Pie>` element is ignored for bar charts.

**Note:** Chart functions are supported only in the Enterprise and Server Editions of Altova products.

```xml
<chart-config>
    <General
        SettingsVersion="1" must be provided
        ChartKind="BarChart" Pie, Pie3d, BarChart, StackedBarChart, BarChart3d, BarChart3dGrouped, LineChart, ValueLineChart, AreaChart, StackedAreaChart, RoundGauge, BarGauge, CandleStick
        BKColor="#ffffff" Color
        BKColorGradientEnd="#ffffff" Color. In case of a gradient, BKColor and BKColorGradientEnd define the gradient's colors
        BKMode="#ffffff" Solid, HorzGradient, VertGradient
        BKFile="Path+Filename" String. If file exists, its content is drawn over the background.
        BKFileMode="Stretch" Stretch, ZoomToFit, Center, Tile
        ShowBorder="1" Bool
        PlotBorderColor="#000000" Color
        PlotBKColor="#ffffff" Color
        Title="" String
        ShowLegend="1" Bool
        OutsideMargin="3.%" PercentOrPixel
        TitleToPlotMargin="3.%" PercentOrPixel
        LegendToPlotMargin="3.%" PercentOrPixel
        Orientation="vert" Enumeration: possible values are: vert, horz
    >
    <TitleFont
        Color="#000000" Color
        Name="Tahoma" String
        Bold="1" Bool
        Italic="0" Bool
```
<LegendFont
    Color="#000000"
    Name="Tahoma"
    Bold="0"
    Italic="0"
    Underline="0"
    MinFontHeight="10.pt"
    Size="3.5%" />

<AxisLabelFont
    Color="#000000"
    Name="Tahoma"
    Bold="1"
    Italic="0"
    Underline="0"
    MinFontHeight="10.pt"
    Size="5.%" />
</General>

<Line
    ConnectionShapeSize="1.%" PercentOrPixel
    DrawFilledConnectionShapes="1" Bool
    DrawOutlineConnectionShapes="0" Bool
    DrawSlashConnectionShapes="0" Bool
    DrawBackslashConnectionShapes="0" Bool />

<Bar
    ShowShadow="1" Bool
    ShadowColor="#a0a0a0" Color
    OutlineColor="#000000" Color
    ShowOutline="1" Bool />

/Area
    Transparency="0" UINT (0-255) 255 is fully transparent, 0 is opaque
    OutlineColor="#000000" Color
    ShowOutline="1" Bool />

<CandleStick
    FillHighClose="0" Bool. If 0, the body is left empty. If 1, FillColorHighClose is used for the candle body
    FillColorHighClose="#ffffff" Color. For the candle body when close > open
    FillHighOpenWithSeriesColor="1" Bool. If true, the series color is used to fill the candlebody when open > close
    FillColorHighOpen="#000000" Color. For the candle body when open > close and FillHighOpenWithSeriesColor is false />
<Colors User-defined color scheme: By default this element is empty except for the
style and has no Color attributes

UseSubsequentColors = "1"  Boolean. If 0, then color in overlay is used. If 1, then subsequent colors from previous chart layer is used

Style="User"  Possible values are: "Default", "Grayscale", "Colorful", "Pastel", "User"

Colors="#52aca0"  Color: only added for user defined color set
Colors1="#d3c15d"  Color: only added for user defined color set
Colors2="#8971d8"  Color: only added for user defined color set
...
ColorsN=""  Up to ten colors are allowed in a set: from Colors to Colors9

</Colors>

<Pie>

ShowLabels="1"  Bool
OutlineColor="#404040"  Color
ShowOutline="1"  Bool
StartAngle="0."  Double
Clockwise="1"  Bool
Draw2dHighlights="1"  Bool
Transparency="0"  Int (0 to 255: 0 is opaque, 255 is fully transparent)
DropShadowColor="#c0c0c0"  Color
DropShadowSize="5.%"  PercentOrPixel
PieHeight="10.%"  PercentOrPixel. Pixel values might be different in the result because of 3d tilting

Tilt="40.0"  Double (10 to 90: The 3d tilt in degrees of a 3d pie)
ShowDropShadow="1"  Bool
ChartToLabelMargin="10.%"  PercentOrPixel
AddValueToLabel="0"  Bool
AddPercentToLabel="0"  Bool
AddPercentToLabels_DecimalDigits="0"  UINT (0 – 2)

</Pie>

<XY>

<XAxis>

AutoRange="1"  Bool
AutoRangeIncludesZero="1"  Bool
RangeFrom="0."  Double: manual range
RangeTill="1."  Double: manual range
LabelToAxisMargin="3.%"  PercentOrPixel
AxisLabel=""  String
AxisColor="#000000"  Color
AxisGridColor="#e6e6e6"  Color
ShowGrid="1"  Bool
UseAutoTick="1"  Bool
ManualTickInterval="1."  Double
AxisToChartMargin="0.px"  PercentOrPixel
TickSize="3.px"  PercentOrPixel
ShowTicks="1"  Bool
ShowValues="1"  Bool
AxisPosition="LeftOrBottom"  Enums: "LeftOrBottom","RightOrTop", "AtValue"
AxisPositionAtValue = "0"  Double

<ValueFont
  Color="#000000"
  Name="Tahoma"
  Bold="0"
  Italic="0"
  Underline="0"
  MinFontHeight="10.pt"
  Size="3.%"/>
</XAxis>

<YAxis
  Axis (same as for XAxis)
  AutoRange="1"
  AutoRangeIncludesZero="1"
  RangeFrom="0."
  RangeTill="1."
  LabelToAxisMargin="3.%"
  AxisLabel=""
  AxisColor="#000000"
  AxisGridColor="#e6e6e6"
  ShowGrid="1"
  UseAutoTick="1"
  ManualTickInterval="1."
  AxisToChartMargin="0.px"
  TickSize="3.px"
  ShowTicks="1"  Bool
  ShowValues="1"  Bool
  AxisPosition="LeftOrBottom"  Enums: "LeftOrBottom","RightOrTop", "AtValue"
  AxisPositionAtValue = "0"  Double

  <ValueFont
    Color="#000000"
    Name="Tahoma"
    Bold="0"
    Italic="0"
    Underline="0"
    MinFontHeight="10.pt"
    Size="3.%"/>
</YAxis>
</XY>

<XY3d
  AxisAutoSize="1"  Bool: If false, XSize and YSize define the aspect ration of x and y axis. If true, aspect ratio is equal to chart window
  XSize="100.%"  PercentOrPixel. Pixel values might be different in the result because of 3d tilting and zooming to fit chart
  YSize="100.%"  PercentOrPixel. Pixel values might be different in the result because of 3d tilting and zooming to fit chart
SeriesMargin="30.%"  PercentOrPixel. Pixel values might be different in the result because of 3d tilting and zooming to fit chart
Tilt="20."  Double. -90 to +90 degrees
Rot="20."  Double. -359 to +359 degrees
FoV="50.">  Double. Field of view: 1-120 degree
</ZAxis>

<XY3d>
<ZAxis
AutoRange="1"
AutoRangeIncludesZero="1"
RangeFrom="0."
RangeTill="1."
LabelToAxisMargin="3.%"
AxisLabel=""
AxisColor="#000000"
AxisGridColor="#e6e6e6"
ShowGrid="1"
UseAutoTick="1"
ManualTickInterval="1."
AxisToChartMargin="0.px"
TickSize="3.px" />
<ValueFont
Color="#000000"
Name="Tahoma"
Bold="0"
Italic="0"
Underline="0"
MinFontHeight="10.pt"
Size="3.%"/>
</ZAxis>
</XY3d>

<Gauge
MinVal="0."
MaxVal="100."  Double
MinAngle="225"  UINT: -359-359
SweepAngle="270"  UINT: 1 -359
BorderToTick="1.%"  PercentOrPixel
MajorTickWidth="3.px"  PercentOrPixel
MajorTickLength="4.%"  PercentOrPixel
MinorTickWidth="1.px"  PercentOrPixel
MinorTickLength="3.%"  PercentOrPixel
BorderColor="#a0a0a0"  Color
FillColor="#303535"  Color
MajorTickColor="#a0c0b0"  Color
MinorTickColor="#a0c0b0"  Color
BorderWidth="2.%"  PercentOrPixel
NeedleBaseWidth="1.5%"  PercentOrPixel
NeedleBaseRadius="5.%"  PercentOrPixel
NeedleColor="#f00000"  Color
NeedleBaseColor="#101414"  Color
TickToTickCountMargin="5.%"  PercentOrPixel
MajorTickStep="10."  Double
MinorTickStep="5."  Double
RoundGaugeBorderToColorRange="0.%"  PercentOrPixel
RoundGaugeColorRangeWidth ="6.%"  PercentOrPixel
BarGaugeRadius="5.%"  PercentOrPixel
BarGaugeMaxHeight="20.%"  PercentOrPixel
RoundGaugeNeedleLength="45.%"  PercentOrPixel
BarGaugeNeedleLength="3.%"  PercentOrPixel
>
<TicksFont
  Color="#a0c0b0"
  Name="Tahoma"
  Bold="0"
  Italic="0"
  Underline="0"
  MinFontHeight="10.pt"
  Size="4.%"
/>
<ColorRanges>
  User-defined color ranges. By default empty with no child element entries
  <Entry
    From="50.0"
    FillWithColor="1"  Bool
    Color="#00ff00"  Color
  />
  <Entry
    From="50.0"
    FillWithColor="1"
    Color="#ff0000"
  />
  ...
</ColorRanges>
</Gauge>
</chart-config>

Example: Chart Functions
The example XSLT document below shows how Altova extension functions for charts can be used. Given further below are an XML document and a screenshot of the output image generated when the XML document is processed with the XSLT document using the XSLT 2.0 or 3.0 Engine.

Note: Chart functions are supported only in the Enterprise and Server Editions of Altova products.

Note: For more information about how chart data tables are created, see the documentation of Altova's XMLSpy and StyleVision products.

XSLT document
This XSLT document (listing below) uses Altova chart extension functions to generate a pie chart. It can be used to process the XML document listed further below.

<?xml 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"
<html>
  <head>
    <title>
      <xsl:text>HTML Page with Embedded Chart</xsl:text>
    </title>
  </head>
  <body>
    <xsl:for-each select="/Data/Region[1]">
      <xsl:variable name="extChartConfig" as="item()*">
        <xsl:variable name="ext-chart-settings" as="item()*">
          <chart-config>
            <General
              SettingsVersion="1"
              ChartKind="Pie3d"
              BKColor="#ffffff"
              ShowBorder="1"
              PlotBorderColor="#000000"
              PlotBKColor="#ffffff"
              Title="{@id}"%
              ShowLegend="1"
              OutsideMargin="3.2%"
              TitleToPlotMargin="3.%"
              LegendToPlotMargin="6.%"
            >
              <TitleFont
                Color="#023d7d"
                Name="Tahoma"
                Bold="1"
                Italic="0"
                Underline="0"
                MinFontHeight="10.pt"
                Size="8.5%" />
            </General>
          </chart-config>
          <xsl:sequence select="altovaext:create-chart-config-from-xml( $ext-chart-settings )"/>
        </xsl:variable>
        <xsl:variable name="chartDataSeries" as="item()*">
          <xsl:variable name="chartDataRows" as="item()*">
            <xsl:for-each select="(Year)">
              <xsl:sequence select="altovaext:create-chart-data-row( (@id), ( . ) )"/>
            </xsl:for-each>
          </xsl:variable>
          <xsl:variable name="chartDataSeriesNames" as="xs:string*" select="( &quot;Series 1&quot; ), &apos;&apos;&apos;&apos; [1]="/>
          <xsl:sequence select="altovaext:create-chart-data-series-from-xml( $chartDataSeries, $chartDataSeriesNames )"/>
        </xsl:variable>
      </xsl:variable>
    </xsl:for-each>
  </body>
</html>
XML document
This XML document can be processed with the XSLT document above. Data in the XML document is used to generate the pie chart shown in the screenshot below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="YearlySales.xsd">
  <ChartType>Pie Chart 2D</ChartType>
  <Region id="Americas">
    <Year id="2005">30000</Year>
    <Year id="2006">90000</Year>
    <Year id="2007">120000</Year>
    <Year id="2008">180000</Year>
    <Year id="2009">140000</Year>
    <Year id="2010">100000</Year>
  </Region>
  <Region id="Europe">
    <Year id="2005">50000</Year>
    <Year id="2006">60000</Year>
    <Year id="2007">80000</Year>
    <Year id="2008">100000</Year>
    <Year id="2009">95000</Year>
    <Year id="2010">80000</Year>
  </Region>
  <Region id="Asia">
    <Year id="2005">10000</Year>
    <Year id="2006">25000</Year>
    <Year id="2007">70000</Year>
    <Year id="2008">110000</Year>
    <Year id="2009">125000</Year>
    <Year id="2010">150000</Year>
  </Region>
</Data>
```
The pie chart shown below is generated when the XML document listed above is processed with the XSLT document.
Barcode Functions

The XSLT Engine uses third-party Java libraries to create barcodes. Given below are the classes and the public methods used. The classes are packaged in AltovaBarcodeExtension.jar, which is located in the folder <ProgramFilesFolder>\Altova\Common2015\jar.

The Java libraries used are in sub-folders of the folder <ProgramFilesFolder>\Altova\Common2015\jar:

- barcode4j\barcode4j.jar (Website: http://barcode4j.sourceforge.net/)
- zxing\core.jar (Website: http://code.google.com/p/zxing/)

The license files are also located in the respective folders.

---

**The com.altova.extensions.barcode package**

The package, com.altova.extensions.barcode, is used to generate most of the barcode types.

The following classes are used:

```java
public class BarcodeWrapper
    static BarcodeWrapper newInstance( String name, String msg, int dpi, int orientation, BarcodePropertyWrapper[] arrProperties )
    double getHeightPlusQuiet()
    double getWidthPlusQuiet()
    org.w3c.dom.Document generateBarcodeSVG()
    byte[] generateBarcodePNG()
    String generateBarcodePngAsHexString()
```

```java
public class BarcodePropertyWrapper

    Used to store the barcode properties that will be dynamically set later

    BarcodePropertyWrapper( String methodName, String propertyValue )
    BarcodePropertyWrapper( String methodName, Integer propertyValue )
    BarcodePropertyWrapper( String methodName, Double propertyValue )
    BarcodePropertyWrapper( String methodName, Boolean propertyValue )
    BarcodePropertyWrapper( String methodName, Character propertyValue )
    String getMethodName()
    Object getPropertyValue()
```

```java
public class AltovaBarcodeClassResolver

    Registers the class com.altova.extensions.barcode.proxy.zxing.QRCodeBean for the qrcode bean, additionally to the classes registered by the org.krysalis.barcode4j.DefaultBarcodeClassResolver.
```

---

**The com.altova.extensions.barcode.proxy.zxing package**

The package, com.altova.extensions.barcode.proxy.zxing, is used to generate the QRCode barcode type.

The following classes are used:
class QRCodeBean

- Extends org.krysaliis.barcode4j.impl.AbstractBarcodeBean
- Creates an AbstractBarcodeBean interface for com.google.zxing.qrcode.encoder

  void generateBarcode(CanvasProvider canvasImp, String msg)
  void setQRErrorCorrectionLevel(QRCodeErrorCorrectionLevel level)
  BarcodeDimension calcDimensions(String msg)
  double getVerticalQuietZone()
  double getBarWidth()

class QRCodeErrorCorrectionLevel Error correction level for the QRCode

  static QRCodeErrorCorrectionLevel byName(String name)
  "L" = ~7% correction
  "M" = ~15% correction
  "H" = ~25% correction
  "Q" = ~30% correction

---

**XSLT example**

Given below is an XSLT example showing how barcode functions are used in an XSLT stylesheet.

```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:altova="http://www.altova.com"
    xmlns:altovaext="http://www.altova.com/xslt-extensions"
    xmlns:altovaext-barcode="java:com.altova.extensions.barcode.BarcodeWrapper"
  <xsl:output method="html" encoding="UTF-8" indent="yes"/>
  <xsl:template match="/">
    <html>
      <head><title/></head>
      <body>
        <img alt="barcode" src="{altovaext:get-temp-folder()}barcode.png"/>
      </body>
    </html>
  </xsl:template>
  <xsl:result-document
    href="{altovaext:get-temp-folder()}barcode.png"
    method="text" encoding="base64tobinary">
    <xsl:variable name="barcodeObject"
      select="altovaext-barcode:newInstance('Code39',string('some value'),
        96,0, (altovaext-barcode-property:new( 'setModuleWidth',
        25.4 div 96 * 2 ) ) )"/>
    <xsl:value-of select="xs:base64Binary(xs:hexBinary(string(altovaext-barcode:generateBarcodePngAsHexString($barcodeObject)) ))"/>
  </xsl:result-document>
</xsl:stylesheet>
```
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. 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.
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

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.

**Note:** If you wish to add a namespace to an XSLT stylesheet being generated from an SPS created in StyleVision, the namespace must be added to the top-level schema element of the XML Schema on which the SPS is based. Note that the following namespace declaration `xmlns:java="java"` is created automatically by default in every SPS created in StyleVision.

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
java:classname
```

where

- `java:` indicates that a user-defined Java function is being called; (Java classes in the current directory will be loaded by default)
- `classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call.
Class file packaged, XSLT/XQuery file in same folder as Java package
The example below calls the getVehicleType() method of the Car class of the com.altova.extfunc package. The com.altova.extfunc package is in the folder JavaProject. The XSLT file is also in the folder JavaProject.

```xml
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:fn="http://www.w3.org/2005/xpath-functions"
    xmlns:car="java:com.altova.extfunc.Car" >
    <xsl:output exclude-result-prefixes="fn car xsl fo xs"/>
    <xsl:template match="/">
        <a>
            <xsl:value-of select="car:getVehicleType()"/>
        </a>
    </xsl:template>
</xsl:stylesheet>
```

Class file not packaged, XSLT/XQuery file in same folder as class file
The example below calls the getVehicleType() method of the Car class of the com.altova.extfunc package. The Car class file is in the following folder location: JavaProject/com/altova/extfunc. The XSLT file is also in the folder JavaProject/com/altova/extfunc.

```xml
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:fn="http://www.w3.org/2005/xpath-functions"
    xmlns:car="java:Car" >
    <xsl:output exclude-result-prefixes="fn car xsl fo xs"/>
    <xsl:template match="/">
        <a>
            <xsl:value-of select="car:getVehicleType()"/>
        </a>
    </xsl:template>
</xsl:stylesheet>
```
Class file packaged, XSLT/XQuery file at any location
The example below calls the `getCarColor()` method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is at any location. In this case, the location of the package must be specified within the URI as a query string. The syntax is:

```
java:classname[?path=uri-of-package]
```

where

- `java:` indicates that a user-defined Java function is being called
- `uri-of-package` is the URI of the Java package
- `classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call. The example below shows how to access a class file that is located in another directory than the current directory.

```xml
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:fn="http://www.w3.org/2005/xpath-functions"
    <xsl:output exclude-result-prefixes="fn car xsl xs"/>
    <xsl:template match="/">
        <xsl:variable name="myCar" select="car:new('red')" />
        <a><xsl:value-of select="car:getCarColor($myCar)" /></a>
    </xsl:template>
</xsl:stylesheet>
```

Class file not 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. The location of the class file is 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.

<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:nsl="java:docx.layout.pages?path=jar:file:///c:/projects/docs/docx.jar!/
    nsl: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:
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:
  ```xml
  <xsl:variable name="currentdate" select="date:new()"
    xmlns:date="java:java.util.Date" />
  ```

- It can be passed to an extension function (see Instance Method and Instance Fields):
  ```xml
  <xsl:value-of select="date:toString(date:new())"
    xmlns:date="java:java.util.Date" />
  ```

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() )))}">
        </enrollment>
    </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>xs:string</th>
<th>java.lang.String</th>
</tr>
</thead>
<tbody>
<tr>
<td>xs:boolean</td>
<td>boolean (primitive), java.lang.Boolean</td>
</tr>
<tr>
<td>xs:integer</td>
<td>int, long, short, byte, float, double, and the wrapper classes of these, such as java.lang.Integer</td>
</tr>
<tr>
<td>xs:float</td>
<td>float (primitive), java.lang.Float, double (primitive)</td>
</tr>
<tr>
<td>xs:double</td>
<td>double (primitive), java.lang.Double</td>
</tr>
<tr>
<td>xs:decimal</td>
<td>float (primitive), java.lang.Float, double (primitive), java.lang.Double</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).
.NET Extension Functions

If you are working on the .NET platform on a Windows machine, you can use extension functions written in any of the .NET languages (for example, C#). A .NET extension function can be used within an XPath or XQuery expression to invoke a constructor, property, or method (static or instance) within a .NET class.

A property of a .NET class is called using the syntax `get_PropertyName()`.

This section is organized into the following sub-sections:

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

---

Form of the extension function

The extension function in the XPath/XQuery expression must have the form `prefix:fname()`.

- The `prefix:` part is associated with a URI that identifies the .NET class being addressed.
- The `fname()` part identifies the constructor, property, or method (static or instance) within the .NET class, and supplies any argument/s, if required.
- The URI must begin with `clitype:` (which identifies the function as being a .NET extension function).
- The `prefix:fname()` form of the extension function can be used with system classes and with classes in a loaded assembly. However, if a class needs to be loaded, additional parameters containing the required information will have to be supplied.

---

Parameters

To load an assembly, the following parameters are used:

- **asm** The name of the assembly to be loaded.
- **ver** The version number (maximum of four integers separated by periods).
- **sn** The key token of the assembly's strong name (16 hex digits).
- **from** A URI that gives the location of the assembly (DLL) to be loaded. If the URI is relative, it is relative to the XSLT or XQuery document. If this parameter is present, any other parameter is ignored.
- **partialname** The partial name of the assembly. It is supplied to `Assembly.LoadWith.PartialName()`, which will attempt to load the assembly. If `partialname` is present, any other parameter is ignored.
- **loc** The locale, for example, en-US. The default is neutral.
If the assembly is to be loaded from a DLL, use the `from` parameter and omit the `sn` parameter. If the assembly is to be loaded from the Global Assembly Cache (GAC), use the `sn` parameter and omit the `from` parameter.

A question mark must be inserted before the first parameter, and parameters must be separated by a semi-colon. The parameter name gives its value with an equals sign (see example below).

### Examples of namespace declarations

An example of a namespace declaration in XSLT that identifies the system class `System.Environment`:

```xml
xmlns:myns="clitype:System.Environment"
```

An example of a namespace declaration in XSLT that identifies the class to be loaded as `Trade.Forward.Scrip`:

```xml
xmlns:myns="clitype:Trade.Forward.Scrip?asm=forward;version=10.6.2.1"
```

An example of a namespace declaration in XQuery that identifies the system class `MyManagedDLL.testClass`.

Two cases are distinguished:

1. When the assembly is loaded from the GAC:

   ```xml
   declare namespace cs="clitype:MyManagedDLL.testClass?asm=MyManagedDLL;ver=1.2.3.4;loc=neutral;sn=b9f091b72dce5ba8";
   ```

2. When the assembly is loaded from the DLL (complete and partial references below):

   ```xml
   declare namespace cs="clitype:MyManagedDLL.testClass?from=file://C:/Altova
   Projects/extFunctions/MyManagedDLL.dll;
   declare namespace cs="clitype:MyManagedDLL.testClass?
   from=MyManagedDLL.dll;
   ```

### XSLT example

Here is a complete XSLT example that calls functions in system class `System.Math`:

```xml
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:fn="http://www.w3.org/2005/xpath-functions">
    <xsl:output method="xml" omit-xml-declaration="yes" />
    <xsl:template match="/"
        <math xmlns:math="clitype:System.Math">
            <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:

```xml
<math xmlns:math="clitype:System.Math">
  {math:Sqrt(9)}
</math>
```

As with the XSLT example above, the namespace declaration identifies the .NET class, in this case a system class. The XQuery expression identifies the method to be called and supplies the argument.

**.NET: Constructors**

An extension function can be used to call a .NET constructor. All constructors are called with the pseudo-function `new()`. If there is more than one constructor for a class, then the constructor that most closely matches the number of arguments supplied is selected. If no constructor is deemed to match the supplied argument/s, then a 'No constructor found' error is returned.

**Constructors that return XPath/XQuery datatypes**
If the result of a .NET constructor call can be implicitly converted to XPath/XQuery datatypes, then the .NET extension function will return a sequence that is an XPath/XQuery datatype.

**Constructors that return .NET objects**
If the result of a .NET constructor call cannot be converted to a suitable XPath/XQuery datatype, then the constructor creates a wrapped .NET object with a type that is the name of the class returning that object. For example, if a constructor for the class `System.DateTime` is called (with `System.DateTime.new()`), then an object having a type `System.DateTime` is returned.

The lexical format of the returned object may not match the lexical format of a required XPath datatype. In such cases, the returned value would need to be: (i) converted to the lexical format of the required XPath datatype; and (ii) cast to the required XPath datatype.

There are three things that can be done with a .NET object created by a constructor:
It can be used within a variable:
<xs: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:
In the example above, a System.DateTime constructor (\texttt{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 \texttt{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 (\texttt{new(2008, 4, 29)}) as its first and only argument. In one of these instances, the variable \texttt{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 \texttt{releasedate} contains a .NET object, and it is this variable that is passed as the argument of \texttt{ToString()} in the second date element constructor. Therefore, the \texttt{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.

<table>
<thead>
<tr>
<th>XML Type</th>
<th>.NET Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xs:string</code></td>
<td>StringValue, string</td>
</tr>
<tr>
<td><code>xs:boolean</code></td>
<td>BooleanValue, bool</td>
</tr>
<tr>
<td><code>xs:integer</code></td>
<td>IntegerValue, decimal, long, integer, short, byte, double, float</td>
</tr>
<tr>
<td><code>xs:float</code></td>
<td>FloatValue, float, double</td>
</tr>
<tr>
<td><code>xs:double</code></td>
<td>DoubleValue, double</td>
</tr>
<tr>
<td><code>xs:decimal</code></td>
<td>DecimalValue, decimal, double, float</td>
</tr>
</tbody>
</table>

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the .NET type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct .NET method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error.

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, .NETs `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).
MSXSL Scripts for XSLT

The `<msxsl:script>` element contains user-defined functions and variables that can be called from within XPath expressions in the XSLT stylesheet. The `<msxsl:script>` is a top-level element, that is, it must be a child element of `<xsl:stylesheet>` or `<xsl:transform>`.

The `<msxsl:script>` element must be in the namespace `urn:schemas-microsoft-com:xslt` (see example below).

---

Scripting language and namespace

The scripting language used within the block is specified in the `<msxsl:script>` element's `language` attribute and the namespace to be used for function calls from XPath expressions is identified with the `implements-prefix` attribute (see below).

```
<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"
```
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.
22.3 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
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>adVarWChar</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:unsignedInteger</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>
### 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>
**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>
### 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>
## ODBC

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>
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:dateTime</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.
### 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>
22.4 Technical Data

This section contains useful background information on the technical aspects of your software. It is organized into the following sections:

- OS and Memory Requirements
- Altova XML Validator
- Altova XSLT and XQuery Engines
- Unicode Support
- Internet Usage
OS and Memory Requirements

Operating System
Altova software applications are available for the following platforms:


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. 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.
Altova XML Validator

When opening any XML document, the application uses its built-in XML validator to check for well-formedness, validate the document against a schema (if specified), and 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 specification. 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.
Altova 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 3.0 Engines. Documentation about implementation-specific behavior for each engine is in the appendices of the documentation (Engine Information), should that engine be used in the product.

Note: Altova MapForce generates code using the XSLT 1.0, 2.0 and XQuery 1.0 engines.
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 X-HTML 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 X-HTML 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.
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.
22.5 License Information

This section contains:

- Information about the distribution of this software product
- Information about software activation and license metering
- Information about the intellectual property rights related to this software product
- The End-User License Agreement governing the use of this software product

Please read this information carefully. It is binding upon you since you agreed to these terms when you installed this software product.
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 before making a purchasing decision.
- Once you decide to buy the software, you can place your order online at the Altova website and immediately 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 a comprehensive integrated onscreen help system. The latest version of the user manual is available at www.altova.com (i) in HTML format for online browsing, and (ii) in 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 this 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 have to purchase an Altova Software License Agreement, which is delivered in the form of a key-code that you enter into the Software Activation dialog to unlock the product. You can purchase your license at the online shop at the Altova website.

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 only distribute the Setup programs, provided that they are not modified in any way. Any person that 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.

For further details, please refer to the Altova Software License Agreement at the end of this section.
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 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 website (http://www.iana.org/) 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.

You will also notice that, if you are online, your Altova product contains many useful functions; these are unrelated to the license-metering technology.
Intellectual Property Rights

The Altova Software and any copies that you are authorized by Altova to make are the intellectual property of and are owned by Altova and its suppliers. The structure, organization and code of the Software are the valuable trade secrets and confidential information of Altova and its suppliers. The Software is protected by copyright, including without limitation by United States Copyright Law, international treaty provisions and applicable laws in the country in which it is being used. Altova retains the ownership of all patents, copyrights, trade secrets, trademarks and other intellectual property rights pertaining to the Software, and that Altova’s ownership rights extend to any images, photographs, animations, videos, audio, music, text and ”applets” incorporated into the Software and all accompanying printed materials. Notifications of claimed copyright infringement should be sent to Altova’s copyright agent as further provided on the Altova Web Site.

Altova software contains certain Third Party Software that is also protected by intellectual property laws, including without limitation applicable copyright laws as described in detail at http://www.altova.com/legal_3rdparty.html.

All other names or trademarks are the property of their respective owners.
Altova End User License Agreement

THIS IS A LEGAL DOCUMENT -- RETAIN FOR YOUR RECORDS

ALTOVA® END USER LICENSE AGREEMENT

Licensor:
Altova GmbH
Rudolfsplatz 13a/9
A-1010 Wien
Austria

Important - Read Carefully. Notice to User:

This End User License Agreement ("Agreement") is a legal document between you and Altova GmbH ("Altova"). It is important that you read this document before using the Altova-provided software ("Software") and any accompanying documentation, including, without limitation printed materials, ‘online’ files, or electronic documentation ("Documentation"). By clicking the "I accept" and "Next" buttons below, or by installing, or otherwise using the Software, you agree to be bound by the terms of this Agreement as well as the Altova Privacy Policy ("Privacy Policy") including, without limitation, the warranty disclaimers, limitation of liability, data use and termination provisions below, whether or not you decide to purchase the Software. You agree that this agreement is enforceable like any written agreement negotiated and signed by you. If you do not agree, you are not licensed to use the Software, and you must destroy any downloaded copies of the Software in your possession or control. You may print a copy of this Agreement as part of the installation process at the time of acceptance. Alternatively, a copy of this Agreement may be found at http://www.altova.com/eula and a copy of the Privacy Policy may be found at http://www.altova.com/privacy.

1. SOFTWARE LICENSE

(a) License Grant.

(i) Upon your acceptance of this Agreement Altova grants you a non-exclusive, non-transferable (except as provided below), limited license, without the right to grant sublicenses, to install and use a copy of the Software on one compatible personal computer or workstation up to the Permitted Number of computers. Subject to the limitations set forth in Section 1(c), you may install and use a copy of the Software on more than one of your compatible personal computers or workstations if you have purchased a Named-User license. Subject to the limitations set forth in Sections 1(d) and 1(e), users may use the software concurrently on a network. The Permitted Number of computers and/or users and the type of license, e.g. Installed, Named-Users, and Concurrent-User, shall be determined and specified at such time as you elect to purchase the Software. Installed user licenses are intended to be fixed and not concurrent. In other words, you cannot uninstall the Software on one machine in order to reinstall that license to a different machine and then uninstall and reinstall back to the original machine. Installations should be static. Notwithstanding the foregoing, permanent uninstallations and redeployments are acceptable in limited circumstances such as if an employee leaves the company or the machine is permanently decommissioned. During the evaluation period, hereinafter defined, only a single user may install and use the software on one (1) personal computer or workstation. If you have licensed the Software as part of a suite of Altova software products (collectively, the "Suite") and have not installed each product individually, then the Agreement governs your use of all of the software included in the Suite.
(ii) If you have licensed SchemaAgent, then the terms and conditions of this Agreement apply to your use of the SchemaAgent server software ("SchemaAgent Server") included therein, as applicable, and you are licensed to use SchemaAgent Server solely in connection with your use of Altova Software and solely for the purposes described in the accompanying documentation.

(iii) If you have licensed Software that enables users to generate source code, your license to install and use a copy of the Software as provided herein permits you to generate source code based on (i) Altova Library modules that are included in the Software (such generated code hereinafter referred to as the "Restricted Source Code") and (ii) schemas or mappings that you create or provide (such code as may be generated from your schema or mapping source materials hereinafter referred to as the "Unrestricted Source Code"). In addition to the rights granted herein, Altova grants you a non-exclusive, non-transferable, limited license to compile the complete generated code (comprised of the combination of the Restricted Source Code and the Unrestricted Source Code) into executable object code form, and to use, copy, distribute or license that executable. You may not distribute or redistribute, sublicense, sell, or transfer the Restricted Source Code to a third-party in the un-compiled form unless said third-party already has a license to the Restricted Source Code through their separate agreement with Altova. Notwithstanding anything to the contrary herein, you may not distribute, incorporate or combine with other software, or otherwise use the Altova Library modules or Restricted Source Code, or any Altova intellectual property embodied in or associated with the Altova Library modules or Restricted Source Code, in any manner that would subject the Restricted Source Code to the terms of a copyleft, free software or open source license that would require the Restricted Source Code or Altova Library modules source code to be disclosed in source code form. Notwithstanding anything to the contrary herein, you may not use the Software to develop and distribute other software programs that directly compete with any Altova software or service without prior written permission. Altova reserves all other rights in and to the Software. With respect to the feature(s) of UModel that permit reverse-engineering of your own source code or other source code that you have lawfully obtained, such use by you does not constitute a violation of this Agreement. Except as otherwise expressly permitted in Section 1(j) reverse engineering of the Software is strictly prohibited as further detailed therein.

(iv) In the event Restricted Source Code is incorporated into executable object code form, you will include the following statement in (1) introductory splash screens, or if none, within one or more screens readily accessible by the end-user, and (2) in the electronic and/or hard copy documentation: "Portions of this program were developed using Altova® [name of Altova Software, e.g. MapForce® 2011] and includes libraries owned by Altova GmbH, Copyright © 2007-2011 Altova GmbH (www.altova.com)."

(b) **Server Use for Installation and Use of SchemaAgent.** You may install one (1) copy of the Software on a computer file server within your internal network solely for the purpose of downloading and installing the Software onto other computers within your internal network up to the Permitted Number of computers in a commercial environment only. If you have licensed SchemaAgent, then you may install SchemaAgent Server on any server computer or workstation and use it in connection with your Software. No other network use is permitted, including without limitation using the Software either directly or through commands, data or instructions from or to a computer not part of your internal network, for Internet or Web-hosting services or by any user not licensed to use this copy of the Software through a valid license from Altova.

(c) **Named-Use.** If you have licensed the “Named-User” version of the software, you may install the Software on up to five (5) compatible personal computers or workstations of which you are the primary user thereby allowing you to switch from one computer to the other as necessary provided that only one (1) instance of the Software will be used by you as the Named-User at any
given time. If you have purchased multiple Named-User licenses, each individual Named-User will receive a separate license key code.

(d) Concurrent Use in Same Physical Network or Office Location. If you have licensed a “Concurrent-User” version of the Software, you may install the Software on any compatible computers in a commercial environment only, up to ten (10) times the Permitted Number of users, provided that only the Permitted Number of users actually use the Software at the same time and further provided that the computers on which the Software is installed are on the same physical computer network. The Permitted Number of concurrent users shall be delineated at such time as you elect to purchase the Software licenses. Each separate physical network or office location requires its own set of separate Concurrent User Licenses for those wishing to use the Concurrent User versions of the Software in more than one location or on more than one network, all subject to the above Permitted Number limitations and based on the number of users using the Software. If a computer is not on the same physical network, then a locally installed user license or a license dedicated to concurrent use in a virtual environment is required. Home User restrictions and limitations with respect to the Concurrent User licenses used on home computers are set forth in Section 1(g).

(e) Concurrent Use in Virtual Environment. If you have purchased Concurrent-User Licenses, you may install a copy of the Software on a terminal server (Microsoft Terminal Server or Citrix Metaframe), application virtualization server (Microsoft App-V, Citrix XenApp, or VMWare ThinApp) or virtual machine environment within your internal network for the sole and exclusive purpose of permitting individual users within your organization to access and use the Software through a terminal server, application virtualization session, or virtual machine environment from another computer provided that the total number of users that access or use the Software concurrently at any given point in time on such network, virtual machine or terminal server does not exceed the Permitted Number; and provided that the total number of users authorized to use the Software through the terminal server, application virtualization session, or virtual machine environment does not exceed ten (10) times the Permitted Number of users. In a virtual environment, you must deploy a reliable and accurate means of preventing users from exceeding the Permitted Number of concurrent users. Altova makes no warranties or representations about the performance of Altova software in a terminal server, application virtualization session, or virtual machine environment and the foregoing are expressly excluded from the limited warranty in Section 5 hereof. Technical support is not available with respect to issues arising from use in such environments.

(f) Backup and Archival Copies. You may make one (1) backup and one (1) archival copy of the Software, provided your backup and archival copies are not installed or used on any computer and further provided that all such copies shall bear the original and unmodified copyright, patent and other intellectual property markings that appear on or in the Software. You may not transfer the rights to a backup or archival copy unless you transfer all rights in the Software as provided under Section 3.

(g) Home Use (Personal and Non-Commercial). In order to further familiarize yourself with the Software and allow you to explore its features and functions, you, as the primary user of the computer on which the Software is installed for commercial purposes, may also install one copy of the Software on only one (1) home personal computer (such as your laptop or desktop) solely for your personal and non-commercial (“HPNC”) use. This HPNC copy may not be used in any commercial or revenue-generating business activities, including without limitation, work-from-home, teleworking, telecommuting, or other work-related use of the Software. The HPNC copy of the Software may not be used at the same time on a home personal computer as the Software is being used on the primary computer.
(h) **Key Codes, Upgrades and Updates.** Prior to your purchase and as part of the registration for the thirty (30) day evaluation period, as applicable, you will receive an evaluation key code. You will receive a purchase key code when you elect to purchase the Software from either Altova GmbH or an authorized reseller. The purchase key code will enable you to activate the Software beyond the initial evaluation period. You may not re-license, reproduce or distribute any key code except with the express written permission of Altova. If the Software that you have licensed is an upgrade or an update, then the latest update or upgrade that you download and install replaces all or part of the Software previously licensed. The update or upgrade and the associated license keys does not constitute the granting of a second license to the Software in that you may not use the upgrade or updated copy in addition to the copy of the Software that it is replacing and whose license has terminated.

(i) **Title.** Title to the Software is not transferred to you. Ownership of all copies of the Software and of copies made by you is vested in Altova, subject to the rights of use granted to you in this Agreement. As between you and Altova, documents, files, stylesheets, generated program code (including the Unrestricted Source Code) and schemas that are authored or created by you via your utilization of the Software, in accordance with its Documentation and the terms of this Agreement, are your property unless they are created using Evaluation Software, as defined in Section 4 of this Agreement, in which case you have only a limited license to use any output that contains generated program code (including Unrestricted Source Code) such as Java, C++, C#, VB.NET or XSLT and associated project files and build scripts, as well as generated XML, XML Schemas, documentation, UML diagrams, and database structures only for the thirty (30) day evaluation period.

(j) **Reverse Engineering.** Except and to the limited extent as may be otherwise specifically provided by applicable law in the European Union, you may not reverse engineer, decompile, disassemble or otherwise attempt to discover the source code, underlying ideas, underlying user interface techniques or algorithms of the Software by any means whatsoever, directly or indirectly, or disclose any of the foregoing, except to the extent you may be expressly permitted to decompile under applicable law in the European Union, if it is essential to do so in order to achieve operability of the Software with another software program, and you have first requested Altova to provide the information necessary to achieve such operability and Altova has not made such information available. Altova has the right to impose reasonable conditions and to request a reasonable fee before providing such information. Any information supplied by Altova or obtained by you, as permitted hereunder, may only be used by you for the purpose described herein and may not be disclosed to any third party or used to create any software which is substantially similar to the expression of the Software. Requests for information from users in the European Union with respect to the above should be directed to the Altova Customer Support Department.

(k) **Other Restrictions.** You may not loan, rent, lease, sublicense, distribute or otherwise transfer all or any portion of the Software to third parties except to the limited extent set forth in Section 3 or as otherwise expressly provided. You may not copy the Software except as expressly set forth above, and any copies that you are permitted to make pursuant to this Agreement must contain the same copyright, patent and other intellectual property markings that appear on or in the Software. You may not modify, adapt or translate the Software. You may not, directly or indirectly, encumber or suffer to exist any lien or security interest on the Software; knowingly take any action that would cause the Software to be placed in the public domain; or use the Software in any computer environment not specified in this Agreement. You may not permit any use of or access to the Software by any third party in connection with a commercial service offering, such as for a cloud-based or web-based SaaS offering.

You will comply with applicable law and Altova’s instructions regarding the use of the
Software. You agree to notify your employees and agents who may have access to the Software of the restrictions contained in this Agreement and to ensure their compliance with these restrictions.

(i) NO GUARANTEE. THE SOFTWARE IS NEITHER GUARANTEED NOR WARRANTED TO BE ERROR-FREE NOR SHALL ANY LIABILITY BE ASSUMED BY ALTOMA IN THIS RESPECT. NOTWITHSTANDING ANY SUPPORT FOR ANY TECHNICAL STANDARD, THE SOFTWARE IS NOT INTENDED FOR USE IN OR IN CONNECTION WITH, WITHOUT LIMITATION, THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT NAVIGATION, COMMUNICATION SYSTEMS, AIR TRAFFIC CONTROL EQUIPMENT, MEDICAL DEVICES OR LIFE SUPPORT SYSTEMS, MEDICAL OR HEALTH CARE APPLICATIONS, OR OTHER APPLICATIONS WHERE THE FAILURE OF THE SOFTWARE OR ERRORS IN DATA PROCESSING COULD LEAD TO DEATH, PERSONAL INJURY OR SEVERE PHYSICAL OR ENVIRONMENTAL DAMAGE. YOU AGREE THAT YOU ARE SOLELY RESPONSIBLE FOR THE ACCURACY AND ADEQUACY OF THE SOFTWARE AND ANY DATA GENERATED OR PROCESSED BY THE SOFTWARE FOR YOUR INTENDED USE AND YOU WILL DEFEND, INDEMNIFY AND HOLD ALTOMA, ITS OFFICERS AND EMPLOYEES HARMLESS FROM ANY THIRD PARTY CLAIMS, DEMANDS, OR SUITS THAT ARE BASED UPON THE ACCURACY AND ADEQUACY OF THE SOFTWARE IN YOUR USE OR ANY DATA GENERATED BY THE SOFTWARE IN YOUR USE.

2. INTELLECTUAL PROPERTY RIGHTS

You acknowledge that the Software and any copies that you are authorized by Altova to make are the intellectual property of and are owned by Altova and its suppliers. The structure, organization and code of the Software are the valuable trade secrets and confidential information of Altova and its suppliers. The Software is protected by copyright, including without limitation by United States Copyright Law, international treaty provisions and applicable laws in the country in which it is being used. You acknowledge that Altova retains the ownership of all patents, copyrights, trade secrets, trademarks and other intellectual property rights pertaining to the Software, and that Altova’s ownership rights extend to any images, photographs, animations, videos, audio, music, text and “applets” incorporated into the Software and all accompanying printed materials. You will take no actions which adversely affect Altova’s intellectual property rights in the Software. Trademarks shall be used in accordance with accepted trademark practice, including identification of trademark owners’ names. Trademarks may only be used to identify printed output produced by the Software, and such use of any trademark does not give you any right of ownership in that trademark. Altova®, XMLSpy®, Authentic®, StyleVision®, MapForce®, UModel®, DatabaseSpy®, DiffDog®, SchemaAgent®, SemanticWorks®, MissionKit®, Markup Your Mind®, Nanonull™, RaptorXML™, RaptorXML Server™, RaptorXML +XBRL Server™, Powered By RaptorXML™, FlowForce Server™, StyleVision Server™, and MapForce Server™ are trademarks of Altova GmbH. (pending or registered in numerous countries). Unicode and the Unicode Logo are trademarks of Unicode, Inc. Windows, Windows XP, Windows Vista, Windows 7, and Windows 8 are trademarks of Microsoft. W3C, CSS, DOM, MathML, RDF, XHTML, XML and XSL are trademarks (registered in numerous countries) of the World Wide Web Consortium (W3C); marks of the W3C are registered and held by its host institutions, MIT, INRIA and Keio. Except as expressly stated above, this Agreement does not grant you any intellectual property rights in the Software. Notifications of claimed copyright infringement should be sent to Altova’s copyright agent as further provided on the Altova Web Site.

3. LIMITED TRANSFER RIGHTS

Notwithstanding the foregoing, you may transfer all your rights to use the Software to another person or legal entity provided that: (a) you also transfer this Agreement, the Software and all
other software or hardware bundled or pre-installed with the Software, including all copies, updates and prior versions, and all copies of font software converted into other formats, to such person or entity; (b) you retain no copies, including backups and copies stored on a computer; (c) the receiving party secures a personalized key code from Altova; and (d) the receiving party accepts the terms and conditions of this Agreement and any other terms and conditions upon which you legally purchased a license to the Software. Notwithstanding the foregoing, you may not transfer education, pre-release, or not-for-resale copies of the Software.

4. PRE-RELEASE AND EVALUATION PRODUCT ADDITIONAL TERMS

If the product you have received with this license is pre-commercial release or beta Software ("Pre-release Software"), then this Section applies. In addition, this section applies to all evaluation and/or demonstration copies of Altova software ("Evaluation Software") and continues in effect until you purchase a license. To the extent that any provision in this section is in conflict with any other term or condition in this Agreement, this section shall supersede such other term(s) and condition(s) with respect to the Pre-release and/or Evaluation Software, but only to the extent necessary to resolve the conflict. You acknowledge that the Pre-release Software is a pre-release version, does not represent final product from Altova, and may contain bugs, errors and other problems that could cause system or other failures and data loss. CONSEQUENTLY, THE PRE-RELEASE AND/OR EVALUATION SOFTWARE IS PROVIDED TO YOU "AS-IS" WITH NO WARRANTIES FOR USE OR PERFORMANCE, AND ALTOVA DISCLAIMS ANY WARRANTY OR LIABILITY OBLIGATIONS TO YOU OF ANY KIND, WHETHER EXPRESS OR IMPLIED. WHERE LEGALLY LIABILITY CANNOT BE EXCLUDED FOR PRE-RELEASE AND/OR EVALUATION SOFTWARE, BUT IT MAY BE LIMITED, ALTOVA’S LIABILITY AND THAT OF ITS SUPPLIERS SHALL BE LIMITED TO THE SUM OF FIFTY DOLLARS (USD $50) IN TOTAL. If the Evaluation Software has a time-out feature, then the software will cease operation after the conclusion of the designated evaluation period. Upon such expiration date, your license will expire unless otherwise extended. Your license to use any output created with the Evaluation Software that contains generated program code (including Unrestricted Source Code) such as Java, C++, C, VB.NET or XSLT and associated project files and build scripts as well as generated XML, XML Schemas, documentation, UML diagrams, and database structures terminates automatically upon the expiration of the designated evaluation period but the license to use such output is revived upon your purchase of a license for the Software that you evaluated and used to create such output. Access to any files created with the Evaluation Software is entirely at your risk. You acknowledge that Altova has not promised or guaranteed to you that Pre-release Software will be announced or made available to anyone in the future, that Altova has no express or implied obligation to you to announce or introduce the Pre-release Software, and that Altova may not introduce a product similar to or compatible with the Pre-release Software. Accordingly, you acknowledge that any research or development that you perform regarding the Pre-release Software or any product associated with the Pre-release Software is done entirely at your own risk. During the term of this Agreement, if requested by Altova, you will provide feedback to Altova regarding testing and use of the Pre-release Software, including error or bug reports. If you have been provided the Pre-release Software pursuant to a separate written agreement, your use of the Software is governed by such agreement. You may not sublicense, lease, loan, rent, distribute or otherwise transfer the Pre-release Software. Upon receipt of a later unreleased version of the Pre-release Software or release by Altova of a publicly released commercial version of the Software, whether as a stand-alone product or as part of a larger product, you agree to return or destroy all earlier Pre-release Software received from Altova and to abide by the terms of the license agreement for any such later versions of the Pre-release Software.

5. LIMITED WARRANTY AND LIMITATION OF LIABILITY

(a) Limited Warranty and Customer Remedies. Altova warrants to the person or entity
that first purchases a license for use of the Software pursuant to the terms of this Agreement that
(i) the Software will perform substantially in accordance with any accompanying Documentation
for a period of ninety (90) days from the date of receipt, and (ii) any support services provided by
Altova shall be substantially as described in Section 6 of this agreement. Some states and
jurisdictions do not allow limitations on duration of an implied warranty, so the above limitation
may not apply to you. To the extent allowed by applicable law, implied warranties on the
Software, if any, are limited to ninety (90) days. Altova's and its suppliers' entire liability and your
exclusive remedy shall be, at Altova's option, either (i) return of the price paid, if any, or (ii) repair
or replacement of the Software that does not meet Altova's Limited Warranty and which is
returned to Altova with a copy of your receipt. This Limited Warranty is void if failure of the
Software has resulted from accident, abuse, misapplication, abnormal use, Trojan horse, virus, or
any other malicious external code. Any replacement Software will be warranted for the remainder
of the original warranty period or thirty (30) days, whichever is longer. This limited warranty does
not apply to Evaluation and/or Pre-release Software.

(b) **No Other Warranties and Disclaimer.** THE FOREGOING LIMITED WARRANTY AND
REMEDIES STATE THE SOLE AND EXCLUSIVE REMEDIES FOR ALTova OR ITS
SUPPLIER'S BREACH OF WARRANTY. ALTova AND ITS SUPPLIERS DO NOT AND CANNOT
WARRANT THE PERFORMANCE OR RESULTS YOU MAY OBTAIN BY USING THE
SOFTWARE. EXCEPT FOR THE FOREGOING LIMITED WARRANTY, AND FOR ANY
WARRANTY, CONDITION, REPRESENTATION OR TERM TO THE EXTENT WHICH THE SAME
CANNOT OR MAY NOT BE EXCLUDED OR LIMITED BY LAW APPLICABLE TO YOU IN YOUR
JURISDICTION, ALTova AND ITS SUPPLIERS MAKE NO WARRANTIES, CONDITIONS,
REPRESENTATIONS OR TERMS, EXPRESS OR IMPLIED, WHETHER BY STATUTE,
COMMON LAW, CUSTOM, USAGE OR OTHERWISE AS TO ANY OTHER MATTERS. TO THE
MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, ALTova AND ITS SUPPLIERS
DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED,
INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, INFORMATIONAL
CONTENT OR ACCURACY, QUIET ENJOYMENT, TITLE AND NON-INFRINGEMENT, WITH REGARD TO THE
SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES.
THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHERS,
WHICH VARY FROM STATE/JURISDICTION TO STATE/JURISDICTION.

(c) **Limitation of Liability.** TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE
LAW EVEN IF A REMEDY FAILS ITS ESSENTIAL PURPOSE, IN NO EVENT SHALL ALTova
OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, DIRECT, INDIRECT OR
CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES
FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS
INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OR
INABILITY TO USE THE SOFTWARE OR THE PROVISION OF OR FAILURE TO PROVIDE
SUPPORT SERVICES, EVEN IF ALTova HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGES. IN ANY CASE, ALTova'S ENTIRE LIABILITY UNDER ANY PROVISION OF
THIS AGREEMENT SHALL BE LIMITED TO THE AMOUNT ACTUALLY PAID BY YOU FOR THE
SOFTWARE PRODUCT. Because some states and jurisdictions do not allow the exclusion or
limitation of liability, the above limitation may not apply to you. In such states and jurisdictions,
Altova's liability shall be limited to the greatest extent permitted by law and the limitations or
exclusions of warranties and liability contained herein do not prejudice applicable statutory
consumer rights of person acquiring goods otherwise than in the course of business. The
disclaimer and limited liability above are fundamental to this Agreement between Altova and you.

(d) **Infringement Claims.** Altova will indemnify and hold you harmless and will defend or
settle any claim, suit or proceeding brought against you by a third party that is based upon a
claim that the content contained in the Software infringes a copyright or violates an intellectual or proprietary right protected by United States or European Union law ("Claim"), but only to the extent the Claim arises directly out of the use of the Software and subject to the limitations set forth in Section 5 of this Agreement except as otherwise expressly provided. You must notify Altova in writing of any Claim within ten (10) business days after you first receive notice of the Claim, and you shall provide to Altova at no cost such assistance and cooperation as Altova may reasonably request from time to time in connection with the defense of the Claim. Altova shall have sole control over any Claim (including, without limitation, the selection of counsel and the right to settle on your behalf on any terms Altova deems desirable in the sole exercise of its discretion). You may, at your sole cost, retain separate counsel and participate in the defense or settlement negotiations. Altova shall pay actual damages, costs, and attorney fees awarded against you (or payable by you pursuant to a settlement agreement) in connection with a Claim to the extent such direct damages and costs are not reimbursed to you by insurance or a third party, to an aggregate maximum equal to the purchase price of the Software. If the Software or its use becomes the subject of a Claim or its use is enjoined, or if in the opinion of Altova’s legal counsel the Software is likely to become the subject of a Claim, Altova shall attempt to resolve the Claim by using commercially reasonable efforts to modify the Software or obtain a license to continue using the Software. If in the opinion of Altova’s legal counsel the Claim, the injunction or potential Claim cannot be resolved through reasonable modification or licensing, Altova, at its own election, may terminate this Agreement without penalty, and will refund to you on a pro rata basis any fees paid in advance by you to Altova. THE FOREGOING CONSTITUTES ALTOVA’S SOLE AND EXCLUSIVE LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT. This indemnity does not apply to situations where the alleged infringement, whether patent or otherwise, is the result of a combination of the Altova software and additional elements supplied by you.

6. SUPPORT AND MAINTENANCE

Altova offers multiple optional “Support & Maintenance Package(s)” ("SMP") for the version of Software product edition that you have licensed, which you may elect to purchase in addition to your Software license. The Support Period, hereinafter defined, covered by such SMP shall be delineated at such time as you elect to purchase a SMP. Your rights with respect to support and maintenance as well as your upgrade eligibility depend on your decision to purchase SMP and the level of SMP that you have purchased:

(a) If you have not purchased SMP, you will receive the Software AS IS and will not receive any maintenance releases or updates. However, Altova, at its option and in its sole discretion on a case by case basis, may decide to offer maintenance releases to you as a courtesy, but these maintenance releases will not include any new features in excess of the feature set at the time of your purchase of the Software. In addition, Altova will provide free technical support to you for thirty (30) days after the date of your purchase (the "Support Period" for the purposes of this paragraph 6(a), and Altova, in its sole discretion on a case by case basis, may also provide free courtesy technical support during your thirty (30) day evaluation period. Technical support is provided via a Web-based support form only, and there is no guaranteed response time.

(b) If you have purchased SMP, then solely for the duration of its delineated Support Period, you are eligible to receive the version of the Software edition that you have licensed and all maintenance releases and updates for that edition that are released during your Support Period. For the duration of your SMP’s Support Period, you will also be eligible to receive upgrades to the comparable edition of the next version of the Software that succeeds the Software edition that you have licensed for applicable upgrades released during your Support Period. The specific upgrade edition that you are eligible to receive based on your Support Period is further detailed in the SMP that you have purchased. Software that is introduced as separate product is not included in SMP. Maintenance releases, updates and upgrades may or may not include additional features. In
addition, Altova will provide Priority Technical Support to you for the duration of the Support Period. Priority Technical Support is provided via a Web-based support form only and Altova will make commercially reasonable efforts to respond via e-mail to all requests within forty-eight (48) hours during Altova’s business hours (MO-FR, 8am UTC – 10pm UTC, Austrian and US holidays excluded) and to make reasonable efforts to provide work-arounds to errors reported in the Software.

During the Support Period you may also report any Software problem or error to Altova. If Altova determines that a reported reproducible material error in the Software exists and significantly impairs the usability and utility of the Software, Altova agrees to use reasonable commercial efforts to correct or provide a usable work-around solution in an upcoming maintenance release or update, which is made available at certain times at Altova’s sole discretion.

If Altova, in its discretion, requests written verification of an error or malfunction discovered by you or requests supporting example files that exhibit the Software problem, you shall promptly provide such verification or files, by email, telecopy, or overnight mail, setting forth in reasonable detail the respects in which the Software fails to perform. You shall use reasonable efforts to cooperate in diagnosis or study of errors. Altova may include error corrections in maintenance releases, updates, or new major releases of the Software. Altova is not obligated to fix errors that are immaterial. Immaterial errors are those that do not significantly impact use of the Software as determined by Altova in its sole discretion. Whether or not you have purchased the Support & Maintenance Package, technical support only covers issues or questions resulting directly out of the operation of the Software and Altova will not provide you with generic consultation, assistance, or advice under any circumstances.

Updating Software may require the updating of software not covered by this Agreement before installation. Updates of the operating system and application software not specifically covered by this Agreement are your responsibility and will not be provided by Altova under this Agreement. Altova’s obligations under this Section 6 are contingent upon your proper use of the Software and your compliance with the terms and conditions of this Agreement at all times. Altova shall be under no obligation to provide the above technical support if, in Altova’s opinion, the Software has failed due to the following conditions: (i) damage caused by the relocation of the Software to another location or CPU; (ii) alterations, modifications or attempts to change the Software without Altova’s written approval; (iii) causes external to the Software, such as natural disasters, the failure or fluctuation of electrical power, or computer equipment failure; (iv) your failure to maintain the Software at Altova’s specified release level; or (v) use of the Software with other software without Altova’s prior written approval. It will be your sole responsibility to: (i) comply with all Altova-specified operating and troubleshooting procedures and then notify Altova immediately of Software malfunction and provide Altova with complete information thereof; (ii) provide for the security of your confidential information; (iii) establish and maintain backup systems and procedures necessary to reconstruct lost or altered files, data or programs.

7. SOFTWARE ACTIVATION, UPDATES AND LICENSE METERING

(a) License Metering. The Software includes a built-in license metering module that is designed to assist you with monitoring license compliance in small local networks. The metering module attempts to communicate with other machines on your local area network. You permit Altova to use your internal network for license monitoring for this purpose. This license metering module may be used to assist with your license compliance but should not be the sole method. Should your firewall settings block said communications, you must deploy an accurate means of monitoring usage by the end user and preventing users from using the Software more than the Permitted Number.
(b) **License Compliance Monitoring.** You are required to utilize a process or tool to ensure that the Permitted Number is not exceeded. Without prejudice or waiver of any potential violations of the Agreement, Altova may provide you with additional compliance tools should you be unable to accurately account for license usage within your organization. If provided with such a tool by Altova, you (a) are required to use it in order to comply with the terms of this Agreement and (b) permit Altova to use your internal network for license monitoring and metering and to generate compliance reports that are communicated to Altova from time to time.

(c) **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 Master 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 between your computer and the Altova Master License Server. You agree that Altova may use these measures and you agree to follow any applicable requirements. You further agree that use of license key codes that are not or were not generated by Altova and lawfully obtained from Altova, or an authorized reseller as part of an effort to activate or use the Software violates Altova’s intellectual property rights as well as the terms of this Agreement. You agree that efforts to circumvent or disable Altova’s copyright protection mechanisms, the license management mechanism, or the Altova Master License Server violate Altova’s intellectual property rights as well as the terms of this Agreement. Altova expressly reserves the rights to seek all available legal and equitable remedies to prevent such actions and to recover lost profits, damages and costs.

(d) **LiveUpdate.** Altova provides a new LiveUpdate notification service to you, which is free of charge. Altova may use your internal network and Internet connection for the purpose of transmitting license-related data to an Altova-operated LiveUpdate server to validate your license at appropriate intervals and determine if there is any update available for you.

(e) **Use of Data.** The terms and conditions of the Privacy Policy are set out in full at [http://www.altova.com/privacy](http://www.altova.com/privacy) and are incorporated by reference into this Agreement. By your acceptance of the terms of this Agreement and/or use of the Software, you authorize the collection, use and disclosure of information collected by Altova for the purposes provided for in this Agreement and/or the Privacy Policy. Altova has the right in its sole discretion to amend this provision of the Agreement and/or Privacy Policy at any time. You are encouraged to review the terms of the Privacy Policy as posted on the Altova Web site from time to time.

(f) **Audit Rights.** You agree that Altova may audit your use of the Software for compliance with the terms of this Agreement at any time, upon reasonable notice. In the event that such audit reveals any use of the Software by you other than in full compliance with the terms of this Agreement, you shall reimburse Altova for all reasonable expenses related to such audit in addition to any other liabilities you may incur as a result of such non-compliance.

(g) **Notice to European Users.** Please note that the information as described in paragraph 7(d) above may be transferred outside of the European Economic Area, for purposes of processing, analysis, and review, by Altova, Inc., a company located in Beverly, Massachusetts, U.S.A., or its subsidiaries or Altova’s subsidiaries or divisions, or authorized partners, located worldwide. You are advised that the United States uses a sectoral model of privacy protection that relies on a mix of legislation, governmental regulation, and self-regulation. You are further advised that the Council of the European Union has found that this model does not provide "adequate" privacy protections as contemplated by Article 25 of the European Union's Data Directive. (Directive 95/46/EC, 1995 O.J. (L 281) 31). Article 26 of the European Union's Data Directive
allows for transfer of personal data from the European Union to a third country if the individual has unambiguously given his consent to the transfer of personal information, regardless of the third country’s level of protection. By agreeing to this Agreement, you consent to the transfer of all such information to the United States and the processing of that information as described in this Agreement and the Privacy Policy.

8. TERM AND TERMINATION

This Agreement may be terminated (a) by your giving Altova written notice of termination; (b) by Altova, at its option, giving you written notice of termination if you commit a breach of this Agreement and fail to cure such breach within ten (10) days after notice from Altova; or (c) at the request of an authorized Altova reseller in the event that you fail to make your license payment or other monies due and payable. In addition the Agreement governing your use of a previous version of the Software that you have upgraded or updated is terminated upon your acceptance of the terms and conditions of the Agreement accompanying such upgrade or update. Upon any termination of the Agreement, you must cease all use of the Software that this Agreement governs, destroy all copies then in your possession or control and take such other actions as Altova may reasonably request to ensure that no copies of the Software remain in your possession or control. The terms and conditions set forth in Sections 1(h), 1(i), 1(j), 1(k), 1(l), 2, 5, 7, 9, 10, 11, and 11 survive termination as applicable.

9. RESTRICTED RIGHTS NOTICE AND EXPORT RESTRICTIONS

The Software was developed entirely at private expense and is commercial computer software provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the U.S. Government or a U.S. Government contractor or subcontractor is subject to the restrictions set forth in this Agreement and as provided in FAR 12.211 and 12.212 (48 C.F.R. §12.211 and 12.212) or DFARS 227. 7202 (48 C.F.R. §227-7202) as applicable. Consistent with the above as applicable, Commercial Computer Software and Commercial Computer Documentation licensed to U.S. government end users only as commercial items and only with those rights as are granted to all other end users under the terms and conditions set forth in this Agreement. Manufacturer is Altova GmbH, Rudolfsplatz 13a/9, A-1010 Vienna, Austria/EU. You may not use or otherwise export or re-export the Software or Documentation except as authorized by United States law and the laws of the jurisdiction in which the Software was obtained. In particular, but without limitation, the Software or Documentation may not be exported or re-exported (i) into (or to a national or resident of) any U.S. embargoed country or (ii) to anyone on the U.S. Treasury Department's list of Specially Designated Nationals or the U.S. Department of Commerce's Table of Denial Orders. By using the Software, you represent and warrant that you are not located in, under control of, or a national or resident of any such country or on any such list.

10. U.S. GOVERNMENT ENTITIES

Notwithstanding the foregoing, if you are an agency, instrumentality or department of the federal government of the United States, then this Agreement shall be governed in accordance with the laws of the United States of America, and in the absence of applicable federal law, the laws of the Commonwealth of Massachusetts will apply. Further, and notwithstanding anything to the contrary in this Agreement (including but not limited to Section 5 (Indemnification)), all claims, demands, complaints and disputes will be subject to the Contract Disputes Act (41 U.S.C. §§7101 et seq.), the Tucker Act (28 U.S.C. §1346(a) and §1491), or the Federal Tort Claims Act (28 U.S.C. §§1346(b), 2401-2402, 2671-2672, 2674-2680), FAR 1.601(a) and 43.102 (Contract Modifications); FAR 12.302(b), as applicable, or other applicable governing authority. For the avoidance of doubt, if you are an agency, instrumentality, or department of the federal, state or local government of the U.S. or a U.S. public and accredited educational institution, then your
indemnification obligations are only applicable to the extent they would not cause you to violate any applicable law (e.g., the Anti-Deficiency Act), and you have any legally required authorization or authorizing statute.

11. THIRD PARTY SOFTWARE

The Software may contain third party software which requires notices and/or additional terms and conditions. Such required third party software notices and/or additional terms and conditions are located at our Website at http://www.altova.com/legal_3rdparty.html and are made a part of and incorporated by reference into this Agreement. By accepting this Agreement, you are also accepting the additional terms and conditions, if any, set forth therein.

12. JURISDICTION, CHOICE OF LAW, AND VENUE

If you are located in the European Union and are using the Software in the European Union and not in the United States, then this Agreement will be governed by and construed in accordance with the laws of the Republic of Austria (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the Handelsgericht, Wien (Commercial Court, Vienna) and you further agree and expressly consent to the exercise of personal jurisdiction in the Handelsgericht, Wien (Commercial Court, Vienna) in connection with any such dispute or claim.

If you are located in the United States or are using the Software in the United States then this Agreement will be governed by and construed in accordance with the laws of the Commonwealth of Massachusetts, USA (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the federal or state courts of the Commonwealth of Massachusetts and you further agree and expressly consent to the exercise of personal jurisdiction in the federal or state courts of the Commonwealth of Massachusetts in connection with any such dispute or claim.

If you are located outside of the European Union or the United States and are not using the Software in the United States, then this Agreement will be governed by and construed in accordance with the laws of the Republic of Austria (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the Handelsgericht, Wien (Commercial Court, Vienna) and you further agree and expressly consent to the exercise of personal jurisdiction in the Handelsgericht Wien (Commercial Court, Vienna) in connection with any such dispute or claim. This Agreement will not be governed by the conflict of law rules of any jurisdiction or the United Nations Convention on Contracts for the International Sale of Goods, the application of which is expressly excluded.

13. TRANSLATIONS

Where Altova has provided you with a foreign translation of the English language version, you agree that the translation is provided for your convenience only and that the English language version will control. If there is any contradiction between the English language version and a translation, then the English language version shall take precedence.

14. GENERAL PROVISIONS

This Agreement contains the entire agreement and understanding of the parties with respect to
the subject matter hereof, and supersedes all prior written and oral understandings of the parties with respect to the subject matter hereof. Any notice or other communication given under this Agreement shall be in writing and shall have been properly given by either of us to the other if sent by certified or registered mail, return receipt requested, or by overnight courier to the address shown on Altova’s Web site for Altova and the address shown in Altova’s records for you, or such other address as the parties may designate by notice given in the manner set forth above. This Agreement will bind and inure to the benefit of the parties and our respective heirs, personal and legal representatives, affiliates, successors and permitted assigns. The failure of either of us at any time to require performance of any provision hereof shall in no manner affect such party’s right at a later time to enforce the same or any other term of this Agreement. This Agreement may be amended only by a document in writing signed by both of us. In the event of a breach or threatened breach of this Agreement by either party, the other shall have all applicable equitable as well as legal remedies. Each party is duly authorized and empowered to enter into and perform this Agreement. If, for any reason, any provision of this Agreement is held invalid or otherwise unenforceable, such invalidity or unenforceability shall not affect the remainder of this Agreement, and this Agreement shall continue in full force and effect to the fullest extent allowed by law. The parties knowingly and expressly consent to the foregoing terms and conditions.

Last updated: 2013-10-17
Index

.docx (Enterprise Edition only), 24, 46
.NET,
  differences to StyleVision standalone, 956
  integration of StyleVision with, 954
.NET extension functions,
  constructors, 1518
  datatype conversions, .NET to XPath/XQuery, 1521
  datatype conversions, XPath/XQuery to .NET, 1520
  for XSLT and XQuery, 1516
  in XPath expressions, 978, 980
  instance methods, instance fields, 1519
  overview, 1516
  static methods, static fields, 1519
  support for, in Authentic View, 978, 980

A
Abbreviations,
  used in user manual, 6
About StyleVision, 1180
Activating the software, 1178
Active configuration, 1162
Add Active and Related Files to Project, 1055
Add Active File to Project, 1055
Add Altova Resource to Project, 1053
Add External Folder / Web Folder to Project, 1056
Add Files to Project, 1053
Add name, 623
Add Project Folder to Project, 1056
Add URL to Project, 1053
Adding schema, 1005
Additional editing procedures, 598
Additional Validation, 628
Alias,
  see Global Resources, 599
Aligning table cell content,
  in SPSs, 1128
Alternative processing, 625
Altova Engines,
  in Altova products, 1537
Altova extensions,
  chart functions (see chart functions), 1441
Altova Global Resources,
  see under Global Resources, 599
Altova website, 1180
Altova XML Parser,
  about, 1536
AltovaRowStatus,
  in DB-based SPS, 756
AltovaXML,
  and FOP, 947
Ambiguity,
  of content model, 153
API,
  accessing, 1406
  documentation, 1207
  overview, 1209
Append,
  column to table in SPS, 1121
  row to table in SPS, 1121
Appendices, 1428
Application,
  ActiveDocument, 1234
  Application, 1235
  Documents, 1235
  Parent, 1237
  Quit, 1237
Application Events, 1202
Application-level,
  integration of StyleVision, 1344
area chart features, 233
Area charts, 278
ASP.NET application, 695
ASPX web application, 695
Assign predefined formats,
  in Quick Start tutorial, 95
Attributes entry helper,
  in Authentic View, 886
Authentic Browser, 36
Authentic Desktop, 36
Authentic menu, 1129
  dynamic table editing, 881
  markup display, 881
Authentic node properties, 623
Authentic Scripting, 926, 929
  macros in, 930
Authentic toolbar, 1131

Authentic View, 901
and SPS, 129
and standard industry schemas, 129
and Working XML File, 44
context menus, 890
description of, 44
document creation process, 129
document display, 883
editing data in an XML DB, 1139
entry helpers in, 886
formatting text in, 881
in Altova products, 36
main window in, 883
markup display in, 881, 883
overview of GUI, 880
paste as XML/Text, 890
save edits, 1027
SPS Tables, 899
synchronizing with new version of StyleVision, 131
tables (SPS and XML), 898
toolbar buttons for, 994
toolbar icons, 881
usage of important features, 892
usage of XML tables, 901
XML table icons, 905
XML tables, 901

Authentic View Events, 1202

Authentic XML, 878

AuthenticRange,
AppendRow, 1254
Application, 1255
CanPerformAction, 1255
CanPerformActionWith, 1255
Close, 1256
CollapseToBegin, 1256
CollapseToEnd, 1256
Copy, 1257
Cut, 1257
Delete, 1257
DeleteRow, 1257
DuplicateRow, 1258
ExpandTo, 1259
FirstCursorPosition, 1259
FirstXMLData, 1260
FirstXMLDataOffset, 1261
GetElementAttributeNames, 1262
GetElementAttributeValue, 1262
GetElementHierarchy, 1263
GetEntityNames, 1263
Goto, 1264
GotoNext, 1264
GotoNextCursorPosition, 1265
GotoPrevious, 1265
GotoPreviousCursorPosition, 1266
HasElementAttribute, 1266
InsertEntity, 1266
InsertRow, 1267
IsCopyEnabled, 1268
IsCutEnabled, 1268
IsDeleteEnabled, 1268
IsEmpty, 1268
isEqual, 1269
IsFirstRow, 1269
IsInDynamicTable, 1269
IsLastRow, 1269
IsPasteEnabled, 1270
IsTextStateApplied, 1270
LastCursorPosition, 1270
LastXMLData, 1271
LastXMLDataOffset, 1272
MoveBegin, 1273
MoveEnd, 1273
MoveRowDown, 1274
MoveRowUp, 1274
Parent, 1274
Paste, 1274
PerformAction, 1275
Select, 1276
SelectNext, 1276
SelectPrevious, 1277
SetElementAttributeValue, 1277
SetFromRange, 1279
Text, 1279

AuthenticView, 1294
Application, 1287
AsXMLString, 1287
DocumentBegin, 1288
DocumentEnd, 1288
Event, 1289
Goto, 1290
IsRedoEnabled, 1291
IsUndoEnabled, 1291
MarkupVisibility, 1291
OnBeforeCopy, 1281
OnBeforeCut, 1282

© 2015 Altova GmbH
Index 1559

AuthenticView, 1294
  OnBeforeDelete, 1282
  OnBeforeDrop, 1282
  OnBeforePaste, 1283
  OnMouseEvent, 1284
  OnSelectionChanged, 1284
  Parent, 1292
  Print, 1292
  Redo, 1292
  Selection, 1293
  Undo, 1294
  WholeDocument, 1294
  XMLDataRoot, 1295

Auto Hide,
  feature of Design Entry Helpers, 48

Auto-add Date Picker, 1136

Auto-Calculations, 410
  and conditions, 429
  and output escaping, 510
  based on result of other Auto-Calculations, 416
  command for inserting in design, 1070
  creating, editing, formatting, 410
  example files, 418
  examples, 437
  formatting of date results, 657
  hiding, 413
  how to use, 409
  in Quick Start tutorial, 101
  Java and .NET functions in (Enterprise edition only), 410
  moving, 410
  symbol in Design View, 973
  updating node with value of, 1070
  updating nodes in XML document with value of, 413, 416

Auto-completion in DB Queries, 771

Auto-Macro setting, 1204

Automated processing, 940

Auto-numbering, 488

Batch files,
  and scheduled tasks, 951

Blueprints for layout, 311

Bookmarks, 304, 494
  command for inserting in design, 1086
  creating and editing, 495
  deleting, 495
  enclosing with, 1112

Bookmarks (anchors),
  symbol in Design View, 973

Bookmarks in DB Queries, 771

Borders,
  of SPS tables, 1125

Breaks, 577

Browser pane,
  in Database Query window, 767

Bullets and Numbering, 193, 194, 196, 1083, 1151
  enclosing with, 1111

Buttons, 303

C

C#,
  integration of StyleVision, 1351

CALS/HTML tables, 188, 1127

candlestick chart features, 233

Candlestick charts, 283

Carriage return key,
  see Enter key, 923

Catalog files, 141

CDATA sections, 149
  inserting in SPS tables, 1123

Cell (of table),
  split horizontally, 1124
  split vertically, 1124

Cells,
  joining in SPS tables, 1123

Certificates in XML signatures, 691

Change To command, 323

Changing view,
  to Authentic View, 881

Character references,
  and output escaping, 510

Chart functions,
  chart data structure for, 1495
  example, 1500

© 2015 Altova GmbH
Chart functions, listing, 1491

Charts,
- 3d settings, 242
- adding legend, 231
- and series, 265
- appearance, 220
- area chart features, 233
- area charts, 278
- background color, 231
- bar chart features, 233
- bar charts, 265
- basics, 210
- Candlestick, 283
- candlestick chart features, 233
- category line graphs, 271
- Chart Configuration dialog, 207
- color range, 236
- color schema, 236
- color scheme, 214
- configuration, 214
- creating grouped bars with series, 265
- data selection for, 247, 252
- defining colors, 236
- dynamic XPath settings, 214
- flexible selection, 252
- fonts, 214, 243
- gauge chart features, 233
- gauge charts, 289
- grid lines, 238, 240, 241
- grouped bar charts, 265
- how they work, 210
- insert, 1076
- labels, 214
- legends, 214
- line chart features, 233
- line graphs, 271, 274
- margins, 242
- overlay charts, 292
- overview, 207
- pie chart features, 233
- pie charts, 260
- plot, 214
- removing legend, 231
- series color, 236
- settings, 214
- simple examples, 210
- simple selection, 247
- sizes, 242
- stacked bar charts, 265
- tick size, 242
- title, 231
- using editable Authentic variables to select axes data, 260, 289
- using Java and .NET functions to select axes data, 274
- using the series axis, 271
- value line graphs, 274
- X-axis, 238
- Y-axis, 240
- Z-axis, 241

Check boxes, 298

Class attributes,
in Quick Start tutorial, 95

Class ID,
in StyleVision integration, 1352

Close (SPS) command, 1012

Close Project, 1052

Column,
- append to SPS table, 1121
- delete from table in SPS, 1122
- insert in SPS table, 1121

Columns,
- forcing breaks, 1102
- inserting, 1102

Columns (of tables),
- hiding in HTML output, 186

Columns for print output, 564

COM-API,
documentation, 1207

Combo box,
in Quick Start tutorial, 105

Combo boxes, 300

Command line, 940
- and parameters, 448
- and scheduled tasks, 951

Command line utility, 33

Commands,
customizing, 1163

Comments in DB Queries, 771

Companion software,
for download, 1180

Complex global template, 373

Component download center,
at Altova web site, 1180

Composite styles, 543
- for Authentic View, 547
Index

Condition,
command for inserting in design, 1092

Conditional Presence, 430

Conditional templates, 1092
see under: Conditions, 422
symbol in Design View, 973

Conditions,
and Auto-Calculations, 429
editing, 426
enclosing with, 1113
for different outputs, 427
in Quick Start tutorial, 105
output-based, 427
setting up, 423

Configurations,
of a global resource, 601, 1162

Configurations in global resources, 622

Connection strings,
for DBs, via ADO, 719
for DBs, via ODBC, 726

Consecutive markup, 42

Content editing procedures, 148

Content model,
effect of ambiguity on design, 153

Contents,
command for inserting in design, 1064

Contents placeholder,
in Quick Start tutorial, 83
inserting node as contents, 149

Context menus,
in Authentic View, 890

Context node,
in XPath dialog, 978, 980

Copy command, 1042

Copyright information, 1540

Cover pages, 561

Creating new SPS document,
in Quick Start tutorial, 79

Criteria,
in DB Filters, 751

Cross references, 492

CSS files,
managing in Design Overview sidebar, 52

CSS styles,
in Modular SPSs, 362
in Quick Start tutorial, 95
see also Styles, 67

CSS stylesheets,
also see Styles, 525
external stylesheets, 525
import precedence of external, 525
media applied to, 525

Custom buttons for Authentic toolbar, 1131

Custom dictionaries,
for SPS spell-checks, 1157

Customize dialog,
for customizing StyleVision, 1060

Customizing StyleVision, 1163

Cut command, 1042

D

Data Source Names (DSNs),
and ODBC connections, 726

Database,
toolbar buttons for editing, 1002

Database (Enterprise and Professional editions),
see under DB, 3

Database Query,
Browser pane in DB Query window, 767
Connecting to DB for query, 764
creating the query, 775
Messages pane, 776
Results of, 776

Database Query window, 762
toggling view on and off, 1147

Databases,
and global resources, 620
see also DB, 907
see under DB, 708

DatabaseSpy,
3d charts, 242
area chart features, 233
bar chart features, 233
candlestick chart features, 233
chart background, 231
chart colors, 236
chart font options, 243
chart fonts, 243
chart grid, 238, 240, 241
chart legend, 231
chart title, 231
chart X-axis, 238
chart Y-axis, 240
DatabaseSpy,
  chart Z-axis, 241
cart sizes, 242
gauge chart features, 233
line chart features, 233
pie chart features, 233
Data-entry devices, 295
  menu commands for inserting, 1068
  symbol in Design View, 973
Datatypes,
  in DB Filters, 751
Date,
  formatting of, 513
Date Picker,
  adding by default to date nodes, 1136
  and XSD datatypes, 655
  command for inserting in design, 1072
  description of use, 655
  inserting in SPS, 655
  lexical format of date entries, 655
  using in Authentic View, 655, 916
Dates,
  and Date Picker, 1072
  and the Date Picker, 655, 1136
  changing manually, 917
  examples of data manipulation with XPath 2.0, 653
  formatting of, 657
  how to use in SPS, 653
DB, 747, 907, 909
  building connection strings for, 719, 726
  connecting to an IBM DB2 XML DB, 714
  connecting to an IBM DB2 XML DB via ADO, 719
  connecting to an IBM DB2 XML DB via JDBC, 729
  connecting to an IBM DB2 XML DB via ODBC, 726
  connecting to an MS Access DB, 714
  connecting to an MS SQL Server via ADO, 719
  connecting to from StyleVision, 712, 714, 719, 726, 729, 733
  connecting to via a global resource, 733
  connecting to with ADO, 719
  connecting to with JDBC, 729
  connecting to with ODBC, 726
  connecting with the Connection Wizard, 714
  creating queries, 909
  editing in Authentic View, 907, 913
  filtering display in Authentic View, 909
  generated XML data files, 710
  generated XML Schema file, 710
  generating XML files and HTM/L PDF output, 760
  navigating tables in Authentic View, 908
  parameters in DB queries, 909
  queries in Authentic View, 907
  records to display in Authentic View, 623
  schema file for, 747
  selecting schema for SPS, 734
  selecting schema for SPS (non-XML DBs), 735
  selecting schema for SPS (XML DBs), 743
  selecting working XML data for SPS, 734
  selecting working XML data for SPS (non-XML DBs), 735
  selecting working XML data for SPS (XML DBs), 743
  setting up a DB-based SPS, 712
  using network shares and UNC paths, 714
  work mechanism in StyleVision, 710
  working with in StyleVision, 708
XML data file, 747
DB controls, 1137
  auto-insertion, 756
  command for inserting, 1069
DB Filters,
  clearing, 1148
  creating and modifying, 751
  datatypes in, 751
  editing, 1148
  filtering data for XML file, 751
DB Parameter Defaults, 751
DB Parameters,
  creating and editing, 1047
  usage, 751
DB Query button,
  inserting in SPS, 756
DB schemas (Enterprise and Professional editions), 339
DB table navigation controls, 1137
Debugging macros, 1206
Decimals,
  formatting of, 513
Default user dictionary,
  for SPS spell-checks, 1157
Defining,
  3d settings, 242
  area chart features, 233
  bar chart features, 233
  candlestick chart features, 233
  chart fonts, 243
  charts colors, 236
  charts sizes, 242
  charts title, 231
Index

Defining,
- color of charts, 236
- fonts in charts, 243
- gauge chart features, 233
- grid lines, 238, 240, 241
- line chart features, 233
- pie chart features, 233
- X-axis settings, 238
- Y-axis settings, 240
- Z-axis settings, 241

Delete,
- column from table in SPS, 1122
- row from table in SPS, 1122
- table in SPS, 1119

Delete command, 1042

Deleting,
- a DB Filter, 751

Design elements, 997

Design Entry Helper windows,
- docking, 48
- floating, 48

Design Entry Helpers,
- Auto Hide, 48
- description of, 48
- Hide, 48
- switching display on and off, 1061

Design Filters,
- switching on and off, 1062

Design Fragment,
- insert, 1098

Design Fragments, 387

Design Overview,
- sidebar window, 52

Design structure, 328

Design Tree,
- and Modular SPSs, 362
- see also Design Entry Helpers, 48
- sidebar window, 60

Design View,
- and JavaScript Editor, 42
- description of, 42
- display of markup, 42
- symbols in SPS design, 973

Dictionaries,
- for SPS spell-checks, 1157

disable-output-escaping, 510

Distribution,
- of Altova's software products, 1540, 1541, 1543

Docking,
- Design Entry Helper windows, 48

Document,
- Application, 1297
- Close, 1297
- FullName, 1298
- GetPathName, 1298
- IsValid, 1298
- Name, 1299
- Path, 1300
- Save, 1300
- SaveAs, 1300
- Saved, 1301
- SetPathName, 1305

Document element,
- definition of, 29

Document elements (see Root elements), 330

Document Events, 1202

Document node,
- definition of, 29

Document properties, 405

Document styles, 405

Document views,
- in GUI, 40

Documentation,
- overview of, 6

Document-level,
- examples of integration of XMLSpy, 1350
- integration of StyleVision, 1346, 1347, 1348, 1349
- integration of StyleVisionControl, 1345

Documents,
- Count, 1306
- Item, 1307
- opening and closing, 40

DPI, 1035
- and pixel-defined lengths, 584
- and print output, 584

DSN,
- see under Data Source Names (DSNs), 726

DTD,
- declaring unparsed entities, 630

DTDs,
- as SPS source, 333

Dynamic (SPS) tables in Authentic View,
- usage of, 899

Dynamic content,
- in Quick Start tutorial, 83

Dynamic lists, 193, 196, 1083
Dynamic table,
  toolbar buttons for editing, 992
Dynamic tables, 165
  and global templates, 171
  difference from appended/inserted rows, 171
  editing, 881
  editing in Authentic View, 1145
  headers and footers in, 171
  nested dynamic tables, 171
  see also SPS tables, 171
  see also Tables, 181

Entity references,
  and output escaping, 510
Entry helpers in Design View,
  switching display on and off, 1061
Enumerations,
  in StyleVisionControl, 1426
  SPYAuthenticationActions, 1334
  SPYAuthenticationDocumentPosition, 1335
  SPYAuthenticationElementKind, 1336
  SPYAuthenticationMarkupVisibility, 1337
  SPYMouseEvent, 1339
  SPYXMLDataKind, 1342
Evaluation key,
  for your Altova software, 1178
Evaluation period,
  of Altova's software products, 1540, 1541, 1543
Event, 1281, 1282, 1283, 1284
Event handlers,
  assigning functions to, 664
  in Scripting Project, 1202
  overview, 1190
Events, 1202
  and event handlers, 1192
Exit command, 1041
Expressions,
  in DB Filter criteria, 751
Extension functions for XSLT and XQuery, 1506
Extension Functions in .NET for XSLT and XQuery,
  see under .NET extension functions, 1516
Extension Functions in Java for XSLT and XQuery,
  see under Java extension functions, 1507
Extension Functions in MSXSL scripts, 1523

F

FAQs on StyleVision, 1180
Features,
  of StyleVision, 24
File menu, 1004
  command Exit, 1041
  File | Close, 1012
  File | Encoding, 1035
  File | New, 1005
  File | Open, 1012
  File | Print, 1040
  File | Print Preview, 1040
File menu, 1004
  File | Save As, 1023
  File | Save Authentic XML Data, 1027
  File | Save Design, 1018
  File | Save Generated Files, 1028

File modification alerts,
  in Modular SPSSs, 362

Files,
  open recently used, 1041

Filters,
  for viewing templates selectively, 1000

Filters (for DB),
  clearing, 1148
  editing, 1148

Filters for design templates,
  switching on and off, 1062

Filters on node-templates, 383

Find command, 1044

Find Next command, 1044

Floating,
  Design Entry Helper Windows, 48

FO processor, 1169
  setting up, 1169
  setting up in StyleVision, 1169

FO processor (Enterprise edition),
  setting up, 33

FO transformations, 946, 947

Fonts,
  for PDF output, 580

Footers,
  adding in table, 1120
  in tables, 181

Footers and headers,
  containing subtotals, 574
  in paged media output, 571, 574

FOP,
  and font handling, 580

FOP compliance, 1169

Form controls,
  menu commands for inserting, 1068

Form Object Palette, 1186

Form Object properties, 1197

Format strings,
  defining for Input Formatting, 1152

Formatting,
  also see Presentation, 506
  for tables, 181
  lists, 991
  nodes on insertion, 152
  of numeric fields, 513
  overview of procedures, 506
  predefined HTML formats, 991
  text alignment, 991
  text properties, 991
  toolbar buttons for, 991

Formatting numbers,
  in Auto-Numbering, 488

Form-based designs, 310, 1005

Forms,
  and event handling, 1200
  and Form Objects, 1197
  creating new, 1195
  in Scripting Projects, 1194
  invocation of, 1192
  naming, 1195
  overview, 1190
  properties of, 1195
  setting tab sequence of objects, 1197

Functions,
  in XPath, defined by user, 636

G

gauge chart features, 233

General usage procedure, 120

Generated files, 132

Global declarations, 1193
  overview, 1190

Global Resource,
  and databases, 733

Global Resources, 599
  changing configurations, 622
  defining, 601
  defining database-type, 612
  defining file-type, 604
  defining folder-type, 610
  dialog, 1161
  selecting configuration via toolbar, 1001
  toolbar, 1001
  using, 615, 616, 620, 622

Global Resources XML File, 601

Global styles,
  see under Styles, 530

Global templates, 371, 372, 373
Global templates, 371, 372, 373  
  effect on rest-of-contents, 156  
in Quick Start tutorial, 112  
Global types,  
in templates, 373  
Graphics,  
  overview of use in SPS, 199  
  see also under Images, 199  
Graphics formats,  
in Authentic View, 922  
Grid View Events, 1202  
Grouped bar charts, 265  
Grouping, 432  
  group-by example (Persons.sps), 435  
  group-by example (Scores.sps), 437  
GUI,  
  description of, 38  
document views in, 40  
  Main Window of, 40  
multiple documents in, 40  
  of HTML tables, 673  
  schema structure, 670  
  SPS design, 670  
HTML output, 132  
  and image support, 203  
HTML tables, 188, 1127  
HTML to XML conversion, 667  
Hyperlink,  
  command for inserting in design, 1088  
Hyperlinks, 304, 494  
  and unparsed entities, 497  
  creating and editing, 497  
  enclosing with, 1112  
  linking to bookmarks, 497  
  linking to external resources, 497  
  locating via hyperlinks, 630  
  removing and deleting, 497  
  symbol in Design View, 973  
IBM DB2 XML DB,  
  connecting to with ADO, 719  
  connecting to with ODBC, 726  
  connecting to with the Connection Wizard, 714  
IE 9,  
  see under Internet Explorer compatibility, 127  
Image,  
  command for inserting in design, 1078  
Image formats,  
in Authentic View, 922  
Images,  
  accessing for output rendering, 200  
  and unparsed entity URIs, 200  
  example files, 206  
  in Quick Start tutorial, 90  
  locating via unparsed entities, 630  
  specifying URIs for, 200  
  supported types, 203  
  symbol in Design View, 973  
Import of XSLT templates,  
  into SPS, 391  
Initial Document Section, 561  
Input fields, 297  
Input formatting,  
  defining format strings for, 1152
Index

Input formatting,
of dates, 657

Insert,
column in SPS table, 1121
row in SPS table, 1121

Insert menu, 1063
Bullets and Numbering, 1083
Insert | Auto-Calculation, 1070
Insert | Bookmarks, 1086
Insert | Condition, 1092
Insert | Contents, 1064
Insert | Date Picker, 1072
Insert | Design Fragment, 1098
Insert | Horizontal Line, 1081
Insert | Hyperlink, 1088
Insert | Image, 1078
Insert | Page, 1102
Insert | Paragraph, 1073
Insert | Rest of contents, 1065
Insert | Special Paragraph, 1073

Inserting design elements via the toolbar, 997

Integer,
formatting of, 513

Integrating,
StyleVision in applications, 1343

Interface,
see GUI, 38

Internet Explorer compatibility, 127
Internet usage,
in Altova products, 1539

iSeries,
must disable timeout, 712

JavaScript,
see under Scripts, 660

JavaScript Editor, 660, 662
in Design View, 42

JDBC connections to DBs, 729

Joining cells,
in SPS tables, 1123

JRE,
for StyleVision Plugin for Eclipse, 959

K

Keeps, 577

Keyboard shortcuts,
customizing for commands, 1163

Key-codes,
for your Altova software, 1178

L

Labels in XBRL, 786
Language,
scripting language - changing, 1192

Layout,
of views in the GUI, 48

Layout Box, 1099
Layout Boxes, 315

Layout Container, 1099
Layout Containers, 311

Layout containers and elements, 997

Layout Modules,
steps for creating, 310

Legal information, 1540
License, 1544
information about, 1540

License metering,
in Altova products, 1542

Licenses,
for your Altova software, 1178

Line,
Line, in Layout Containers, 1099
line chart features, 233
Links, following in Authentic View, 893
see under Hyperlinks, 304, 494
List properties, 1151
Lists, 193
enclosing with, 1111
imported from HTML document, 673
in Quick Start tutorial, 105
Lists (static and dynamic), 1083
Local styles,
see under Styles, 533
Local template, 371, 372

M
Macros,
creating with Scripting Editor, 1204
debugging, 1206
editing with Scripting Editor, 1204
execution of, 1192
functions for, in Global Declarations, 1193
how to use in Scripting Project, 1203
overview, 1190
running, 1205
setting as Auto-Macro in Scripting Editor, 1204
Main schema, 372
Main schema (Enterprise Edition only), 56
Main template, 371, 372
definition of, 29
Markup,
in Authentic View, 881, 883, 1143
Markup tags in Design View, 42
Memory requirements, 1535
Menu,
customizing, 1163
Menu bar,
moving, 38
Microsoft Office 2007 (Enterprise Edition only), 24, 46
Mixed markup, 623
Modular SPS,
activating and de-activating, 362
adding the SPS module, 362
and CSS styles, 358, 362
and file modification alerts, 362
and module objects, 358
and namespace declarations, 358
and schema sources, 358, 362
and Scripts, 358
and Template XML Files, 358
and Working XML Files, 358
creating, 362
effect of order on precedence, 362
example project, 366
overview, 355
the SPS module to add, 362
working with, 362
Modules,
managing in Design Overview sidebar, 52
MS Access DB,
connecting to with the Connection Wizard, 714
using network shares and UNC paths, 714
MS SQL Server,
connecting to with ADO, 719
msxml:script, 1523
Multiline input fields, 297
Multiple document-outputs, 1101
Multiple languages examples, 502
Multiple output-documents, 393
and output previews, 402
linking between, 399
location of files, 402
Multiple schemas (Enterprise edition),
and global templates, 349
and main schema, 349
and schema parameters, 349

N
Named templates, 371
Namespaces,
adding to the SPS, 56, 122, 132, 330
in the SPS, 56
overview of, 60
Network shares,
for connecting to a DB, 714
New command, 1005
New document templates, 393
and design structure, 397
inserting, 395
New document templates, 393
   URLs of, 399
New features, 10
   v2010, 16
   v2011, 14
   v2012, 13
   v2013, 12
New from XSLT, 632
New Project, 1051
New releases of StyleVision, synchronizing with StyleVision, 131
Node, changing what it is created as, 323
Node-templates,
   and chaining to child templates, 383
   and global templates, 383
   and XPath filters, 383
   operations on, 383
   User-Defined, 157
Numbering nodes automatically, 488
Numbers, formatting of, 513
Numeric fields, formatting of, 513

O
Object Locator, in Database Query window, 767
ODBC connections to DBs, 726
Office Open XML (Enterprise Edition only), 24, 46
Onscreen help,
   index of, 1177
   searching, 1177
   table of contents, 1177
OOXML (Enterprise Edition only), 24, 46
Open,
   recently used files, 1041
Open (SPS) command, 1012
Open Project, 1051
Options,
   3d charts, 242
   area chart features, 233
   bar chart features, 233
   candlestick chart features, 233
   chart colors, 236
   chart fonts, 243
   chart grid, 238, 240, 241
   chart legend, 231
   chart title, 231
   chart X-axis, 238
   chart Y-axis, 240
   chart Z-axis, 241
   charts background, 231
   charts sizes, 242
   gauge chart features, 233
   line chart features, 233
   pie chart features, 233
Ordering Altova software, 1178
OS, for Altova products, 1535
Otherwise condition branch, 423
Output encoding, 1035
Output escaping, 510
Output files, from DB-based <%SV-PS%>, 760
generating, 132
Output Views, description of, 46
Output-based conditions, 427
Overlay charts, 292
Overview, of XMLSpy API, 1209

P
Page, 564
   commands for design, 1102
   numbering in PDF output, 564
   setting margins of for PDF output, 564
   setting size of for PDF output, 564
Page breaks, 577, 1102
Page numbers (Enterprise Edition), 1102
Page properties in PDF, 564
Page total (Enterprise Edition), 1102
Paged media,
   and pixel-defined lengths, 584
designing for, 556
   headers and footers, 571
   margins, 564
   page definitions, 564
   page size, 564

© 2015 Altova GmbH
Paged media,
pagination, 564
properties, 556
Paragraph,
command for inserting in design, 1073
enclosing with, 1110
Parameter defaults,
in DB Filters, 751
Parameters, 447
and Authentic View, 448
and command line, 448
creating and editing, 1047
for design fragments, 450
for schema sources, 453
general description, 448
in DB Filters, 751
in DB queries, 909
in SPS, 448
locating nodes in in multiple documents with, 453
managing in Design Overview sidebar, 52
overview of user-defined parameters, 60
Parent, 1300
Parser,
built into Altova products, 1536
Paste,
as Text, 893
as XML, 893
Paste As,
Text, 890
XML, 890
Paste command, 1042
PDF,
defining page properties, 564
PDF Bookmarks, 587
PDF fonts, 580
PDF output,
see Paged Media, 571
PDF output (Enterprise edition), 132
and image support, 203
PDF Preview,
setting up the FO processor, 1169
Pie chart features, 233
Pie charts,
see also under Charts, 260
settings, 260
Pixel-defined lengths,
and paged media, 584
Pixels,
and print media lengths, 1035
and screen resolution, 1035
Platforms,
for Altova products, 1535
Precedence,
of styles, 64
Predefined format strings,
for input formatting, 1152
Predefined formats,
command for inserting in design, 1073
on inserting a node, 152
symbol in Design View, 973
Presentation,
also see Formats, Formatting, 506
overview of procedures, 506
Print command, 1040
Print output,
see Paged media, 556
Print Preview media, 1040
Problems with preview, 33
Processors,
for download, 1180
Product features,
listing of, 24
Project menu, 1049
Project Add Active and Related Files to Project command, 1055
Add Active File to Project command, 1055
Add Altova Resource to Project command, 1053
Add External Folder / Web Folder to Project command, 1056
Add Files to Project command, 1053
Add Project Folder to Project command, 1056
Add URL to Project command, 1053
Close command, 1052
New command, 1051
Open command, 1051
Reload command, 1051
Save command, 1052
Project options, 1169
Project sidebar, 74
Projects,
and drag-and-drop, 1049
detailed description of, 135
using, 135
Properties,
and property groups, 69
defining, 69
for nodes in Authentic View, 623
of SPS tables, 992, 1126
Properties,
   see also Design Entry Helpers, 48
   sidebar window, 69
Properties and Events pane, 1186
Properties Entry Helper,
   Event group, 664
Properties menu, 1150
   Bullets and Numbering, 1151
Properties of output documents, 405
PXF files, 699
   creating, 700
deploying, 706
   editing, 704
   saving as, 1023

Q

Queries,
   for DB display in Authentic View, 909
Query,
   see under Database Query, 762
Query button,
   inserting in SPS, 756
Query Database,
   see under Database Query, 762
Query Database command, 762
Query pane,
   in Database Query window, 771
Quick Start tutorial,
   Auto-Calculations, 101
class attributes, 95
   combo boxes, 105
   conditions, 105
   contents placeholder, 83
creating new SPS document, 79
   CSS styles, 95
dynamic content, 83
generating XSLT stylesheets, 117
global templates, 112
horizontal lines, 90
images, 90
introduction, 78
lists, 105
predefined formats, 95
required files, 78
rest-of-contents, 112
setting up new SPS document, 79
static content, 90
static text, 90
testing Authentic View (Enterprise and Professional editions), 117

R

Radio buttons, 303
RaptorXML, 940
   and FOP, 946
Recently used files, 1041
Records displayed in Authentic View,
   setting, 756
Redo command, 1043
Regions in DB Queries, 771
Registering your Altova software, 1178
Reload Project, 1051
Replace command (Enterprise and Professional editions), 1044
Rest-of-contents, 156
   and global templates, 373
   command for inserting in design, 1065
   in Quick Start tutorial, 112
Restore toolbars and windows, 1168
Return key,
   see Enter key, 923
Rich Edit, 549
RichEdit,
   commands in Authentic View, 1144
   insert, 1066
RichEdit toolbar, 996
Root elements, 56
Root elements (aka document elements),
   and schema sources, 330
   selecting for schema, 330
Row,
   append to SPS table, 1121
delete from table in SPS, 1122
   insert in SPS table, 1121
Rows (of tables),
   expanding/collapsing in HTML output, 186
RTF output,
   see Paged Media, 571
RTF output (Enterprise edition), 132
   and image support, 203
Running totals,
in headers and footers, 574

Save,
Working XML File, 1027
Save Authentic XML Data command, 1027
Save Design command, 1018
Save Generated Files command, 1028
Save Project, 1052
Scheduled task,
creating a StyleVisionBatch command as, 951
StyleVisionBatch batch files in, 951
Schema for DB-based SPSSs, 735, 743
Schema sources, 122, 1005
and root elements (document elements), 330
changing sources, 453
managing in Design Overview sidebar, 52
multiple in SPSS (Enterprise edition), 330
multiple sources and locating nodes, 453
multiple sources and XPath, 453
overview of, 60
selecting for SPSS, 330
sidebar window, 56
Schema Sources window,
see also Design Entry Helpers, 48
Schema structure,
and SPSS design, 153
Schema tree,
for XBRL, 783
Schema tree options, 1169
Schemas,
as SPSS source, 333
from DB for SPSS, 339
from XBRL for SPSS, 340
user-defined, 346
Screen resolution,
and pixel-defined lengths, 584
Scripting,
in Authentic View, 926, 929, 930
Scripting Editor, 929
GUI description, 1186
Main Window, 1186
Scripting Environment, 1183
usage overview, 1185
Scripting language, 1192
Scripting Project,
and Events, 1202
application event handlers, 1202
Event Handlers, 1190
Forms, 1190
Forms in, 1194
Global Declarations, 1190
Global Declarations in, 1193
Macros, 1190
Macros in, 1203
steps for creating, 1192
Scripting Project Tree pane, 1186
Scripts,
and JavaScript functions, 660
defining JavaScript functions, 662
in the Design Tree, 660
JavaScript functions as event handlers, 664
overview of, 60
using in an SPSS, 660
Scripts in XSLT/XQuery,
see under Extension functions, 1506
Scroll buttons,
in Main Window, 40
Sections,
and page layout, 558
default properties for new sections, 561
deleting, 558
in the SPSS design, 558
Initial Document Section, 561
inserting, 1102
Select All command, 1043
Select Tables dialog,
for DB-based SPSSs, 735, 743
Setting up new SPSS document,
in Quick Start tutorial, 79
Setting up StyleVision, 33
Settings,
3d charts, 242
area chart features, 233
bar chart features, 233
candlestick chart features, 233
chart background, 231
chart colors, 236
chart fonts, 243
chart grid, 238, 240, 241
chart legend, 231
chart title, 231
**Index**

**Settings**,  
chart X-axis, 238  
chart Y-axis, 240  
chart Z-axis, 241  
charts sizes, 242  
gauge chart features, 233  
line chart features, 233  
pie chart features, 233

**Shortcuts**,  
customizing for keyboard, 1163

**Show large markup**, 881, 883
**Show markup**, 42
**Show mixed markup**, 881, 883
**Show small markup**, 881

**Signatures**,  
see under XML signatures, 677

**Simple global template**, 373

**Software product license**, 1544

**Sorting**, 441  
example files, 444  
of groups and within groups, 432, 435, 437  
Sorting mechanism, 442  
Sort-keys, 442

**Sort-keys**, 441

**Source files for SPS**, 122

**Special paragraph**,  
command for inserting in design, 1073  
enclosing with, 1110

**Spell-checker**,  
in StyleVision, 1155

**Spell-checker options**,  
for SPSs, 1157

**Split table cell**,  
horizontally, 1124  
vertically, 1124

**SPS**,  
and Authentic View (Enterprise and Professional editions), 21  
and DBs, 708  
and StyleVision, 21  
and XSLT stylesheets, 21  
closing, 1012  
general description of, 21  
opening, 1012  
reloading, 1012

**SPS and Authentic View**, 129
**SPS based on a a DB**,  
see under DB, 712

**SPS design overview**, 124

**SPS file structure**, 328

**SPS tables**,  
editing dynamic tables, 881  
see also Dynamic tables, 165  
see also Static tables, 165

**SPS tables in Authentic View**,  
usage of, 899

**SQL Editor**,  
creating query in, 775  
description of, 771  
in Database Query window, 771

**Stacked area charts**, 278
**Stacked bar charts**, 265

**Static (SPS) tables in Authentic View**,  
usage of, 899

**Static content**,  
in Quick Start tutorial, 90

**Static lists**, 193, 194, 1083, 1111

**Static table**,  
inserting, 1119  
inserting in SPS, 992  
toolbar buttons for editing, 992

**Static tables**, 165  
see also SPS tables, 168  
see also Tables, 181

**Static text**,  
and output escaping, 510  
in Quick Start tutorial, 90

**Status bar**, 1060

**Structure of SPS design**, 328

**Style Repository**,  
and external CSS stylesheets, 525  
and global styles, 530  
see also Design Entry Helpers, 48  
sidebar window, 64

**Styles**,  
and property groups, 67  
assigning CSS stylesheets to SPS, 525  
cascading order, 523  
combining several, 543  
CSS rules combined, 543  
defining, 67  
defining global styles in SPS, 530  
defining local styles, 533  
from XML data, 539  
media for assigned external stylesheets, 525
Styles,
preference of, 64
preference of styles, 530
see also Design Entry Helpers, 48
sidebar window, 67
terminology of, 523
via XPath expressions, 539
working with in StyleVision, 523

Styles of output documents, 405

Stylesheets,
also see under CSS stylesheets, 525
also see under XSLT stylesheets, 525

StyleVision,
integration, 1343
introduction, 20
product features, 24
synchronizing with Authentic, 131
user manual, 3

StyleVision API,
accessing, 1406

StyleVision command table, 1373

StyleVision integration,
example of, 1352, 1353

StyleVision Integration Package, 955, 959

StyleVision perspective in Eclipse, 966

StyleVision Plugin for Eclipse,
installing, 959

StyleVision Plugin for VS .NET,
installing, 955

StyleVision Power Stylesheet,
see under SPS, 3

StyleVisionBatch, 33, 940

StyleVisionCommand,
in StyleVisionControl, 1408

StyleVisionCommands,
in StyleVisionControl, 1410

StyleVisionControl, 1411
documentation of, 1343
example of integration at application level, 1352, 1353
examples of integration at document level, 1350
integration at application level, 1344
integration at document level, 1345, 1346, 1347, 1348, 1349
integration using C#, 1351
integration using HTML, 1354
object reference, 1407

StyleVisionControlDocument, 1417

StyleVisionControlPlaceHolder, 1423

Subtotals,
in headers and footers, 574

Support for StyleVision, 1180

Support options, 6

Symbols in Design View,
of Auto-Calculations, 973
of bookmarks (anchors), 973
of conditional templates, 973
of data-entry devices, 973
of hyperlinks, 973
of images, 973
of predefined formats, 973
of XML document content, 973
of XML document nodes, 973

Table,
adding headers and footers, 1120
append column to, 1121
append row to, 1121
cell content, 1118
delete column from, 1122
delete row from, 1122
deleting in SPS, 1119
editing in Authentic View, 1145
editing properties of, 1126
headers and footers, 1118
insert column in, 1121
insert row in, 1121
inserting a static table, 1119
navigating, 1118
show/hide borders in StyleVision, 1125
vertical alignment of cell content, 1128

Table menu, 1118

Table of contents,
see under TOC, 460

Tables,
Close button to hide columns, 186
creating, 1082
creating dynamic tables, 171
creating static tables, 168
defining dynamic (SPS) tables, 881
expanding/collapsing rows, 186
formatting, 181
headers and footers in PDF, 181
Tables,
  hiding empty columns, 186
  imported from HTML document, 673
  joining cells in, 1123
  overview, 165
  styles for alternate rows, 539
Tables (SPS),
  editing of properties, 992
  toolbar buttons for editing, 992
Tables in Authentic View,
  icons for editing XML tables, 905
  usage of, 898
  using SPS (static and dynamic) tables, 899
  using XML tables, 901
Tables in Design View,
  enclosing with and removing templates, 179
  representation of, 179
Tags,
  expanding and collapsing, 1048
Technical Information, 1534
Technical support for StyleVision, 1180
Template,
  changing the node match for, 323
  enclosing with, 1107
  inserting, 1094
Template filters, 1000
Template XML File (Enterprise and Professional editions), 122
  definition of, 29
Templates,
  enclosing table rows and columns with, 179
  removing from around table rows and columns, 179
  switching view on and off, 1062
  tree of, 60
Templates for nodes,
  see Node-templates, 383
Temporary output document, 33
terminate, 1237
Terminology,
  used in StyleVision, 29
Text,
  editing in Authentic View, 893
  formatting in Authentic View, 893
Text references, 492
Text state icon, 1131
Text State Icons, 553
Text View Events, 1202
Timeout,
  iSeries - must disable, 712
TOC,
  example, hierarchical and sequential, 484
  example, simple, 480
  marking items for inclusion, 464
  menu commands, 1100
  overview of usage, 460
TOC Bookmarks, 464
  and levels, 469
  creating, 469
  enclosing with, 1115
  wizard for, 469
TOC items,
  constructing, 478
  formatting, 478
TOC Levels, 464, 466
  enclosing with, 1115
TOC references, 477
TOC template,
  creating and editing, 473
  formatting, 478
  level references in, 475
  reflevels in, 475
  structuring, 475
TOCrefs,
  see under TOC references, 477
Toolbar buttons,
  adding and removing, 989
Toolbars, 988
  adding/removing icons in, 988
  Authentic toolbar, 994
  customizing, 1060
  Formatting toolbar, 991
  Insert Design Elements toolbar, 997
  moving, 38
  positioning in GUI, 988
  resetting, 988
  RichEdit, 996
  Standard toolbar, 1002
  switching display on and off, 1060
  switching display on/off, 988
  Table toolbar, 992
Tools menu, 1154
Type-based templates, 373
Types as processing units,
  in global templates, 373
U

User-Defined Elements, 161
User-Defined XML Text Blocks, 163
UNC paths,
for connecting to a DB, 714
Undo command, 1043
Unicode support,
in Altova products, 1538
unparsed-entity-uri function of XSLT, 630
Updating nodes,
with values of Auto-Calculations, 416
Updating nodes (Enterprise and Professional editions),
with an Auto-Calculation result, 409
with values of Auto-Calculations, 413
URIs,
holding in unparsed entities, 630
Usage, 120
User info, 623
User Interface,
see GUI, 38
User manual,
see also Onscreen Help, 1177
User reference, 972
User-Defined Elements, 160, 1105, 1117
User-defined schemas, 346
User-defined template,
enclosing with, 1108
inserting, 1096
User-Defined Templates, 157
User-Defined Text Blocks, 160, 1105
User-defined XPath functions, 636

V

Validate XML,
in Authentic View, 1138
Validator,
in Altova products, 1536
Value formatting, 513
Variable template, 382
enclosing with, 1109
inserting, 1097
Variables, 447, 454
assigning values via Authentic, 625
editing in Authentic View, 457
Version 2010 new features, 16
Version 2011 new features, 14
Version 2012 new features, 13
Version 2013 new features, 12
Vertical alignment of table cell content,
in SPSs, 1128
Vertical text,
in layout boxes, 315
in table cells, 181
View menu, 1059
Views,
layout of in GUI, 48
Visual Studio .Net,
and StyleVision, 954
and StyleVision differences, 956
VS .NET,
and StyleVision Integration Package, 955

W

Watermarks, 592
Window menu, 1175
Windows,
support for Altova products, 1535
Word 2007 (Enterprise Edition only), 24, 46
WordML (Enterprise Edition only), 24, 46
Working XML File, 56, 122
and Authentic View, 44
and Output Views, 46
definition of, 29
print preview, 1040
printing, 1040
validating in Authentic View, 1138

X

XBRL,
<all> item in schema tree, 783
Chart Wizard, 847
creating an XBRL SPS file, 781
creating design based on XBRL taxonomy, 781
Index

XBRL,
formatting of design table elements, 864
in SPS designs, 780
specifying labels to use, 786
table options, 836
table structure, 830
Table Wizard, 828
taxonomy source properties, 786
Taxonomy structure in schema tree, 783
XBRL Chart Wizard, 847
XBRL charts,
categories of, 849
component properties, 854
series in, 851
XBRL Table Wizard, 828
XBRL tables,
component properties, 838
XBRL taxonomy (Enterprise edition), 340
XBRL templates, 789
XML,
inserting in design, 163
XML data,
inserting in SPS design, 149
merging from multiple sources, 352
XML data for DB-based SPSs, 735
XML DB,
loading new data row into Authentic View, 1139
loading new XML data row, 908
XML document content,
symbol in Design View, 973
XML document nodes,
symbol in Design View, 973
XML file,
with data from DB, 760
XML file for DB-based SPSs, 743
XML Parser,
about, 1536
XML Schemas and DTDs,
as SPS source, 333
XML Signatures, 920
and certificates, 691
and XML document validity, 689
configuration, 680
overview, 677
settings, 682
signing in Authentic View, 685, 1140
verification in Authentic View, 687, 1140
XML tables (Enterprise and Professional editions), 165
XML tables in Authentic View,
icons for editing, 905
usage of, 901
XMLData,
appendChild, 1318
EraseAllChildren, 1319
EraseCurrentChild, 1319
GetChild, 1320
GetChildKind, 1321
GetCurrentChild, 1322
GetFirstChild, 1322
GetNextChild, 1323
HasChildren, 1324
HasChildrenKind, 1324
InsertChild, 1325
IsSameNode, 1326
Kind, 1326
MayHaveChildren, 1326
Name, 1326
Parent, 1327
TextValue, 1327
XMLSpy, 36
XMLSpy API,
documentation, 1207
overview, 1209
XMLSpyLib, 1207
Application, 1234
AuthenticRange, 1253
AuthenticView, 1281
Document, 1296
Documents, 1306
XMLData, 1317
XPath,
locating nodes in multiple documents, 453
XPath 1.0,
and dates, 653
XPath 2.0,
and dates, 653
XPath dialog,
description of, 978, 980, 985
XPath expressions,
and styles, 539
building in Edit XPath Expression dialog, 978, 980
evaluating in Edit XPath Expression dialog, 985
XPath filter,
on global templates, 373
XPath filters on node-templates, 383
XPath functions,
XPath functions,
in XPath dialog, 978, 980
user-defined, 636
XPath operators,
in XPath dialog, 978, 980
XPath to selected node, 880
XPath version in SPS, 126
XQuery,
Extension functions, 1506
XQuery processor,
in Altova products, 1537
xs:date,
and the Date Picker, 655
xs:dateTime,
and the Date Picker, 655
XSL-FO generation,
setting up, 1169
XSLT,
Extension functions, 1506
inserting code fragment in design, 163
XSLT import, 632
XSLT processors,
in Altova products, 1537
XSLT stylesheet preview,
in Output Views, 46
XSLT Templates, 60
importing into SPS, 391
managing in Design Overview sidebar, 52
XSLT to SPS, 632
XSLT transformations, 946, 947
XSLT version,
setting for SPS, 1002
XSLT version in SPS, 126
XSLTElements,
inserting as code in design, 161