

Altova StyleVision 2024 Professional Edition



User & Reference Manual

Altova StyleVision 2024 Professional Edition User & Reference Manual

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Published: 2024

© 2018-2024 Altova GmbH

Table of Contents

1	Introduction	17
1.1	Product Features.....	18
1.2	Authentic View in Altova Products.....	22
1.3	What Is an SPS?.....	23
1.4	Setting up StyleVision.....	25
1.5	Terminology.....	26
1.6	About This Documentation.....	29
2	User Interface	31
2.1	Main Window.....	32
2.1.1	Design View.....	33
2.1.2	Authentic View.....	35
2.1.3	Output Views.....	36
2.2	Sidebars.....	39
2.2.1	Design Overview.....	42
2.2.2	Schema Tree.....	44
2.2.3	Design Tree.....	48
2.2.4	Style Repository.....	51
2.2.5	Styles	54
2.2.6	Properties.....	55
2.2.7	Project	59
2.2.8	Messages.....	60
2.2.9	Find and Replace.....	61
3	Quick Start Tutorial	63
3.1	Creating and Setting Up a New SPS.....	64
3.2	Inserting Dynamic Content (from XML Source).....	68
3.3	Inserting Static Content.....	75

3.4	Formatting the Content.....	80
3.5	Using Auto-Calculations.....	86
3.6	Using Conditions.....	90
3.7	Using Global Templates and Rest-of-Contents.....	97
3.8	That's It!.....	101
4	Usage Overview	102
4.1	SPS and Sources.....	103
4.2	Creating the Design.....	104
4.3	XSLT and XPath Versions.....	105
4.4	Internet Explorer Compatibility.....	106
4.5	SPS and Authentic View.....	108
4.6	Synchronizing StyleVision and Authentic.....	110
4.7	Generated Files.....	111
4.8	Projects in StyleVision.....	113
4.9	Catalogs in StyleVision.....	118
4.9.1	How Catalogs Work.....	118
4.9.2	Catalog Structure in StyleVision.....	119
4.9.3	Customizing Your Catalogs.....	120
4.9.4	Variables for Windows System Locations.....	122
5	SPS Content	124
5.1	Inserting XML Content as Text.....	125
5.1.1	Inserting Content with a Predefined Format.....	127
5.1.2	Adding Elements in Authentic View.....	128
5.1.3	Rest-of-Contents.....	130
5.2	Inserting MS Word Content.....	132
5.3	Inserting MS Excel Content.....	135
5.4	User-Defined Templates.....	137
5.5	User-Defined Elements, XML Text Blocks.....	140
5.5.1	User-Defined Elements.....	140
5.5.2	User-Defined XML Text Blocks.....	141
5.6	Tables.....	143

5.6.1	Static Tables.....	145
5.6.2	Dynamic Tables.....	146
5.6.3	Conditional Processing in Tables.....	150
5.6.4	Tables in Design View.....	151
5.6.5	Table Formatting.....	153
5.6.6	Row and Column Display.....	157
5.6.7	CALS/HTML Tables.....	158
5.7	Lists.....	163
5.7.1	Static Lists.....	163
5.7.2	Dynamic Lists.....	165
5.8	Graphics.....	168
5.8.1	Images: URIs and Inline Data.....	168
5.8.2	Image Types and Output.....	170
5.8.3	Example: A Template for Images.....	173
5.9	Form Controls.....	174
5.9.1	Input Fields, Multiline Input Fields.....	175
5.9.2	Check Boxes.....	175
5.9.3	Combo Boxes.....	177
5.9.4	Radio Buttons, Buttons.....	180
5.10	Links.....	182
5.11	Barcodes.....	183
5.12	Layout Modules.....	187
5.12.1	Layout Containers.....	187
5.12.2	Layout Boxes.....	190
5.12.3	Lines	194
5.13	The Change-To Feature.....	197
6	SPS Structure	200
6.1	Schema Sources.....	202
6.1.1	DTDs and XML Schemas.....	203
6.1.2	DB Schemas.....	208
6.1.3	User-Defined Schemas.....	209
6.1.4	Schema Manager.....	211
6.2	Merging XML Data from Multiple Sources.....	227

6.3	Modular SPSs.....	230
6.3.1	Available Module Objects.....	231
6.3.2	Creating a Modular SPS.....	234
6.3.3	Example: An Address Book.....	238
6.4	Templates and Design Fragments.....	244
6.4.1	Main Template.....	244
6.4.2	Global Templates.....	244
6.4.3	User-Defined Templates.....	248
6.4.4	Variable Templates.....	251
6.4.5	Node-Template Operations.....	252
6.4.6	Design Fragments.....	255
6.5	XSLT Templates.....	259
6.6	Multiple Document Output.....	261
6.6.1	Inserting a New Document Template.....	262
6.6.2	New Document Templates and Design Structure.....	263
6.6.3	URLs of New Document Templates.....	263
6.6.4	Preview Files and Output Document Files.....	265
6.6.5	Document Properties and Styles.....	268

7 Advanced Features 269

7.1	Auto-Calculations.....	270
7.1.1	Editing and Moving Auto-Calculations.....	270
7.1.2	Updating Nodes with Auto-Calculations.....	272
7.1.3	Auto-Calculations Based on Updated Nodes.....	275
7.1.4	Example: An Invoice.....	276
7.2	Conditions.....	280
7.2.1	Setting Up the Conditions.....	280
7.2.2	Editing Conditions.....	283
7.2.3	Output-Based Conditions.....	284
7.2.4	Conditions and Auto-Calculations.....	285
7.3	Conditional Presence.....	286
7.4	Grouping.....	288
7.4.1	Example: Group-By (Persons.sps).....	290
7.4.2	Example: Group-By (Scores.sps).....	292

7.5	Sorting.....	296
7.5.1	The Sorting Mechanism.....	296
7.5.2	Example: Sorting on Multiple Sort-Keys.....	298
7.6	Parameters and Variables.....	302
7.6.1	User-Declared Parameters.....	302
7.6.2	Parameters for Design Fragments.....	304
7.6.3	SPS Parameters for Sources.....	306
7.6.4	Variables.....	307
7.6.5	Editable Variables in Authentic.....	309
7.7	Table of Contents, Referencing, Bookmarks.....	312
7.7.1	Bookmarking Items for TOC Inclusion.....	315
7.7.2	Creating the TOC Template.....	322
7.7.3	Example: Simple TOC.....	327
7.7.4	Example: Hierarchical and Sequential TOCs.....	331
7.7.5	Auto-Numbering in the Document Body.....	334
7.7.6	Cross-referencing.....	338
7.7.7	Bookmarks and Hyperlinks.....	339
7.8	Example: Multiple Languages.....	346

8 Presentation and Output 349

8.1	Predefined Formats.....	350
8.2	Output Escaping.....	352
8.3	Value Formatting (Formatting Numeric Datatypes).....	354
8.3.1	The Value Formatting Mechanism.....	354
8.3.2	Value Formatting Syntax.....	357
8.4	Working with CSS Styles.....	364
8.4.1	External Stylesheets.....	365
8.4.2	Global Styles.....	369
8.4.3	Local Styles.....	371
8.4.4	Setting Style Values.....	373
8.4.5	Style Properties Via XPath.....	375
8.4.6	Composite Styles.....	378
8.5	Text-Styling Flexibility in Authentic.....	381
8.5.1	Composite Styles.....	381

8.5.2	RichEdit.....	383
8.5.3	Text State Icons.....	387
8.6	HTML Document Properties.....	390
8.7	Designing Print Output.....	392
8.7.1	Document Sections.....	393
8.7.2	Keeps and Breaks.....	407
8.7.3	Footnotes.....	407
8.7.4	Pixel Resolution.....	409
8.7.5	Watermarks.....	411

9 Additional Functionality 415

9.1	Altova Global Resources.....	416
9.1.1	Defining Global Resources.....	416
9.1.2	Using Global Resources.....	427
9.2	Authentic Node Properties.....	433
9.3	Replace Parent Node OnClick With.....	435
9.4	Additional Validation.....	438
9.5	Unparsed Entity URIs.....	440
9.6	New from XSLT, XSL-FO or FO File.....	442
9.7	User-Defined XPath Functions.....	446
9.7.1	Defining an XPath Function.....	448
9.7.2	Reusing Functions to Locate Nodes.....	451
9.7.3	Parameters in XPath Functions.....	452
9.8	Working with Dates.....	461
9.8.1	Using the Date-Picker.....	462
9.8.2	Formatting Dates.....	463
9.9	Using Scripts.....	467
9.9.1	Defining JavaScript Functions.....	468
9.9.2	Assigning Functions as Event Handlers.....	469
9.9.3	External JavaScript Files.....	470
9.10	HTML Import.....	472
9.10.1	Creating New SPS via HTML Import.....	472
9.10.2	Creating the Schema and SPS Design.....	474
9.10.3	Creating Tables and Lists as Elements/Attributes.....	476

9.10.4	Generating Output.....	478
9.11	ASPX Interface for Web Applications.....	479
9.11.1	Example: Localhost on Windows 7.....	480
9.12	PXF File: Container for SPS and Related Files.....	482
9.12.1	Creating a PXF File.....	482
9.12.2	Editing a PXF File.....	485
9.12.3	Deploying a PXF File.....	486

10 Databases 488

10.1	DBs and StyleVision.....	490
10.2	Connecting to a Data Source.....	492
10.2.1	Start Database Connection Wizard.....	493
10.2.2	Database Drivers Overview.....	495
10.2.3	ADO Connection.....	498
10.2.4	ADO.NET Connection.....	503
10.2.5	ODBC Connection.....	510
10.2.6	JDBC Connection.....	513
10.2.7	SQLite Connection.....	517
10.2.8	Native Connection.....	519
10.2.9	Global Resources.....	520
10.2.10	Database Connection Examples.....	521
10.3	DB Data Selection.....	575
10.3.1	Non-XML Databases.....	575
10.3.2	XML Databases.....	582
10.4	The DB Schema and DB XML files.....	586
10.5	DB Filters: Filtering DB Data.....	589
10.6	SPS Design Features for DB.....	594
10.7	Generating Output Files.....	598
10.8	Query Database.....	599
10.8.1	Data Sources.....	600
10.8.2	Browser Pane: Viewing the DB Objects.....	603
10.8.3	Query Pane: Description and Features.....	607
10.8.4	Query Pane: Working with Queries.....	609
10.8.5	Results and Messages.....	610

11	Authentic View	613
11.1	Authentic View Interface.....	614
11.1.1	Overview of the GUI.....	614
11.1.2	Authentic View Toolbar Icons.....	615
11.1.3	Authentic View Main Window.....	617
11.1.4	Authentic View Entry Helpers.....	619
11.1.5	Authentic View Context Menus.....	623
11.2	Editing in Authentic View.....	626
11.2.1	Basic Editing.....	626
11.2.2	Tables in Authentic View.....	631
11.2.3	Editing a DB.....	638
11.2.4	Working with Dates.....	644
11.2.5	Defining Entities.....	646
11.2.6	Images in Authentic View.....	648
11.2.7	Keystrokes in Authentic View.....	649
12	Authentic Scripting	650
12.1	Scripting Editor.....	652
12.2	Macros.....	653
12.2.1	Macros on Design Elements.....	653
12.2.2	Macros on Context Menu Items.....	655
12.2.3	Custom Buttons.....	656
12.3	Event Handlers.....	659
12.4	Scripting Options.....	660
13	Automated Processing	661
13.1	Command Line Interface.....	662
13.1.1	StyleVision.....	662
13.1.2	StyleVision Server.....	663
13.2	Using RaptorXML.....	665
13.2.1	PDF Output.....	665
13.3	Automation with FlowForce Server.....	667

13.4	How to Automate Processing.....	669
14	StyleVision in Visual Studio	670
14.1	Installing the StyleVision Plugin.....	671
14.2	Differences with StyleVision Standalone.....	672
15	StyleVision in Eclipse	673
15.1	Install the Integration Package for Eclipse.....	674
15.2	StyleVision Perspective in Eclipse.....	676
15.3	Other Stylevision Entry Points in Eclipse.....	679
16	Menu Commands and Reference	680
16.1	Design View Symbols.....	681
16.2	Edit XPath Expression Dialog.....	685
16.2.1	Evaluator.....	686
16.2.2	Debugger.....	689
16.2.3	Expression Builder.....	697
16.3	Toolbars.....	702
16.3.1	Format.....	704
16.3.2	Table	705
16.3.3	Authentic.....	706
16.3.4	RichEdit.....	708
16.3.5	Insert Design Elements.....	708
16.3.6	Design Filter.....	711
16.3.7	Global Resources.....	712
16.3.8	Standard.....	712
16.4	File Menu.....	714
16.4.1	New	714
16.4.2	Open, Reload, Close, Close All.....	720
16.4.3	Save Design, Save All.....	725
16.4.4	Save As.....	730
16.4.5	Export as MobileTogether Design File.....	733
16.4.6	Save Authentic XML Data, Save As.....	733

16.4.7	Save Generated Files.....	734
16.4.8	Deploy to FlowForce.....	736
16.4.9	Web Design.....	738
16.4.10	Properties.....	738
16.4.11	Print Preview, Print.....	743
16.4.12	Most Recently Used Files, Exit.....	744
16.5	Edit Menu.....	745
16.5.1	Undo, Redo, Select All.....	745
16.5.2	Find, Find Next, Replace.....	745
16.5.3	Stylesheet Parameters.....	750
16.5.4	Collapse/Expand Markup.....	751
16.6	Project Menu.....	753
16.6.1	New Project, Open Project, Reload Project.....	754
16.6.2	Close Project, Save Project.....	754
16.6.3	Add Files / Global Resource / URL to Project.....	755
16.6.4	Add Active (and Related) Files to Project.....	756
16.6.5	Add Project and External Folders to Project.....	756
16.7	View Menu.....	760
16.7.1	Toolbars and Status Bar.....	760
16.7.2	Design Sidebars.....	761
16.7.3	Design Filter, Zoom.....	762
16.7.4	Output Previews.....	762
16.8	Insert Menu.....	763
16.8.1	Contents.....	763
16.8.2	Rest of Contents.....	764
16.8.3	RichEdit.....	764
16.8.4	Form Controls.....	766
16.8.5	DB Control.....	766
16.8.6	Auto-Calculation.....	767
16.8.7	Date Picker.....	768
16.8.8	Paragraph, Special Paragraph.....	769
16.8.9	Barcode.....	769
16.8.10	Image.....	770
16.8.11	Horizontal Line.....	772
16.8.12	Table.....	773

16.8.13	Bullets and Numbering.....	773
16.8.14	Bookmark.....	775
16.8.15	Hyperlink.....	776
16.8.16	Footnote.....	777
16.8.17	Condition, Output-Based Condition.....	779
16.8.18	Disabled.....	781
16.8.19	Template.....	781
16.8.20	User-Defined Template.....	782
16.8.21	Variable Template.....	783
16.8.22	Design Fragment.....	784
16.8.23	Layout Container, Layout Box, Line.....	784
16.8.24	Table of Contents.....	784
16.8.25	New Document.....	785
16.8.26	Page / Column / Document Section.....	785
16.8.27	User-Defined Item.....	786
16.9	Enclose With Menu.....	787
16.9.1	Template.....	787
16.9.2	User-Defined Template.....	788
16.9.3	Variable Template.....	788
16.9.4	Paragraph, Special Paragraph.....	788
16.9.5	Bullets and Numbering.....	789
16.9.6	Bookmarks and Hyperlinks.....	790
16.9.7	Condition, Output-Based Condition.....	790
16.9.8	Disabled.....	792
16.9.9	TOC Bookmarks and TOC Levels.....	792
16.9.10	New Document.....	793
16.9.11	User-Defined Element.....	793
16.10	Table Menu.....	794
16.10.1	Insert Table, Delete Table.....	794
16.10.2	Add Table Headers, Footers.....	795
16.10.3	Append/Insert Row/Column.....	795
16.10.4	Delete Row, Column.....	796
16.10.5	Join Cell Left, Right, Below, Above.....	796
16.10.6	Split Cell Horizontally, Vertically.....	796
16.10.7	View Cell Bounds, Table Markup.....	797

16.10.8	Table Properties.....	797
16.10.9	Edit CALS/HTML Tables.....	798
16.10.10	Vertical Alignment of Cell Content.....	798
16.11	Authentic Menu.....	799
16.11.1	Edit Authentic Scripts.....	799
16.11.2	Custom Toolbar Buttons.....	800
16.11.3	Check Macro References.....	803
16.11.4	Auto-Add Date Picker.....	804
16.11.5	Auto-Add DB Controls.....	804
16.11.6	Reload Authentic View, Validate XML.....	804
16.11.7	Select New Row with XML Data for Editing.....	805
16.11.8	Define XML Entities.....	806
16.11.9	View Markup.....	806
16.11.10	RichEdit.....	807
16.11.11	(Dynamic Table) Row Commands.....	807
16.12	Database Menu.....	809
16.12.1	Query Database.....	809
16.12.2	Edit DB Filter, Clear DB Filter.....	810
16.13	Properties Menu.....	811
16.13.1	Edit Bullets and Numbering.....	811
16.13.2	Predefined Value Formatting Strings.....	811
16.14	Tools Menu.....	814
16.14.1	Spelling.....	814
16.14.2	Spelling Options.....	815
16.14.3	Global Resources.....	818
16.14.4	Active Configuration.....	819
16.14.5	Schema Manager.....	820
16.14.6	Customize.....	835
16.14.7	Restore Toolbars and Windows.....	839
16.14.8	Options.....	839
16.15	Window Menu.....	847
16.16	Help Menu.....	848
16.16.1	Help	848
16.16.2	Activation, Order Form, Registration, Updates.....	848
16.16.3	Other Commands.....	851

17 Programmers' Reference 853

17.1	Scripting Editor.....	854
17.1.1	Creating a Scripting Project.....	854
17.1.2	Built-in Commands.....	866
17.2	Application API.....	877
17.2.1	Overview.....	878
17.2.2	Interfaces.....	898
17.2.3	Enumerations.....	996
17.3	ActiveX Integration.....	1002
17.3.1	Prerequisites.....	1002
17.3.2	Adding the ActiveX Controls to the Toolbox.....	1003
17.3.3	Integration at Application Level.....	1005
17.3.4	Integration at Document Level.....	1007
17.3.5	ActiveX Integration Examples.....	1010
17.3.6	Command Reference.....	1024
17.3.7	Object Reference.....	1037

18 Appendices 1059

18.1	XSLT and XQuery Engine Information.....	1060
18.1.1	XSLT 1.0.....	1060
18.1.2	XSLT 2.0.....	1060
18.1.3	XSLT 3.0.....	1062
18.1.4	XQuery 1.0.....	1063
18.1.5	XQuery 3.1.....	1066
18.2	XSLT and XPath/XQuery Functions.....	1068
18.2.1	Altova Extension Functions.....	1069
18.2.2	Miscellaneous Extension Functions.....	1147
18.3	Datatypes in DB-Generated XML Schemas.....	1166
18.3.1	ADO.....	1166
18.3.2	MS Access.....	1167
18.3.3	MS SQL Server.....	1168
18.3.4	MySQL.....	1168

18.3.5	ODBC.....	1169
18.3.6	Oracle.....	1170
18.3.7	Sybase.....	1171
18.4	Technical Data.....	1172
18.4.1	OS and Memory Requirements.....	1172
18.4.2	Altova Engines.....	1172
18.4.3	Unicode Support.....	1173
18.4.4	Internet Usage.....	1173
18.5	License Information.....	1174
18.5.1	Electronic Software Distribution.....	1174
18.5.2	Software Activation and License Metering.....	1175
18.5.3	Altova End-User License Agreement.....	1176

Index

1177

1 Introduction

[Altova StyleVision 2024 Professional Edition](#) is an application for graphically designing and editing StyleVision Power Stylesheets, available in 64-bit and 32-bit versions. StyleVision® runs on Windows 10, Windows 11, and Windows Server 2016 or newer. Some functionality of StyleVision and [Altova MissionKit](#) can be integrated with applications of the Microsoft Office suite (MS Access, MS Excel, MS Word), version 2007 or newer.



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, and Text 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 and RTF (Rich Text Format, used by word processing applications such as MS Word).
- To generate, directly from within StyleVision, **HTML, RTF, and Text 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.

Last updated: 8 April 2024

1.1 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](#)¹⁰⁰², and StyleVision an integrated in [Visual Studio](#)⁶⁷⁰ and [Eclipse](#)⁶⁷³.

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](#)⁴¹⁶ 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.
- HTML documents can be [converted to XML](#)⁴⁷².

Interface

Given below are some general GUI features:

- [Multiple SPS designs](#)³¹ can be open simultaneously, with one being active at any given time. Each SPS design is shown in a separate tab.
- [Template filters](#)⁷¹¹ 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](#)³³: Markup tags in Design View can be hidden and collapsed, thus freeing up space in Design View.
- While designing the SPS, [Authentic View](#)³⁵, [output views](#)³⁶ 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](#)²⁰³ or [source DB](#)²⁰⁸, the source document can be edited directly in the Authentic View of StyleVision.

Databases

The following DB features are supported:

- [DB reports](#)⁴⁸⁸ can either be viewed in StyleVision or saved as HTML and RTF files.
- [IBM DB2 databases](#)⁴⁹², which contain XML columns, are supported.
- A [DB can be queried](#)⁵⁹⁹ 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](#)¹⁰⁵ are supported.
- In the Enterprise and Professional Editions, [multiple output formats](#)¹¹¹ (HTML, RTF, and Text) are generated from a single SPS design.
- Conditions can be set on SPS components to [process them differently for different outputs](#)²⁸⁴. 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](#)¹¹¹ can be [generated and saved](#)¹¹¹, either directly from within the GUI or via [StyleVision Server](#).
- Altova has developed a special [PXF File format](#)⁴⁸² 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](#)⁴⁷⁹: 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](#)¹²⁵, which you enter in the SPS, and [dynamic text](#)¹²⁵, which is selected from the [source document](#)²⁰².
- [Dynamic content](#)¹²⁵ is inserted in the design by dragging-and-dropping nodes from the [schema source](#)²⁰². 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](#)¹²⁵ can be inserted as text, or in the form of a [data-entry device](#)¹⁷⁴ (such as an [input field](#)¹⁷⁵ or [combo box](#)¹⁷⁷). When inserted as a data-entry device such as a [combo box](#)¹⁷⁷, additional possibilities are available. For example, the value of the node can be selected (by the Authentic View user) from a list of enumerations.
- The [structure of the design](#)²⁰⁰ is specified and controlled in a single [main template](#)²⁷. This structure can be modified by optional templates for individual elements—known as [global templates](#)²⁴⁴ because they can be applied globally for that element.
- [Global templates](#)²⁴⁴ can also be created for individual datatypes, thus enabling processing to be handled also on the basis of types.
- [Multiple Document Output](#)²⁶¹: 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](#)¹³⁷: 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](#)¹⁴⁰: 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](#)¹⁴⁰: 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](#)²⁵⁵ enable the modularization and re-use of templates within an SPS, and also across multiple SPSs (see [modular SPSs](#)²³⁰), in a manner similar to the way functions are used.

- [SPS modules](#)²³⁰ 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](#)²⁵⁹: 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](#)⁴⁴²: 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](#)⁴⁴⁶: The user can define XPath functions which can be used anywhere in the document where XPath functions may be used.
- [Document Sections](#)³⁹³: 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](#)¹⁸⁷: A Layout Container is a block in which Design Elements can be laid out and absolutely positioned within the block.
- [Blueprints](#)¹⁹⁰: 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](#)¹²⁵ is defined once and applied to each relevant node in turn (the employee node in our example).
- Multiple [tables of contents](#)³¹² can be inserted in XSLT 2.0 and 3.0 SPSs.
- Repeating data structures can also be inserted as [dynamic tables](#)¹⁴³. 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](#)²⁹⁶ you select, and the sorted element set is sent to the output (HTML and RTF).
- [Variables](#)³⁰⁷: 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](#)²⁸⁸ on the basis of common data content (for example, the common value of an attribute value) and their positions.
- The [conditional templates](#)²⁸⁰ 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](#)²⁷⁰ 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](#)²⁷⁰ can also be passed to a node in the source document. This procedure is referred to as [updating the XML node](#)²⁷² (with the value of the Auto-Calculation).
- [Additional validation](#)⁴³⁸ 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](#)¹⁸³: 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](#)²⁸⁶: 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](#)¹⁶⁸ 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](#)¹⁴³: 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](#)¹⁶³ can be created: static and dynamic. In a [static list](#)¹⁶³, each list item is defined in the SPS. In a [dynamic list](#)¹⁶⁵, 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](#)³³⁹ 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](#)³³⁹ can be inserted. These serve as anchors that can be linked to with a hyperlink.
- [Parameters](#)³⁰² 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](#)³⁵⁴ 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](#)²⁷⁰.
- [JavaScript functions](#)⁴⁶⁷ can be used in the SPS to provide user-defined functionality for Authentic View and HTML output.
- [Authentic Scripting](#)⁶⁵⁰: Enables additional flexibility and interactivity in Authentic View.
- A number of [predefined HTML formats](#)³⁵⁰ 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](#)³⁷¹.
- Additionally, CSS styles can be defined for HTML selectors at the [global level](#)³⁶⁹ 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](#)³⁷⁵. 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](#)³⁹², such as orientation, margins, page-numbering, and headers and footers, can be specified in the SPS.

1.2 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](#)³⁵.

1.3 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](#)²⁴ 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](#)²⁷ is the data in one XML document or DB. The [static content](#)²⁸ 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](#)²⁴. 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](#)¹⁰⁸.

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, Text, and Word 2007-and-higher output (*XSL-FO, Text, and Word 2007-and-higher in Enterprise edition only; RTF and Text 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](#)¹¹¹.

The SPS and output

You can also use StyleVision to directly generate output (*HTML, RTF, Text, XSL-FO, and PDF in Enterprise Edition; HTML and RTF in Professional Edition; and HTML in Basic Edition*). The tabs for [Output Views](#)³⁶ 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](#)⁷³⁴ 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

1.4 Setting up StyleVision

Altova StyleVision runs on Windows 10, Windows 11. 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.

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. For [copy-paste from Word documents](#)¹³² (and of content that can be pasted into Word documents, such as Excel tables and HTML page content) Word 2007+ is required
- Microsoft Word 2007-and-higher or Microsoft's Word 2007-and-higher Viewer, for previewing Word 2007-and-higher output in the Word 2007-and-higher Preview tab. Microsoft Word 2003 with compatibility pack can be used for previewing Word 2007-and-higher output, but is sometimes unable to render Word 2007-and-higher files properly.

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.

RaptorXML

[RaptorXML](#)⁶⁶⁵ can be used to transform XML to the required output format.

1.5 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?](#)²³.
- 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](#)²⁶ (*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](#)²⁶ (*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](#)²⁷ in XPath 2.0. In this documentation, we use the newest, XSLT 3.0 and XPath 3.0, terminology.

- absolute XPath** A path expression that starts at the root node of the tree containing the [context node](#)²⁶. 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](#)²⁶.
- context item /** 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).

context node**current node**

The current node is the node being currently processed. The current node is the same as the [context node](#)²⁶ in expressions that do not have sub-expressions. But where there are sub-expressions, the [context node](#)²⁶ may change. Note that the `current()` function is an XSLT function, not an XPath function.

document element

In a well-formed XML document, the outermost element is known as the document element. It is a child of the [document node](#)²⁷, 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.

document node

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'`.

StyleVision-specific terms

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

Blueprint image

A blueprint image is one that is used as the background image of a [layout container](#)²⁶, and would typically be the scan of a form. The SPS design can be modelled on the blueprint image, thus recreating the form design.

dynamic items

Items that originate in XML data sources. Dynamic items may be text, tables, and lists; also images and hyperlinks (when the URIs are dynamic).

global element

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](#)²⁷ can be defined only for global elements.

global template

A global template may be defined for a [global element](#)²⁷. 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](#)²⁷.

Layout container

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.

local template

A local template is the template that defines how an element (global or non-global) is processed within the [main template](#)²⁷. The local template applies to that particular occurrence of the element in the [main template](#)²⁷. Instead of the local template, a [global template](#)²⁷ can be applied to a given occurrence of an element in the [main template](#)²⁷.

main schema

One of the assigned schema sources is designated the main schema; the document node of the [Working XML File](#)²⁸ associated with the main schema is used as the starting point for the [main template](#)²⁷.

main template

The main entry-point template. In StyleVision, this template matches the [document element](#)²⁷ 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](#)²⁷. The [main template](#)²⁷ defines the basic output document structure and defines how the input document/s are to be processed. It can contain [local templates](#)²⁷ and can reference [global templates](#)²⁷.

output	The output produced by processing an XML document with an XSLT stylesheet. Output files that can be generated by StyleVision would be HTML and RTF 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	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).
SPS component	An SPS component can be: (i) a schema node (for example, an element node); (ii) a static SPS component such as an Auto-Calculation ²⁷⁰ or a text string; or (iii) a predefined format ³⁵⁰ (represented in the SPS by its start and end tags).
template	Defined loosely as a set of instructions for processing a node or group of nodes.
Template XML File	A 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.
User-defined element	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	A template that is created for a sequence specified in an XPath expression.
User-defined XML text blocks	XML Text blocks can be freely inserted at any location in the design
Working XML/XBRL File	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 ²⁷). Note that XBRL functionality is available only in the Enterprise edition.
XML document	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.

1.6 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](#)³¹, which provides an overview of the StyleVision GUI.
- A [tutorial](#)⁶³ section, which is a hands-on exercise to familiarize you with StyleVision features.
- [Usage Overview](#)¹⁰², 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](#)¹²⁴, which explains how static (stylesheet-originated) and dynamic (XML document-originated) components are created and edited in the SPS.
- [SPS File Structure](#)²⁰⁰, which shows how an SPS file can be structured and modularized, and describes the handling of StyleVision's templates.
- [SPS File Advanced Features](#)²⁶⁹, 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](#)³⁴⁹, which explains how SPS components are formatted and laid out.
- [SPS File Additional Editing Functionality](#)⁴¹⁵, 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](#)⁴⁸⁸, which explains how databases can be used with SPSs.
- [Authentic View](#)⁶¹⁴, 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](#)⁶⁶¹, which explains how the generation of output files can be automated.
- [StyleVision's integration features](#)¹⁰⁰², which contains the documentation for integrating StyleVision in other applications. There is also information on how to use StyleVision in [Visual Studio](#)⁶⁷⁰ and [Eclipse](#)⁶⁷³.
- A [reference](#)⁶⁸⁰ section containing descriptions of all symbols and commands used in StyleVision.
- [Appendices](#)¹⁰⁵⁹ 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](#)³¹ and [Usage Overview](#)¹⁰² sections first in order to get an overview of StyleVision features and general usage. Doing the [tutorial](#)⁶³ next would provide hands-on experience of creating an SPS. The SPS File sections ([SPS File Content](#)¹²⁴, [SPS File Structure](#)²⁰⁰, [SPS File Advanced Features](#)²⁶⁹, [SPS File Presentation](#)³⁴⁹, [SPS File Additional Functionality](#)⁴¹⁵, [SPS File and Databases](#)⁴⁸⁸) provide detailed descriptions of how to use various StyleVision features. For subsequent reference, the [Reference](#)⁶⁸⁰ section provides a concise description of all toolbar icon, design symbols, and menu commands, organized according to toolbar and menu. The [Authentic View](#)⁶²⁶ section provides information about editing in Authentic View.

File paths in Windows

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 7/8/10/11	C:\Users\ <username>\Documents</username>
-------------------	---

- *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 7/8/10/11	C:\Program Files\Altova\
32-bit version on 64-bit OS	C:\Program Files (x86)\Altova\

Note: StyleVision is also supported on Windows Server 2016 or newer.

Support options

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

1. Check the [Help](#) ¹⁷ 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

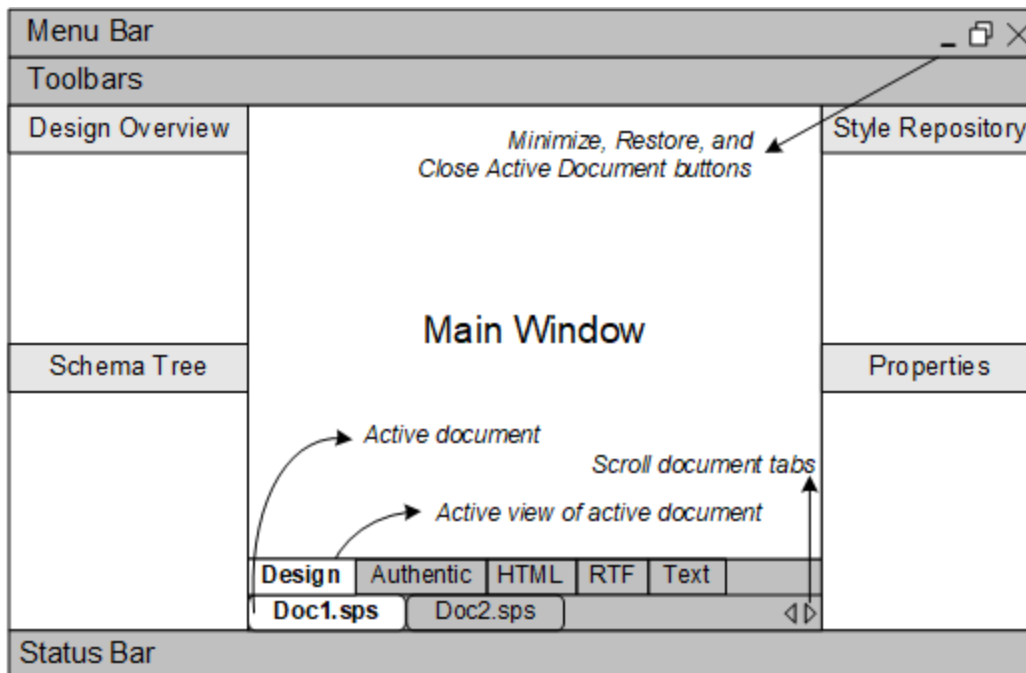
The following abbreviations are used frequently in this documentation:

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

2 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](#)⁶⁸⁰ section. The menu bar also contains the Minimize, Restore, and Close Active Document buttons.
- A **toolbar area**. The various [toolbars](#)⁷⁰² and the command shortcuts in each toolbar are described in the [User Reference](#)⁶⁸⁰ section.
- A tabbed [Main Window](#)³², which displays one or more open SPS documents at a time. In this window, you can [edit the design of the SPS](#)³³, edit the content of [Authentic View](#)³⁵ (in the Authentic eForm tab), and [preview the XSLT stylesheets and output](#)³⁶.
- The [Design Sidebars](#)³⁹—the [Design Overview](#)⁴², [Schema Tree](#)⁴⁴, [Design Tree](#)⁴⁸, [Style Repository](#)⁵¹, [Styles](#)⁵⁴, [Properties](#)⁵⁵, and [Project](#)⁵⁹ 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.

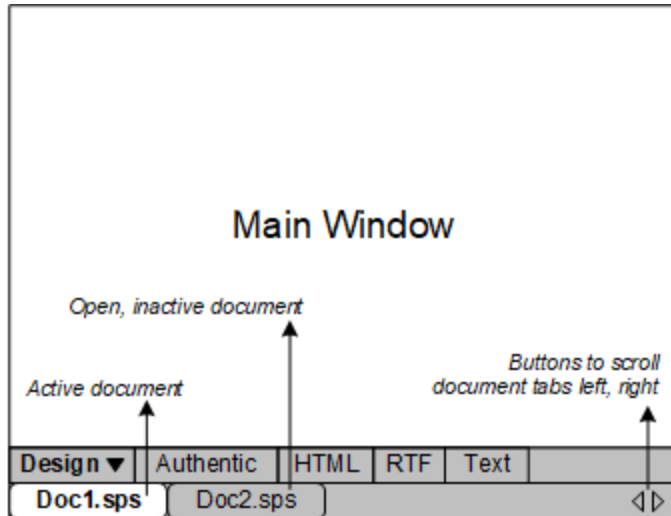


The [Main Window](#)³² and [Design sidebars](#)³⁹ are described in more detail in the sub-sections of this section.

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

2.1 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](#) ⁽⁷²⁰⁾).

Document views

A document is displayed in the following views, one of which can be active at a time:

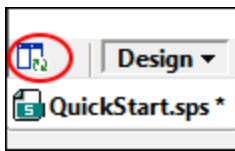
- [Design View](#) ⁽³³⁾, in which you design the SPS and edit JavaScript functions for use in that SPS. The view can be toggled between the design document and the JavaScript Editor by clicking the dropdown menu arrow and selecting Design or JavaScript, as required.
- [Authentic View](#) ⁽³⁵⁾ (Authentic eForm tab), which enables you to immediately see the Authentic View of an XML document (the [Working XML File](#) ⁽²⁸⁾). The SPS is dynamically applied to the [Working XML File](#) ⁽²⁸⁾, thus enabling you to try out Authentic View.
- [Output Views](#) ⁽³⁶⁾ (HTML, Text, and RTF 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.

Output Preview

The Output Preview feature enables you to view the design or Authentic View in the left pane and preview the output in the right pane. This enables you to preview output even as you design and to then modify your design accordingly—not only in terms of presentation but also in terms of content.

You can click the **Output Preview** icon (*circled red in the screenshot below*) to split the Main Window into two vertical panes: (i) [Design View](#)³³ or Authentic View in the left pane and (ii) Output Previews in the right pane. The **Output Preview** icon (*see screenshot below*) is located at the bottom left of the Main Window. In either pane, select the view you want by clicking its tab at the bottom of the pane ([Design View](#)³³ or Authentic View in the left pane; the output preview in the right pane). To switch off Output Preview, click the **Output Preview** icon again.

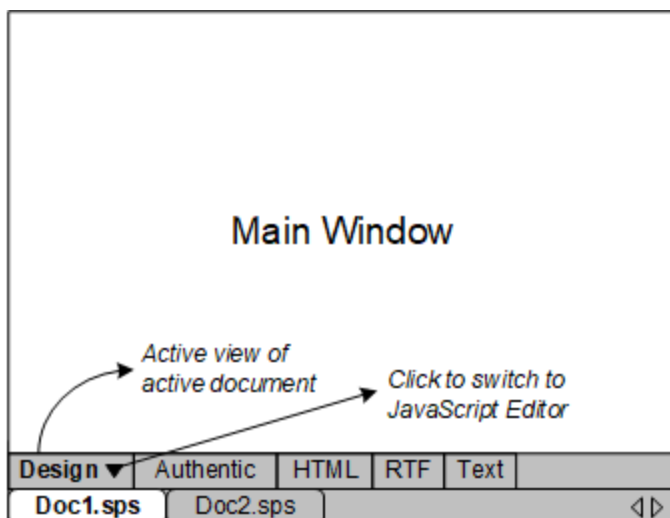


In the output-previews pane:

- Click the icon at extreme left to regenerate the currently selected output preview.
- Click the icon at extreme right to toggle between vertical and horizontal output previews.

2.1.1 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.

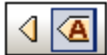


Design View can also be switched to a [JavaScript Editor](#)⁴⁶⁸ or (Authentic) Scripting Editor. In the JavaScript Editor you can create and edit [JavaScript functions](#)⁴⁶⁷ which then become available in the GUI for use in the SPS. In the Scripting Editor you can create and edit scripts for Authentic View. To switch to the [JavaScript Editor](#)⁴⁶⁸ 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](#)⁷¹¹, which are available as [toolbar icons](#)⁷¹¹. 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.

Output Preview

The Output Preview feature enables you to view the design or Authentic View in the left pane and preview the output in the right pane. This enables you to preview output even as you design and to then modify your design accordingly—not only in terms of presentation but also in terms of content.

You can click the **Output Preview** icon (*circled red in the screenshot below*) to split the Main Window into two vertical panes: (i) [Design View](#)³³ or Authentic View in the left pane and (ii) Output Previews in the right pane. The **Output Preview** icon (*see screenshot below*) is located at the bottom left of the Main Window. In either pane, select the view you want by clicking its tab at the bottom of the pane ([Design View](#)³³ or Authentic View in the left pane; the output preview in the right pane). To switch off Output Preview, click the **Output Preview** icon again.



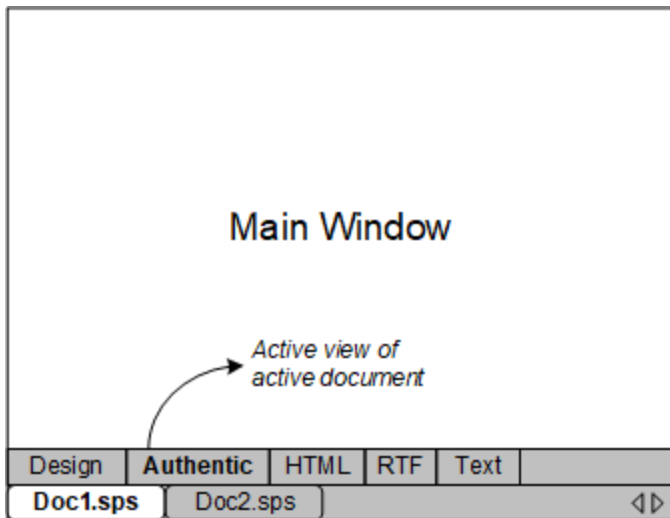
In the output-previews pane:

- Click the icon at extreme left to regenerate the currently selected output preview.
- Click the icon at extreme right to toggle between vertical and horizontal output previews.

2.1.2 Authentic View

In the **Authentic View** tab of the Main Window (the Authentic eForm tab), you can view and edit the [Working XML File](#)²⁸ 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](#)²⁸ **must be assigned**⁴⁴ 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](#)⁷³³).

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

Output Preview

The Output Preview feature enables you to view the design or Authentic View in the left pane and preview the output in the right pane. This enables you to preview output even as you design and to then modify your design accordingly—not only in terms of presentation but also in terms of content.

You can click the **Output Preview** icon (*circled red in the screenshot below*) to split the Main Window into two vertical panes: (i) [Design View](#)³³ or Authentic View in the left pane and (ii) Output Previews in the right pane. The **Output Preview** icon (*see screenshot below*) is located at the bottom left of the Main Window. In either pane, select the view you want by clicking its tab at the bottom of the pane ([Design View](#)³³ or Authentic View

in the left pane; the output preview in the right pane). To switch off Output Preview, click the **Output Preview** icon again.



In the output-previews pane:

- Click the icon at extreme left to regenerate the currently selected output preview.
- Click the icon at extreme right to toggle between vertical and horizontal output previews.

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](#)⁶²⁶. For additional information, please see the Authentic View tutorial in the XMLSpy or Authentic Desktop user manual.

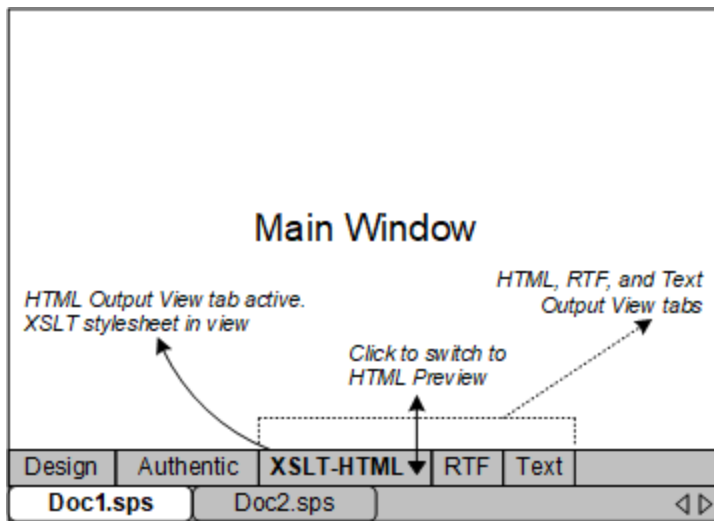
2.1.3 Output Views

There are three **Output View** tabs, one each for HTML, RTF, and Text 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](#)²⁸ with the XSLT stylesheet.

Note: In order for the different views to work correctly, you will need to install programs relevant to each view as described in the section [Setting up StyleVision](#)²⁵.

Note: You can also split the Main Window to show the design in one half and the outputs in the other half. See [Splitt](#)

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](#)⁷⁴⁵) and text from it can be copied to the clipboard (with [Edit | Copy](#)⁷⁴⁵).

Note: The XSLT stylesheets generated from the SPS can be separately generated and saved using the [File | Save Generated Files](#)⁷³⁴ command.

Output preview

The Output preview displays the output produced by transforming the [Working XML File](#)²⁸ 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](#)²⁸ ([File | Save Authentic XML Data](#)⁷³³).

If no [Working XML File](#)²⁸ 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](#)²⁸ 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](#)²⁸.

Note: The output files generated from the SPS can be separately generated and saved using the [File | Save Generated Files](#)⁷³⁴ command.

Output Preview

The Output Preview feature enables you to view the design or Authentic View in the left pane and preview the output in the right pane. This enables you to preview output even as you design and to then modify your design accordingly—not only in terms of presentation but also in terms of content.

You can click the **Output Preview** icon (*circled red in the screenshot below*) to split the Main Window into two vertical panes: (i) [Design View](#)³³ or Authentic View in the left pane and (ii) Output Previews in the right pane. The **Output Preview** icon (*see screenshot below*) is located at the bottom left of the Main Window. In either pane, select the view you want by clicking its tab at the bottom of the pane ([Design View](#)³³ or Authentic View in the left pane; the output preview in the right pane). To switch off Output Preview, click the **Output Preview** icon again.



In the output-previews pane:

- Click the icon at extreme left to regenerate the currently selected output preview.
- Click the icon at extreme right to toggle between vertical and horizontal output previews.

2.2 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](#) ⁴²
- [Schema Tree](#) ⁴⁴
- [Design Tree](#) ⁴⁸
- [Style Repository](#) ⁵¹
- [Styles](#) ⁵⁴
- [Properties](#) ⁵⁵
- [Project](#) ⁵⁹
- [Messages](#) ⁶⁰
- [Find and Replace](#) ⁶¹

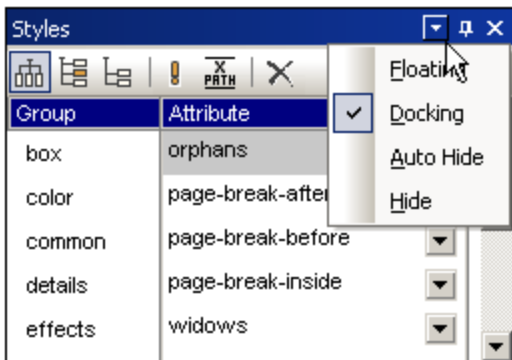
Layout of the views

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, Text, and RTF 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.



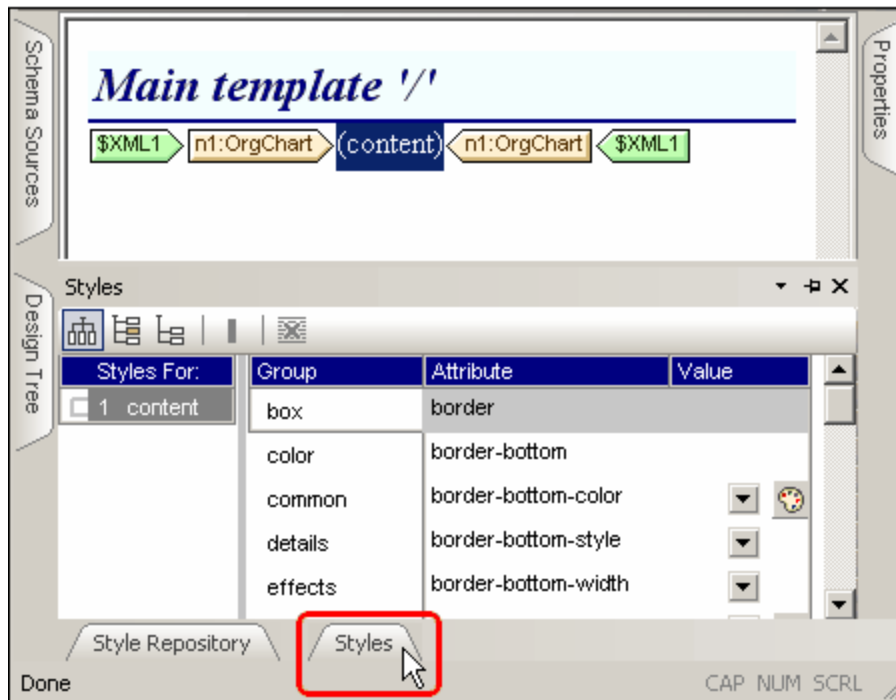
To dock a floating window, right-click the title bar of the floating window and select Docking from the menu that appears; the window will be docked in the position in which it was last docked.

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.

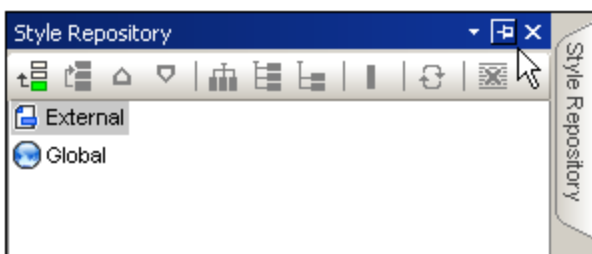
The Auto-Hide feature is useful if you wish to move seldom-used sidebars out of the GUI while at the same time allowing you easy access to them should you need them. This enables you to create more screen space for the Main Window while still allowing easy access to Design sidebar windows.

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](#)⁴⁰, select Auto Hide; (to display the [Menu](#)⁴⁰, right-click the title bar of the window or click the [Menu button](#)⁴⁰ 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](#)⁴⁰, deselect Auto Hide; (to display the [Menu](#)⁴⁰, right-click the title bar of the window or click the [Menu button](#)⁴⁰ 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

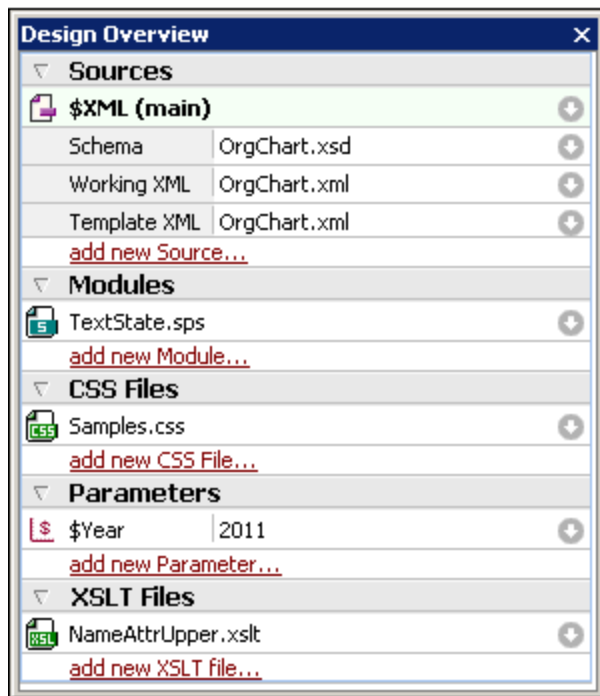
When a sidebar window is hidden it is no longer visible in the GUI, in either its maximized form (docked or floating) or in its minimized form (as a tab at an edge of the GUI, which is done using the [Auto-Hide feature](#)⁴⁰).

To hide a window, click the **Close** button at the top right of a docked or floating window. Alternatively, in the [Menu](#)⁴⁰, select **Hide**; (to display the [Menu](#)⁴⁰, right-click the title bar of the window or click the [Menu button](#)⁴⁰ 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](#)⁷⁶¹ menu. The Design sidebar window is made visible in the position at which it was (docked or floating) when it was hidden.

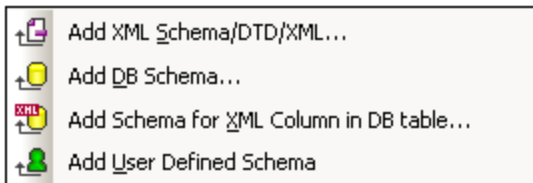
2.2.1 Design Overview

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.



Adding schema sources

Schema sources may be added to an empty SPS. 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; or (iii) a user-defined schema.



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](#) ²⁸.
- The [Template XML File](#) ²⁸.

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](#) ²³⁴ 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.

Modules

The Modules section lists the [SPS modules](#) ²³⁰ 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](#) ²³⁴ 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](#) ²³¹ (objects inside the modules), however, are listed in the [Design Tree sidebar](#) ⁴⁸.

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](#) ³⁶⁵ 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](#) ³⁶⁵ inside the CSS files, however, are listed in the [Style Repository sidebar](#) ⁵¹.

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

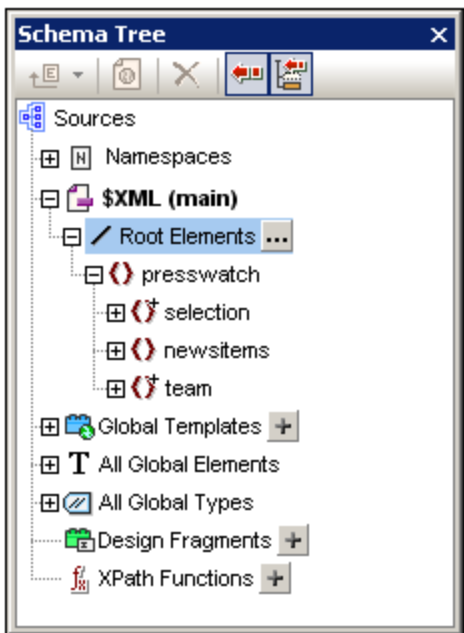
The XSLT Files section lists the XSLT files that have been imported into the SPS. XSLT templates in these XSLT files will be available to the stylesheet as global templates. For a complete description of how this works, see [XSLT Templates](#) ²⁵⁹.

2.2.2 Schema Tree

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


- Select multiple root elements (document elements) for a schema.

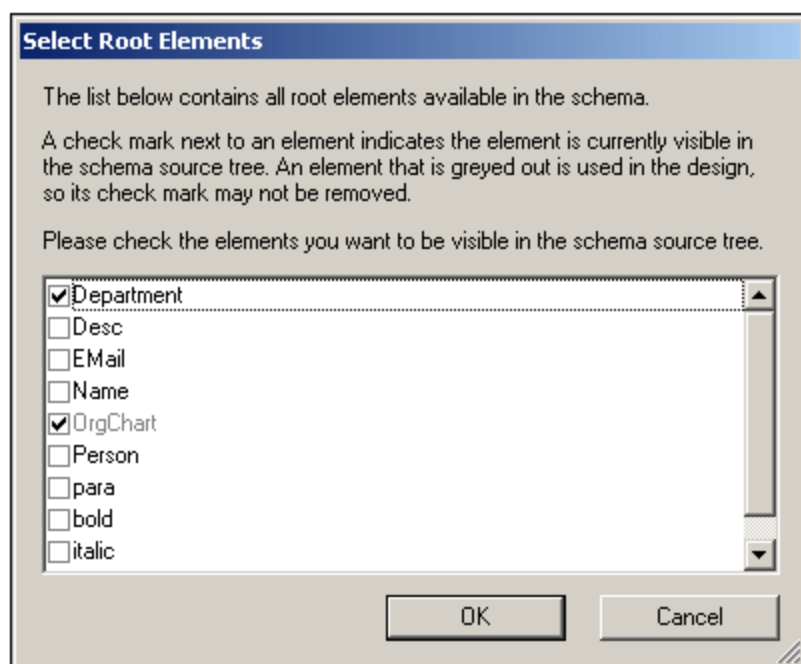
- Drag nodes (elements, attributes, global types) from a schema tree and drop them into the design. These nodes represent the XML content that is to be included in the output.
- View listings of all global elements and types in the schema source. Enables a global element or global type to be created as a global template.
- View a listing of all namespaces used in the SPS.
- Insert and edit [Design Fragments](#)²⁵⁵.
- Insert and and edit user-defined [XPath functions](#)⁴⁴⁶ for the SPS.



Root elements

For each schema, under the \$XML heading, the selected [Root elements](#)²⁷ (or [document elements](#)²⁷) 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](#)²⁰² for an explanation of the possibilities offered by a selection of multiple root elements.




Additionally, all the global elements in the schema are listed under the All Global Elements item. For each global element, a [global template](#)²⁴⁴ can be created.

Global elements and global types

Global elements and global types can be used to create [global templates](#)²⁴⁴ which can be re-used in other templates. Additionally, global types can also be used directly in templates.


Design Fragments

All the [Design Fragments](#)²⁵⁵ 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](#)²⁵⁵ 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](#)⁴⁴⁶, 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](#)²⁰⁹, 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](#)²⁰⁹ 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.



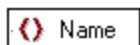
Synchronize tree toggle. When toggled on (icon has border), selecting a node in the tree selects (i) the corresponding node in the design, and (ii) the corresponding node in the Design Tree if the Synchronize Tree icon in the Design Tree window is toggled on. When toggled off, the corresponding node 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.

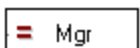
Symbols used in schema trees

Given below is a list of the symbols in schema trees.



Name

Element.



Mgr

Attribute.

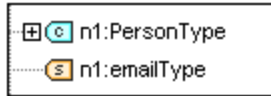


Person

Element with child elements. Double-clicking the element or the +/- symbol to its left causes the element to expand/collapse.



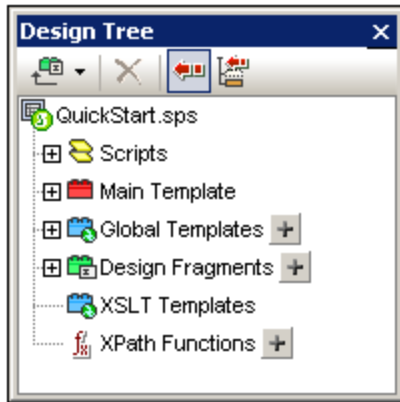
DB Filter applied. Applies only to top-level data table elements in the schema tree.



Global types can be either complex or simple. Complex types are indicated with a cyan icon, simple types with a brown icon.

2.2.3 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](#)⁴⁶⁷, which shows all the JavaScript functions that have been defined for the SPS using the JavaScript Editor of StyleVision.
- [Main Template](#)²⁴⁴, which displays a detailed structure of the main template.
- [Global Templates](#)²⁴⁴, which lists the global templates in the current SPS, as well as the global templates in all included SPS modules.
- [Design Fragments](#)⁵⁰, which shows all the Design Fragments in the design, and enables you to create, edit, rename, and delete them.
- [XSLT Templates](#)⁵¹, which provides the capability to view XSLT templates in imported XSLT files.
- [User-Defined XPath Functions](#)⁴⁴⁶, which enables you to create, edit, rename, and remove your own XPath functions.

Toolbar icons

The following toolbar icons are shortcuts for common Design 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.



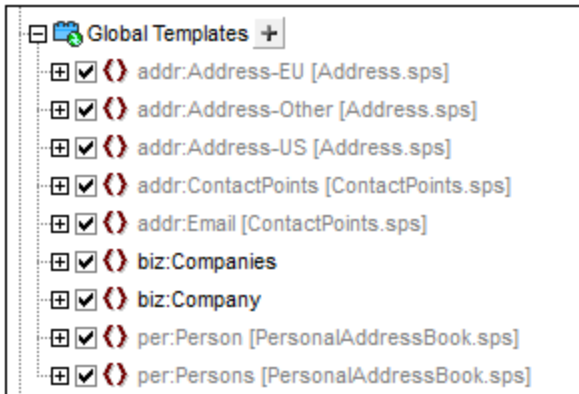
The **Remove** command removes the selected Design Tree item from the Design Tree. **Make Design Fragment** creates a [Design Fragment](#)⁵⁰ in the design and adds an item for it in the Design Tree. **Expand All** expands all the items of the Design Tree.

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

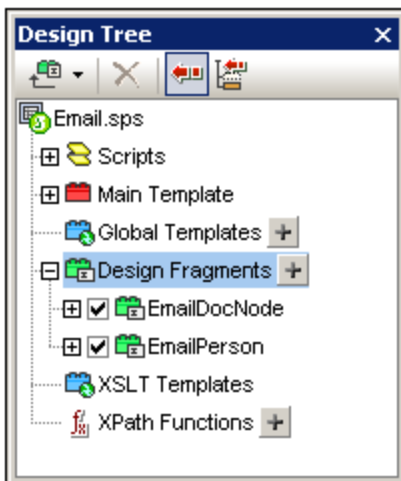
The [Global Templates](#)²⁴⁴ item lists all global templates in the current SPS and in all added SPS modules. Global templates defined in the current SPS are displayed in black, while global templates that have been defined in added modules are displayed in gray (*see screenshot below*). Each global 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

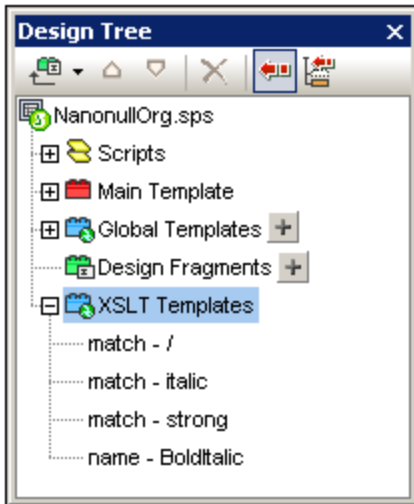
The [Design Fragments](#) ²⁵⁵ item lists all the Design Fragments in the current SPS and in all added SPS modules. Design Fragments defined in the current SPS are displayed in black, while Design Fragments that have been defined in added modules are displayed in gray (see *screenshot below*). Each Design 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

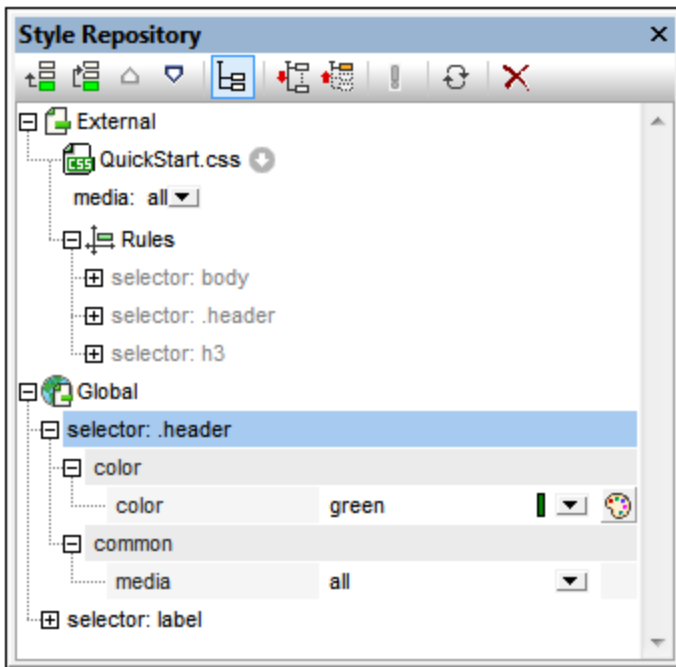
In the Design Tree sidebar (*screenshot below*), the XSLT Templates contained in the imported XSLT file are displayed under the XSLT Templates heading.



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](#)²⁵⁹.

2.2.4 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](#) ³⁶⁴. Note that an external CSS stylesheet can also be added or a stylesheet removed via the [Design Overview sidebar](#) ⁴².


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](#) ³⁶⁴.

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](#) ³⁶⁴.

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  deletes the selected external stylesheet or global style.

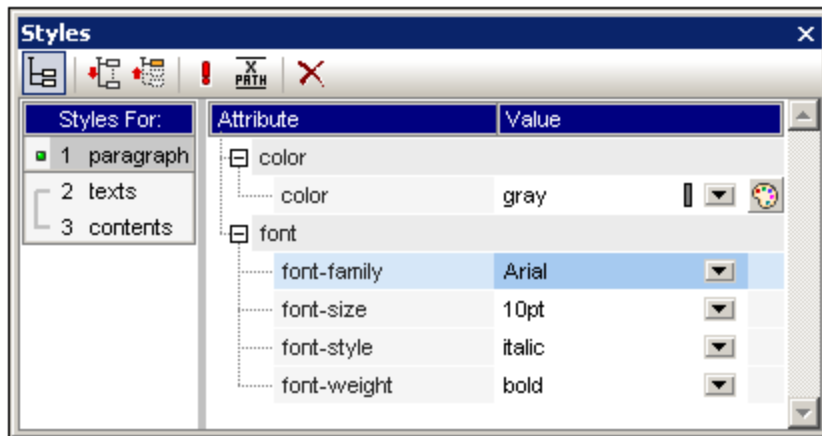
Editing CSS styles in the Style Repository

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](#) ³⁶⁵.
- 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](#) ³⁶⁹.

2.2.5 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](#)⁵¹ 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](#)³⁷¹.
- 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](#)³⁷³. The XPath icon  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** : 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.

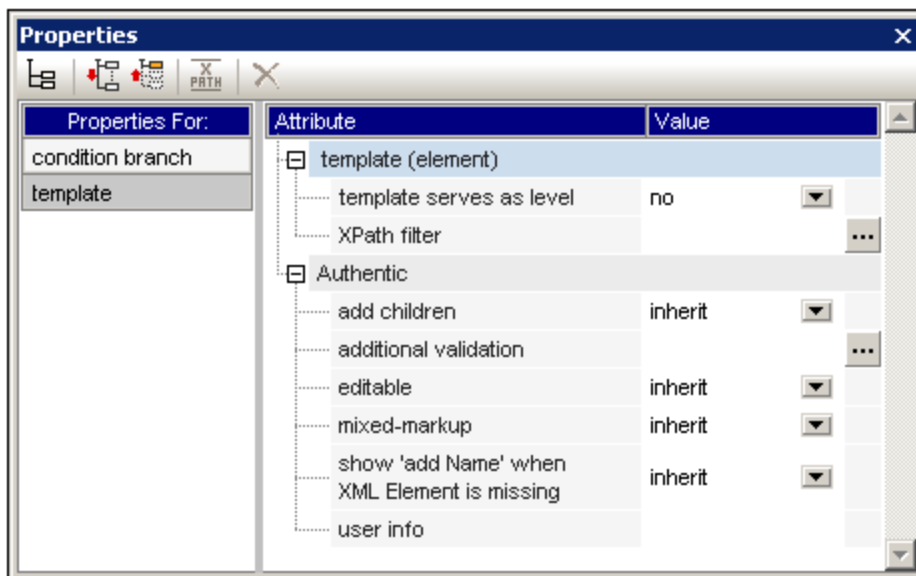
Toggle Important and Reset toolbar icons

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

Clicking the Reset icon  resets the value of the selected property.

2.2.6 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](#) ⁵⁶ 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](#) ⁵⁸ 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.

Component	Property Group
Template	Template; Authentic
Content	Content; Authentic; Common; Event
Text	Text; Common; Event
Auto-Calculation	AutoCalc; Authentic; Common; Event
Condition Branch	When
Data-Entry Device	Authentic; Common; [Data-Entry Device]; Event; HTML

Image	Image; Authentic; Common; Event; HTML
Link	Link; Authentic; Common; Event; HTML
Table	Table; Authentic; Common; Event; HTML; Interactive
Paragraph	Paragraph; Authentic; Common; Event; HTML

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.

Property Group	Description
AutoCalc	These properties are enabled when an Auto-Calculation is selected. The <code>Value Formatting</code> property specifies the formatting ³⁵⁴ of an Auto-Calculation that is a numeric or date datatype. The <code>XPath</code> property specifies the XPath expression that is used for the Auto-Calculation ²⁷⁰ .
Authentic	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 ⁴³³ .
Common	The <i>Common</i> 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: <code>class</code> (a class name), <code>dir</code> (the writing direction), <code>id</code> (a unique ID), <code>lang</code> (the language), and <code>title</code> (a name).
Data-Entry Device	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.
Event	Contains properties that enable JavaScript functions ⁴⁶⁷ to be defined for the following client-side HTML events: <code>onclick</code> , <code>ondblclick</code> , <code>onkeydown</code> , <code>onkeypress</code> , <code>onkeyup</code> , <code>onmousedown</code> , <code>onmousemove</code> , <code>onmouseout</code> , <code>onmouseover</code> , <code>onmouseup</code> .
HTML	Available for the following component types: data-entry devices ¹⁷⁴ ; image ¹⁶⁸ ; link ³³⁹ ; table ¹⁴³ ; paragraphs ³⁵⁰ . Note that there are different types of data-entry devices ¹⁷⁴ and paragraphs ³⁵⁰ , and that tables ¹⁴³ have sub-components. These properties are HTML properties that can be set on the corresponding HTML elements (<code>img</code> , <code>table</code> , <code>p</code> , <code>div</code> , etc). The available properties therefore vary according to the component selected. Values for these properties can be selected using XPath expressions.

In addition, there are component-specific properties for [images](#)¹⁶⁸, [links](#)³⁴¹, [paragraphs and other predefined formats](#)¹²⁷, and [condition branches](#)²⁸³. 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  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](#)³⁷⁵.

Modifying or deleting a property value

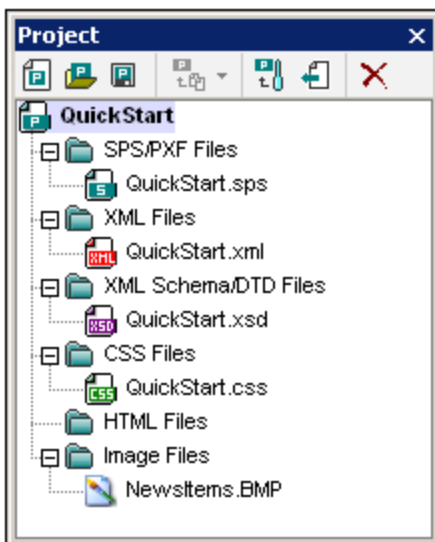
To modify a property value, use any of the applicable methods described in the previous paragraph, [Setting Property Values](#)⁵⁸. To delete a property value, select the property and click the Reset icon  in the toolbar of the Properties sidebar.

2.2.7 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.

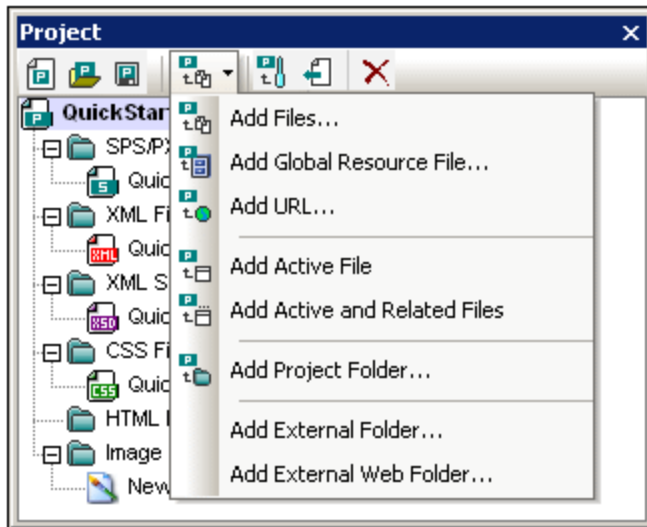
A complete description of how to work with projects is given in the section [Projects in StyleVision](#)¹¹³.

The **Project** sidebar (*screenshot below*) shows the currently active project. Commands in the [Project menu](#)⁷⁵³ apply to the currently active project. The currently active project can be changed by either creating a new project ([Project | New Project](#)⁷⁵⁴) or by opening an existing project ([Project | Open Project](#)⁷⁵⁴). 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](#)⁷⁵⁴; 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](#)⁷⁵³. A useful command is the Add Active and Related Files command, which

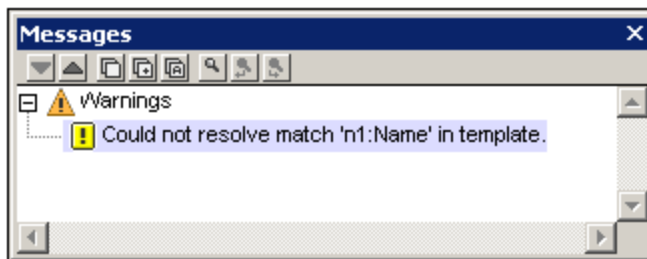
adds to the project all the files related to the currently active SPS. For example, with the tutorial file, `QuickStart.sps`, active, clicking this command will add all related files as shown in the screenshot above.



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](#)¹¹³ for details.

2.2.8 Messages

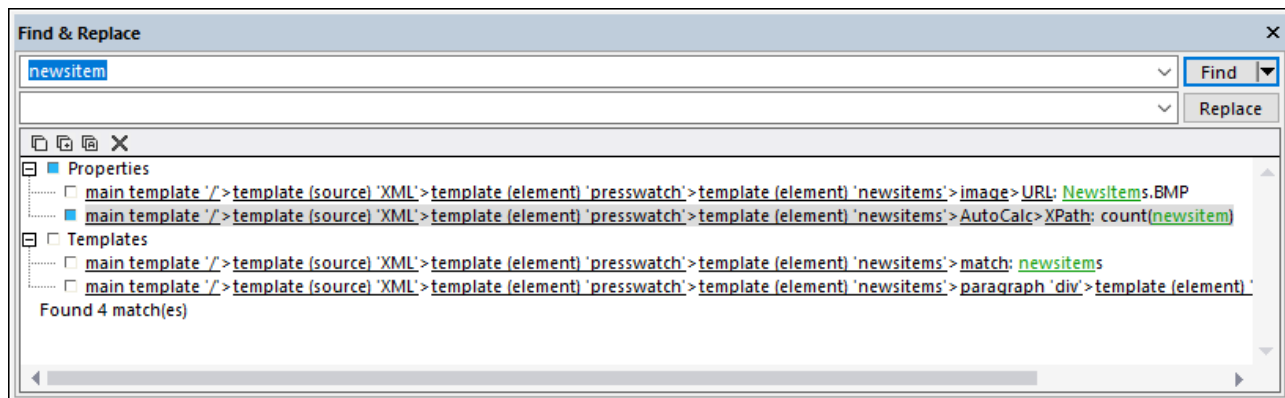
When StyleVision is opened for the first time, the Messages sidebar (*screenshot below*) is displayed below the Main Window of the GUI. To toggle the Messages sidebar on and off, click **View | Messages**.



The Messages sidebar displays warnings in Design View and Authentic View. In Design View, the warnings relate to various aspects related to the SPS document, from a missing Working XML File to errors in the design structure. In Authentic View, the warnings are about the validity of the XML data entered, whether valid according to the underlying schema or according to additional validation criteria.

2.2.9 Find and Replace

The Find & Replace sidebar (*screenshot below*) enables you to search and replace text in Design View. Click the dropdown arrow of the **Find** button (*highlighted blue in the screenshot below*) to select the search options. You can search in text, styles, properties, variables, template matches, and XPath expressions for strings that you enter directly in the *Find* field or construct with regular expressions. All searches carried out in this sidebar apply to Design View. The menu commands [Edit | Find](#)⁷⁴⁵ and [Edit | Replace](#)⁷⁴⁵ sets the focus to this sidebar and places the cursor in the *Find* field, enabling you to proceed with a search in Design View. The results of the search are displayed in the sidebar. You can click on a result to go to the corresponding location in the design. To toggle the Find & Replace sidebar on and off, click **View | Find & Replace**.



For information about searching in other views (JavaScript Editor, Authentic View, and XSLT stylesheets), see the menu-reference topic [Find, Find Next, Replace](#)⁷⁴⁵.

Finding

Enter the term you want to search for in the *Find* field. Then click the dropdown arrow of the **Find** button (*highlighted blue in the screenshot above*) to select the search options. The following options are available:

- *Where to search:* The respective *Include <component>* item should be toggled on for that component to be included in the search.
- *Case and/or whole-word matches:* These are toggle options.
- *Regular expressions:* Your entry will be treated as a regular expression. For a description of how to use regular expressions, see the menu-reference topic [Find, Find Next, Replace](#)⁷⁴⁷.

Results

The results are organized into groups according to the component in which the matched string appears (see *screenshot above*). Each result item is shown as a hierarchical path. You can click any of the links in the hierarchy to go to that item in Design View.

The results pane has a toolbar with icons for the following commands, from left: copy an item or a group of items to the clipboard; clear the results pane.

Replacing

After the results are displayed, you can select one or more of the result items for replacement. The selected item/s will be indicated with a blue bullet (see *screenshot above*) and the **Replace** button will become enabled.

Enter the replacement string in the *Replace* text box and click **Replace**. The replacement is carried out and the blue bullet becomes a green bullet.

3 Quick Start Tutorial

The objective of this tutorial is to take you quickly through the 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](#)²⁹, C:\Documents and Settings\\My Documents\Altova\StyleVision2024\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.

3.1 Creating and Setting Up a New SPS

In this section, you will learn:

- [How to create a new SPS document](#) ⁶⁴
- [How to add a schema source for the SPS](#) ⁶⁶
- [How to select the XSLT version of the SPS](#) ⁶⁷
- [How to assign the Working XML File](#) ⁶⁷
- [How to specify the output encoding](#) ⁶⁷
- [How to save the SPS document](#) ⁶⁷

Files in this section

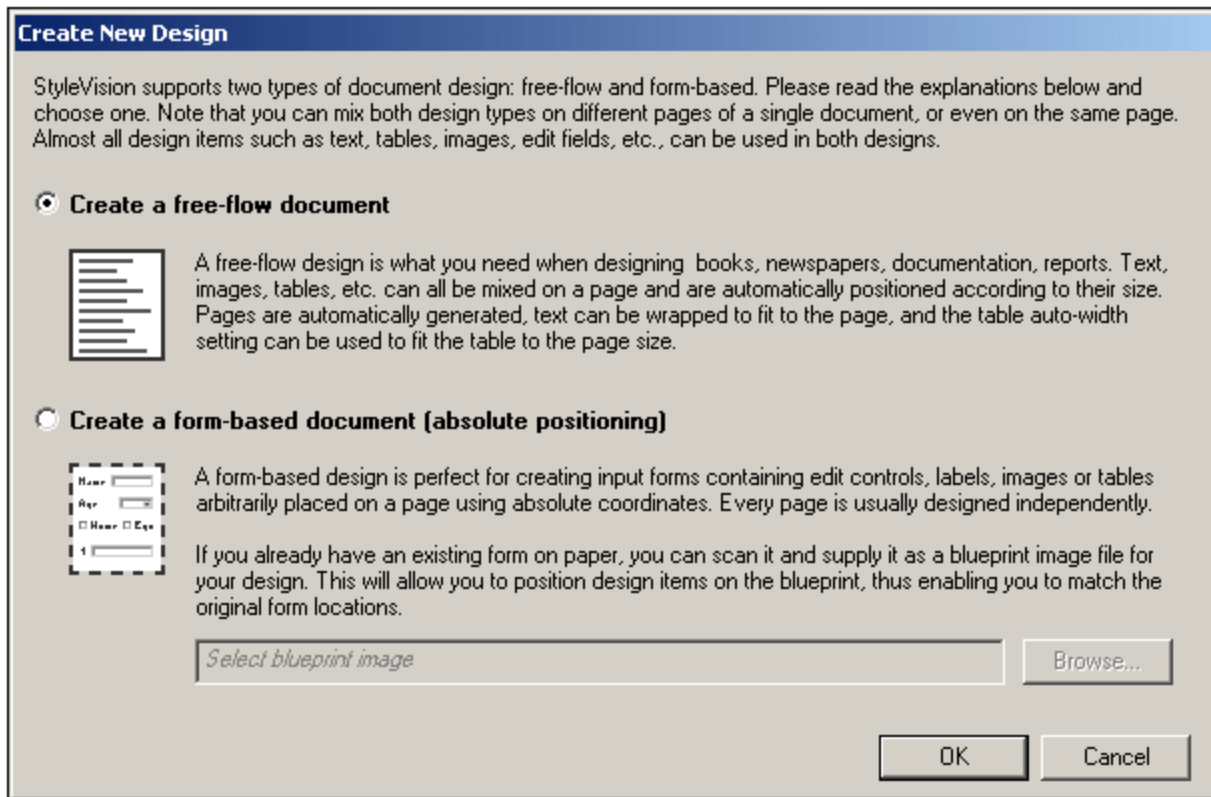
Files referred to in this section are located in the [\(My\) Documents folder](#) ²⁹, C:\Documents and Settings\\My Documents\Altova\StyleVision2024\StyleVisionExamples\Tutorial\QuickStart:

- 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.

Creating a new SPS document

Create a new SPS document by clicking **File | New | New (Empty)** ⁷¹⁴ or select **New (Empty)**  in the dropdown list of the **New icon** ⁷¹²  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).

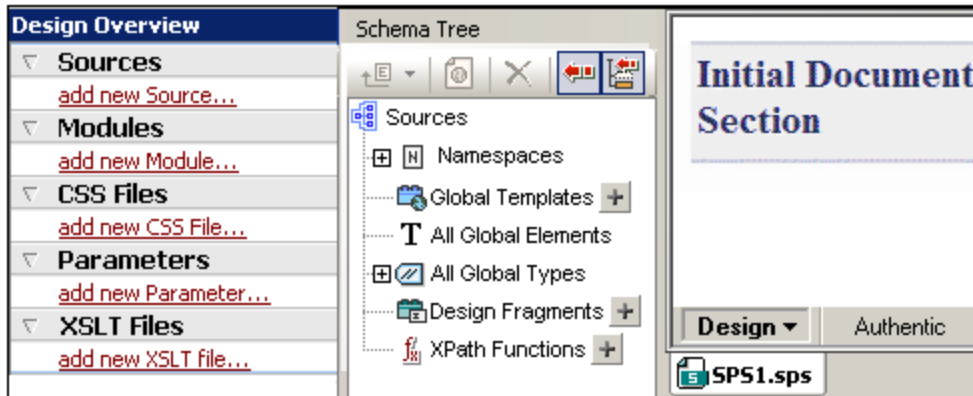


In a free-flowing document design, document content is laid out to fit the output media object or viewer (paper or screen). Items in the document content can only be placed relative to each other, and not absolutely. This kind of design is suited for documents such as reports, articles, and books.

In a form-based document, a single [Layout Container](#)¹⁸⁷ 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](#)¹⁸⁷. 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](#)³³ (screenshot below).

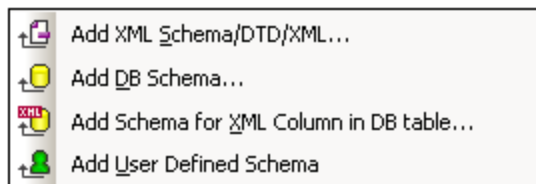


In [Design View](#)³³, an empty main template is displayed. In the [Design Overview](#)⁴² and [Schema Tree](#)⁴⁴ sidebars, there are no schema entries.

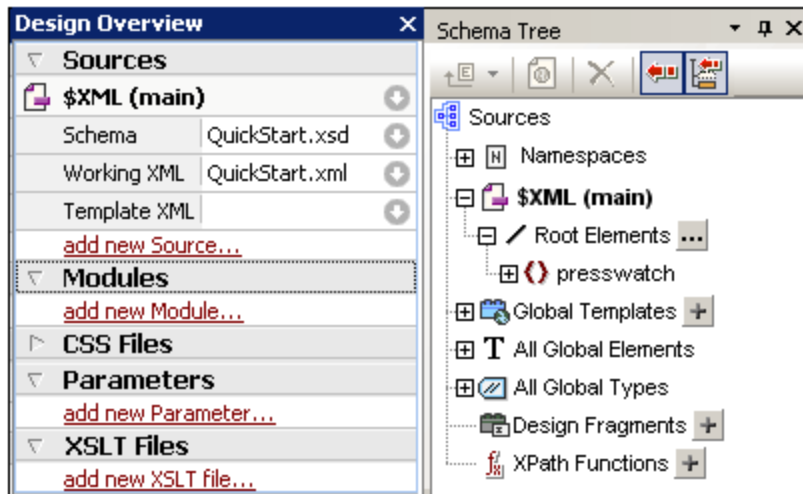
Adding a schema source

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 `QuickStart.xsd` in the [\(My\) Documents folder](#)⁶⁴ (*see above*), 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 `QuickStart.xml` in the [\(My\) Documents folder](#)⁶⁴ (*see above*), and click **Open**. The 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](#)²⁸ and [Template XML File](#)²⁸; (ii) In the Schema Tree sidebar, the Root Elements tree would list the one or more [root elements \(document elements\)](#)²⁷ you select from among the [global elements](#)²⁷ defined in the schema. In the case of this schema, the element `presswatch` is selected by default because it is the one [global element](#)²⁷ in the schema that lies clearly at the top of the hierarchy defined in the schema; (iii) All [global elements](#)²⁷ in the schema are listed in the [All Global Elements tree](#)⁴⁴.

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](#)²⁸ 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](#)²⁸ 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](#)²⁸ is now assigned, and the filename is entered in the Design Overview. Before proceeding, ensure that you have correctly assigned the file `QuickStart.xml`, which is in the [\(My\) Documents folder](#)⁶⁴, as the Working XML File.

Specifying the encoding of output

In the Default Encoding tab of the Options dialog ([Tools | Options](#)⁶³⁹), set the HTML encoding to Unicode UTF-8 and RTF 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](#)⁶⁴. Do this via the menu command [File | Save Design](#)⁷²⁵ or **Ctrl+S**. In the Save Design dialog that pops up, select the *Save as SPS* option, and enter the name of the SPS file to save..

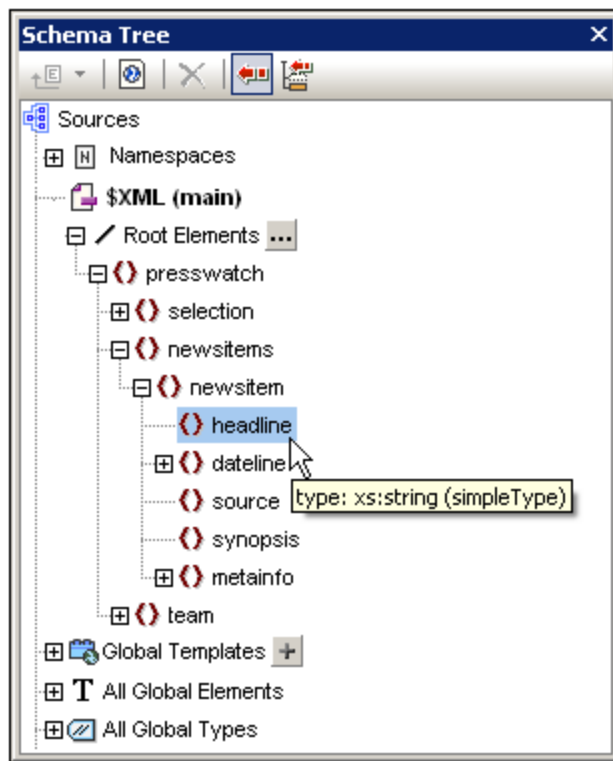
3.2 Inserting Dynamic Content (from XML Source)

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.

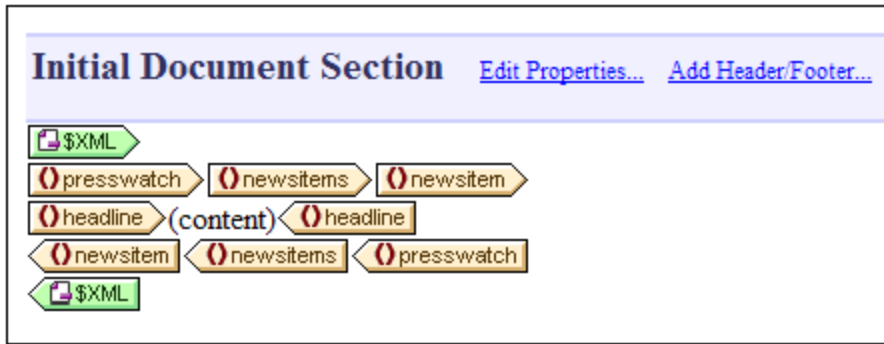
Inserting element contents

In your SPS, do the following:

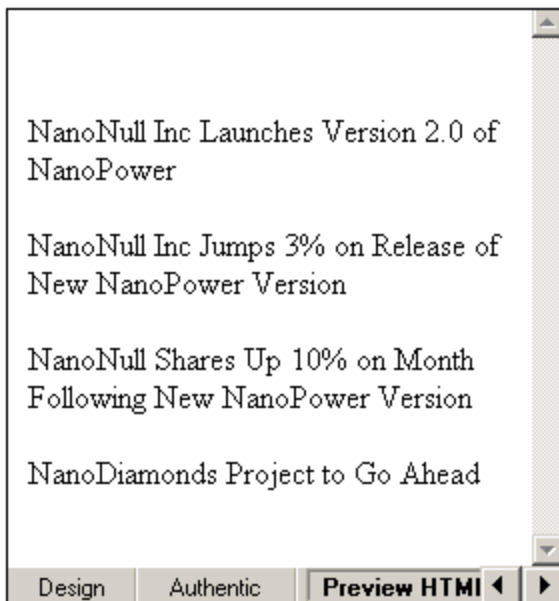
1. In the [Schema Tree sidebar](#)⁴⁴, expand the schema tree up to the children of the `newsitem` element (*screenshot below*).



2. Select the `headline` element (notice that the element's datatype is displayed in a pop-up when you mouseover; *screenshot above*). Drag the element into [Design View](#)³⁹, 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](#) ³⁶ (screenshot below). The HTML preview shows the contents of the `headline` child elements of `newsitem`, each as a text string.



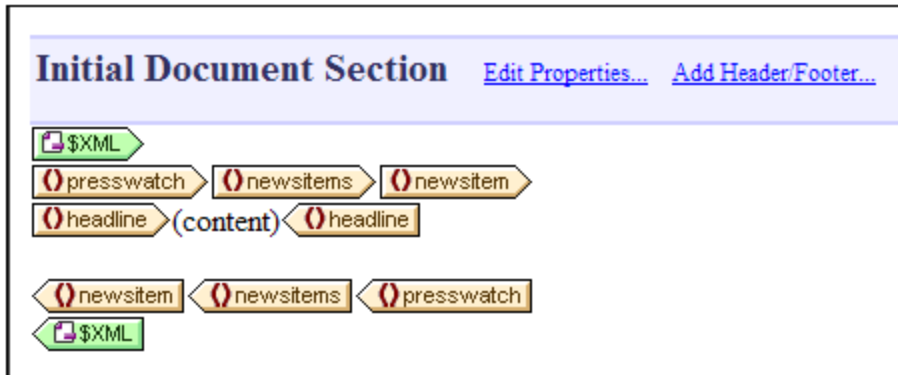
You should also check the preview of [Authentic View](#) ³⁵ and the RTF output.

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](#) ⁷⁰⁸, (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.

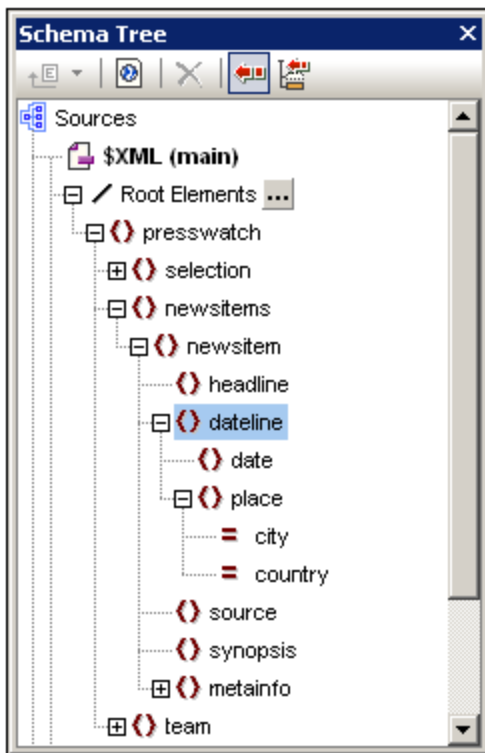
Inserting attribute contents

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).

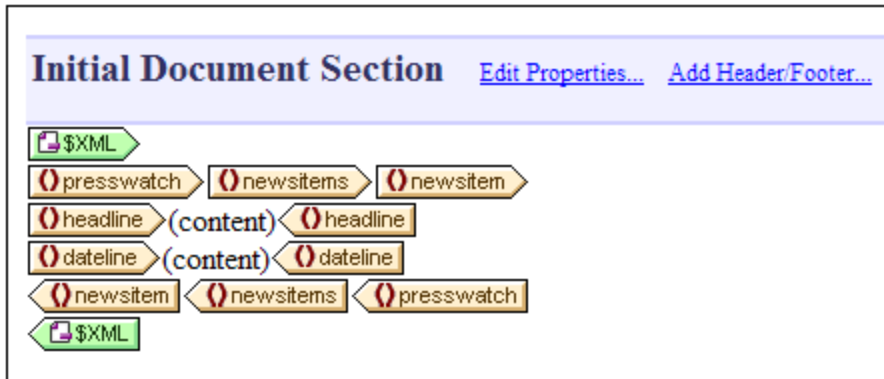


- In the Schema Tree sidebar, expand the `dateline` element (*screenshot below*).



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

- Drag the `dateline` element into the design and drop it at the beginning of the newly created empty line (*screenshot below*).

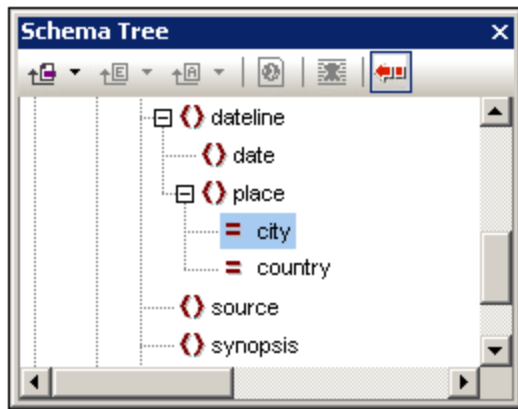


4. Switch to [HTML Preview](#)³⁶ and look carefully at the output of `dateline` (*screenshot below*).



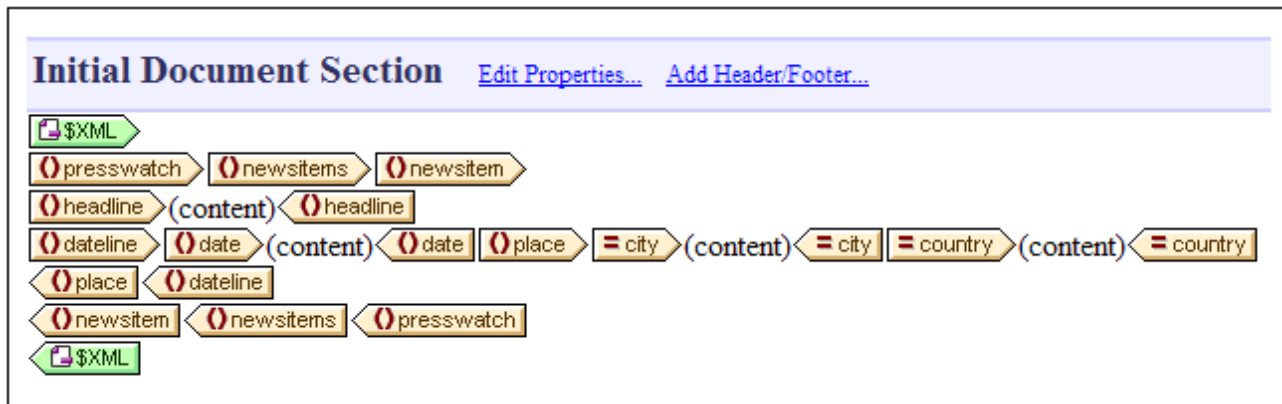
Notice that while the contents of the `date` children of `dateline` elements have been output, no contents have been output for the `place` children of `dateline`. This is because the `place` data is contained in the attributes of the `place` element (in the attributes `city` and `country`) and attribute 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](#)⁸⁰⁴, and toggle it off to deactivate the [auto-addition of the date picker](#)⁴⁶². (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](#)⁴⁴ 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](#)⁴⁴.



7. Drag the @city attribute node into [Design View](#)³³, and drop it (create as contents) just after the end tag of the date element.
8. Drag the @country attribute node into [Design View](#)³³, 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:



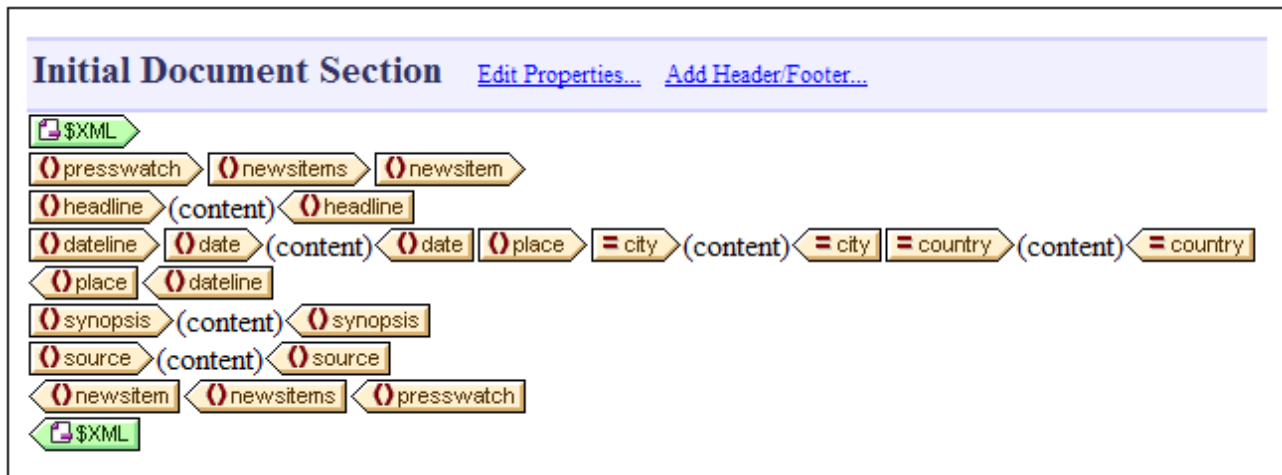
The [HTML Preview](#)³⁶ will look like this:



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](#) ³⁶ to see the output. This is an important point to note: That the order in which nodes

are placed in the [main template](#)²⁷ is the order in which they will appear in the output (see the section, [Templates and Design Fragments](#)²⁴⁴, 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](#)³⁶, 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](#)⁹⁰, you will learn how to create a node as a combo box, and in the section, [Using Global Templates and Rest-of-Contents](#)⁹⁷, how to create a node as a (dynamic) table.

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

3.3 Inserting Static Content

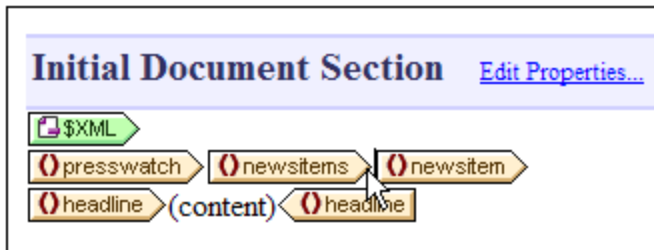
Static content is content you enter or insert directly in the design—as opposed to content that comes from the XML source. A variety of static components can be placed in an SPS design. In this part of the tutorial, you will learn how to insert the following static components:

- [An image](#) ⁷⁵
- [A horizontal line](#) ⁷⁶
- [Text](#) ⁷⁶

Inserting a static image

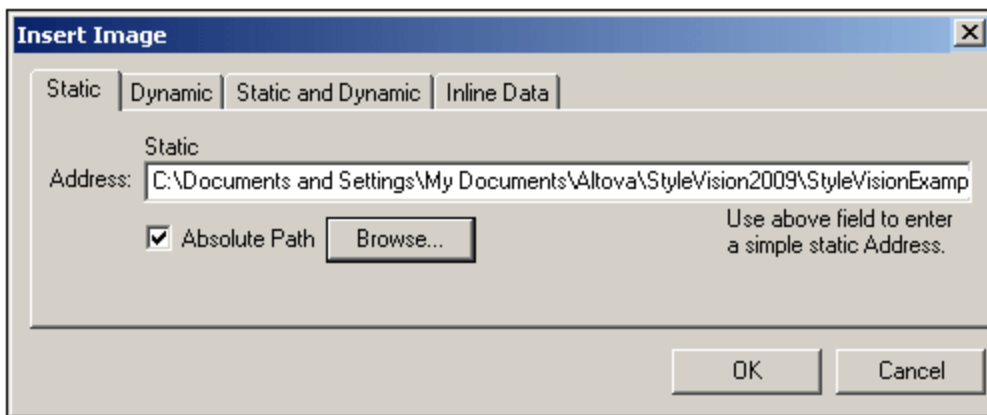
The static image to insert is in the [\(My\) Documents folder](#) ²⁹: C:\Documents and Settings\\My Documents\Altova\StyleVision2024\StyleVisionExamples\Tutorial\QuickStart\NewsItems.BMP. It will be used as the header of the document. To insert this image at the head of the document, do the following:

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



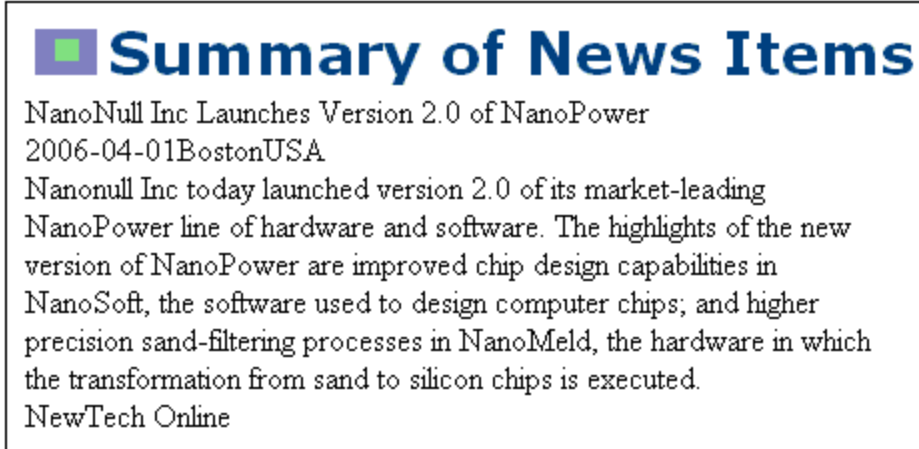
Notice that the cursor is within the `newsitems` element but outside the `newsitem` element. It will therefore be inserted in the output once, at the start of processing of the `newsitems` element (because there is only one `newsitems` element defined in the schema).

2. Right-click, and select [Insert | Image](#) ⁷⁷⁰. The Insert Image dialog pops up (*screenshot below*).



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:



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](#)⁷⁷². A horizontal line is inserted.

Set properties for the line as follows:

1. With the line selected in [Design View](#)³³, in the [Properties sidebar](#)⁵⁵, 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](#)³³, in the [Styles sidebar](#)⁵⁴, 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](#)³⁶.

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 \(from XML Source\)](#)⁶⁸), 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:

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

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*)

The diagram illustrates the structure of the `dateline` element. It shows a sequence of elements: `date`, `content`, `date`, `place`, `city`, `content`, `city`, `country`, and `content`. Static text is added between these elements: a colon and space after the first `date`, a comma and space after the `city` attribute, and a comma and space after the `country` attribute. The final output is: `date : place = city , = country (content)`.

This part of the output will now look like this:

Summary of News Items

NanoNull Inc Launches Version 2.0 of NanoPower
 2006-04-01: Boston, USA

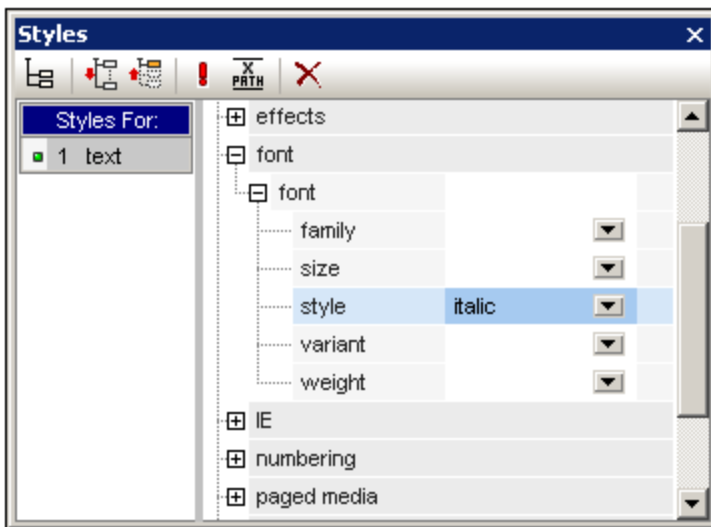
Notice the colon, spacing and comma in the `dateline` output. All of these text items are static text items that were inserted directly in the design.

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*).

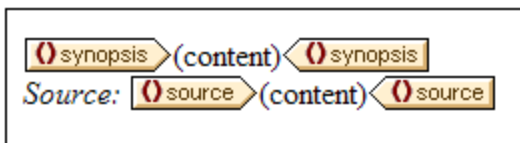
The diagram shows the structure of the `synopsis` element. It consists of `synopsis`, `content`, and `synopsis`. A `source` attribute is added to the `synopsis` element, with the static text "Source: " placed before the start-tag. The final output is: `Source: synopsis (content) synopsis`.

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](#)⁵⁴ (screenshot below), notice that the *1 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:



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.

3.4 Formatting the Content

StyleVision offers a powerful and flexible [styling mechanism](#)³⁶⁴, based on CSS, for formatting components in the design. The following are the key aspects of StyleVision's styling mechanism:

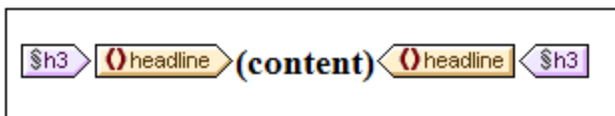
- CSS style rules can be defined for both block components and inline components.
- [Predefined formats](#)³⁵⁰ are block components that have inherent styles and can be used as wrappers for a group of components that need to be treated as a block. The inherent styles of these predefined formats can be 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](#)³⁶⁵ or [global](#)³⁶⁹ style rules.
- You can specify styles at three levels. These are, in increasing order of priority: (i) style rules in [external stylesheets](#)³⁶⁵, (ii) [global style rules](#)³⁶⁹, and (iii) [local style rules](#)³⁷¹. 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 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](#)⁸⁰
- [Assign a component a class attribute](#)⁸¹
- [Define styles in an external CSS stylesheet](#)⁸² and add this stylesheet to the style repository of the SPS
- [Define global style rules](#)⁸²
- Define [local styles for a selection of multiple design components](#)⁸⁴
- Define [local styles for a single component](#)⁸⁴

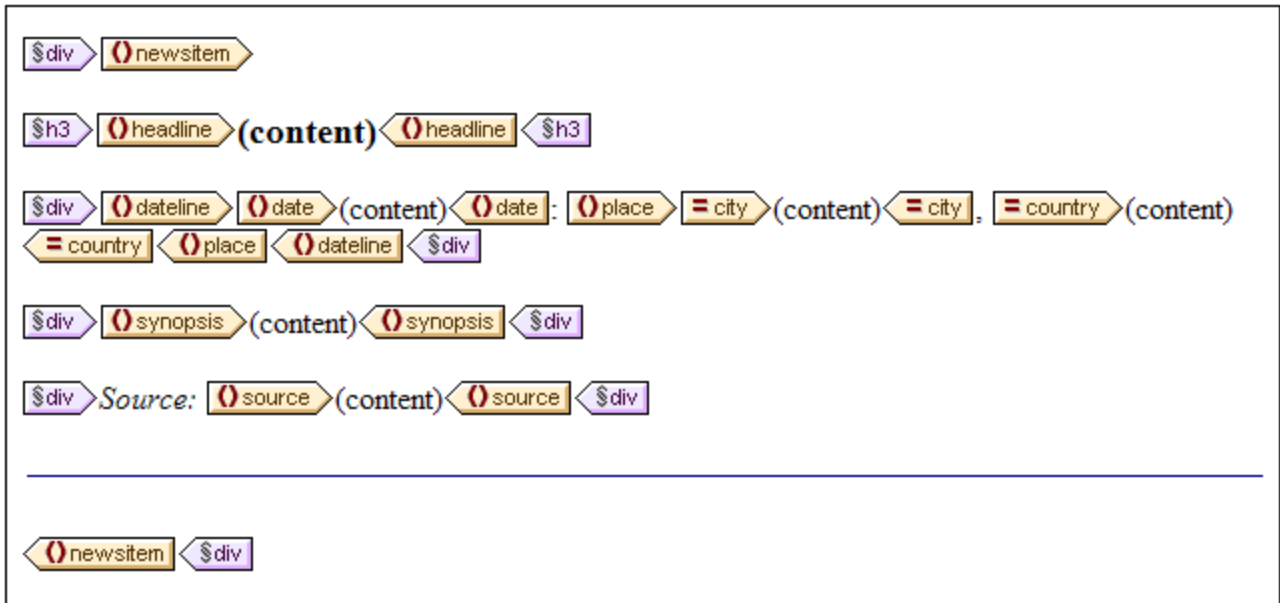
Assigning predefined formats

One reason to assign a [predefined format](#)³⁵⁰ is to give a component the inherent styling of that [predefined format](#)³⁵⁰. 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*).



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.

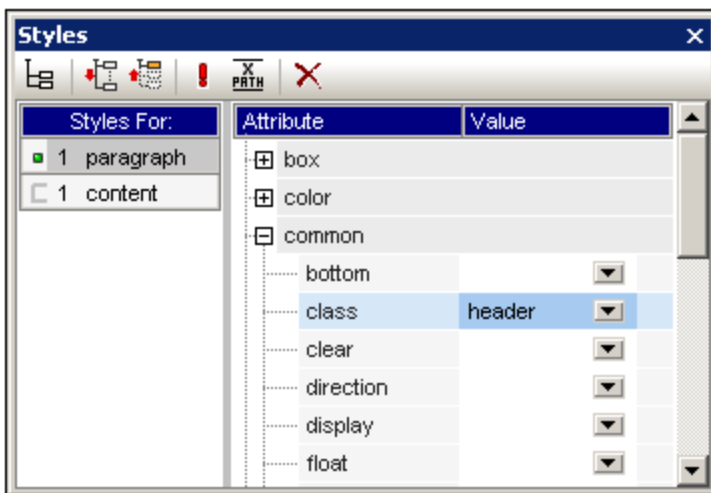


You have now grouped components together in different `div` blocks. [Later in this section](#) ⁸⁴, you will learn how to assign styles to such blocks of grouped components.

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 *1 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 stylesheet 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](#)⁵¹ (in Design View), do the following:

1. Select the `External` item.
2. Click the **Add** button in the toolbar of the [Style Repository sidebar](#)⁵¹. This pops up the Open dialog.
3. **Browse for the file** `C:\Documents and Settings\\My Documents\Altova\StyleVision2024\StyleVisionExamples\Tutorial\QuickStart\QuickStart.css`, which is in the [\(My\) Documents folder](#)²⁹, 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.

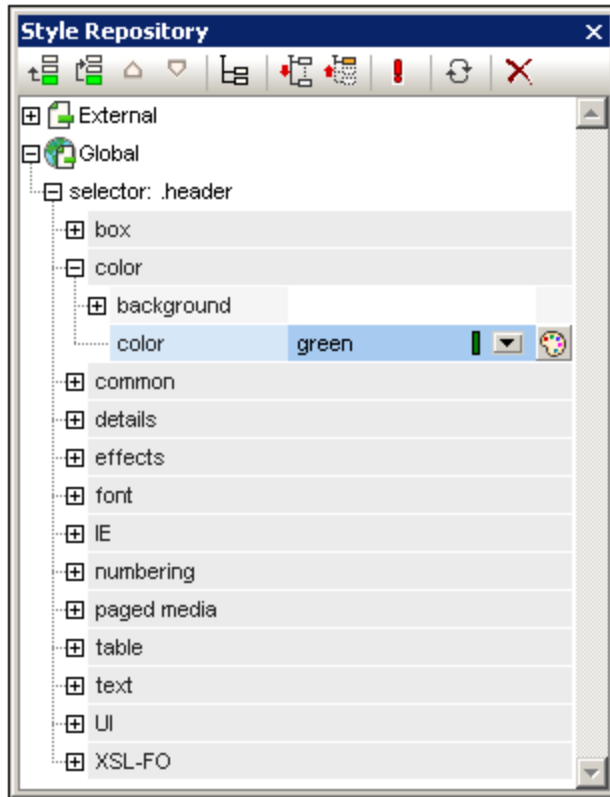
Source: NewTech Online

Defining global style rules

[Global style rules](#)³⁶⁹ can be defined for the entire SPS using CSS selectors. The rules are defined directly in the [Style Repository sidebar](#)⁵¹. Create a global style rule for the `header` class as follows:

1. With [Design View](#)³³ active, in the [Style Repository sidebar](#)³³, 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 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](#)⁵⁴, 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](#)⁵⁴, 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](#)⁵⁴, 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](#)⁵⁴, with the 1 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 1 *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](#)³⁵⁰ 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.

3.5 Using Auto-Calculations

[Auto-Calculations](#)²⁷⁰ 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.

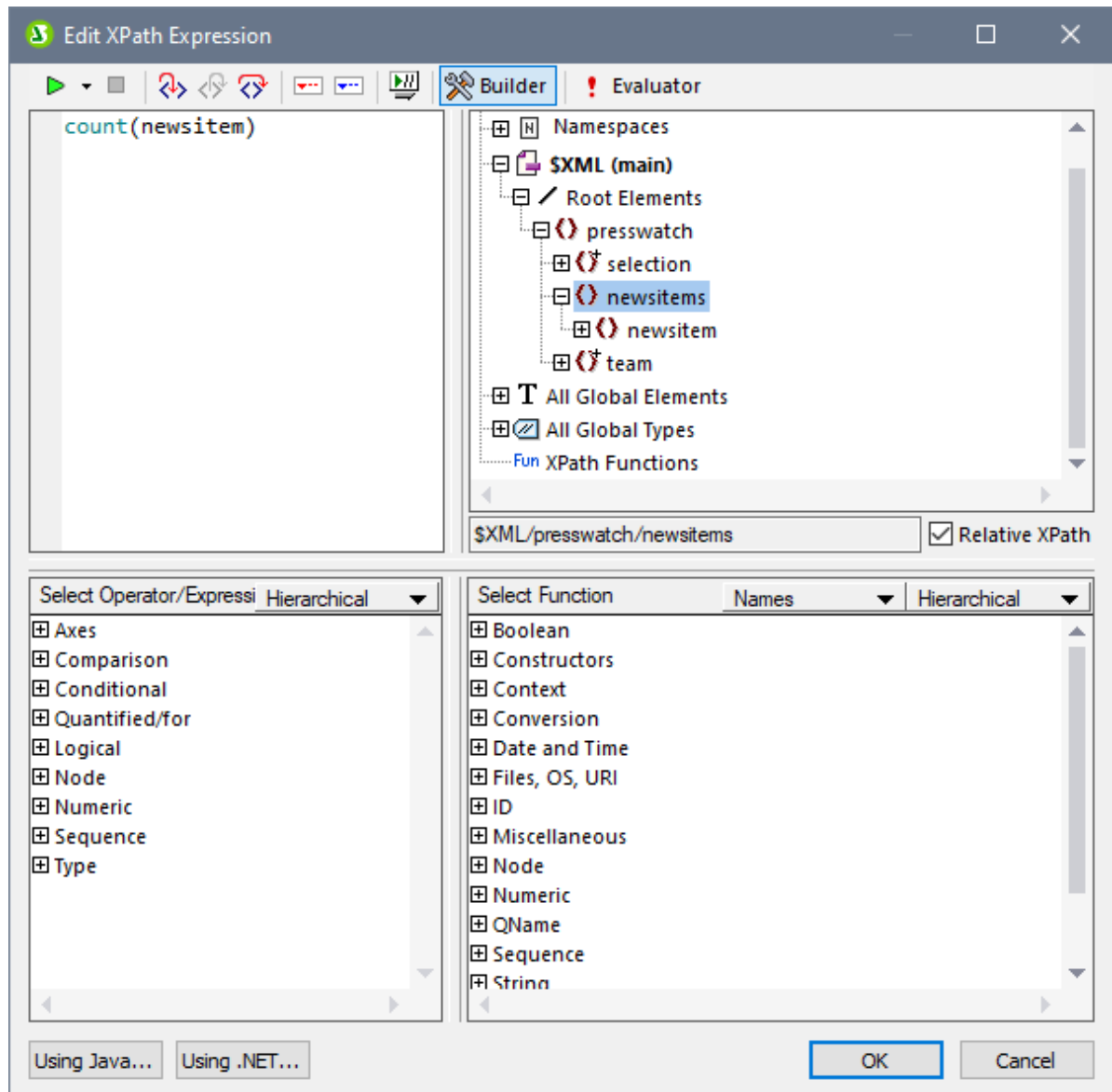
Counting the news item nodes

In the design, do the following:

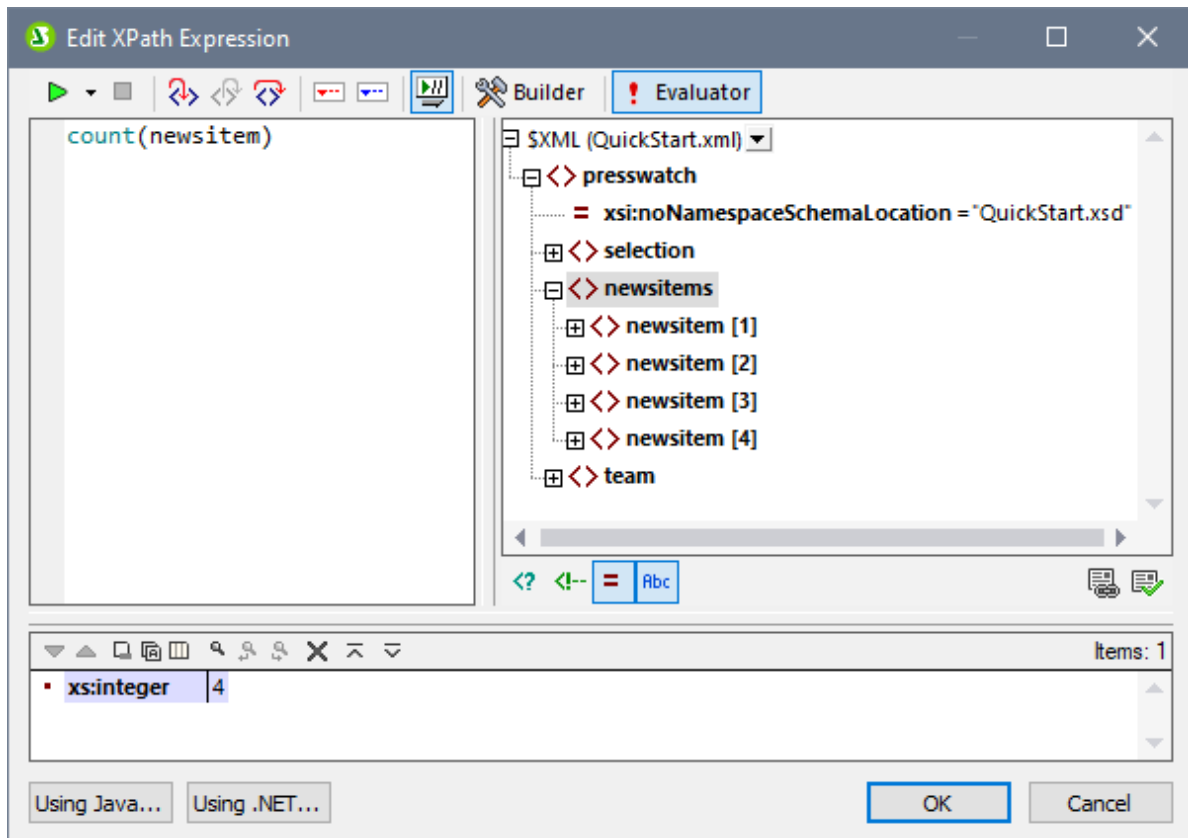
1. Create space, as shown in the screenshot below, for a line of static text (on which the Auto-Calculation will also be placed). Use the **Return** key to add new lines and insert a horizontal line below the space you create (see *screenshot*).



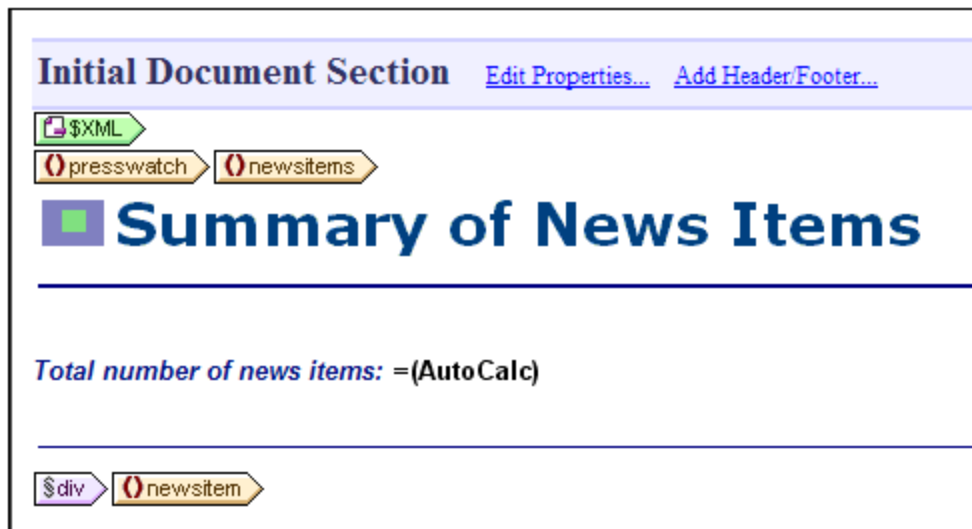
2. Type in the static text "Total number of news items:" as shown in the screenshot above.
3. Apply local styling of your choice to the static text. Do this as described in the section [Formatting the Content](#)⁸⁴.
4. Place the cursor after the colon and select **Insert | Auto-Calculation | Value**. This pops up the [Edit XPath Expression dialog](#)⁶⁸⁵ (*screenshot below*). (Alternatively, you can right-click and select the command in the context menu.)



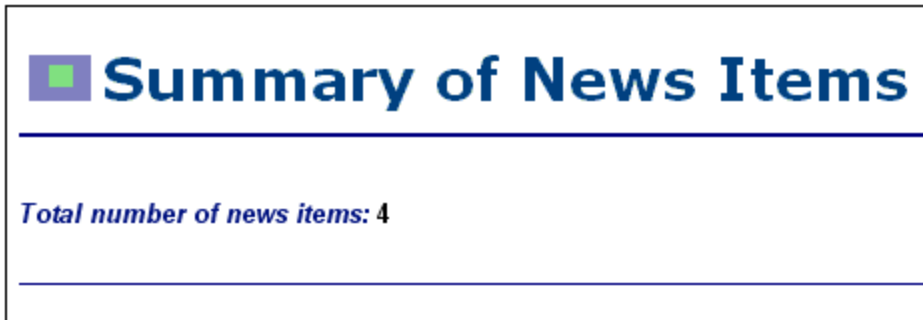
- In the schema tree, note that the context node is `newsitems`, which is highlighted. Now, in the *Expression* text box either type in the expression `count(newstitem)` or build the expression using the entry-helper panes below the Expression text box. (Double-click the `count` function (found in the *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 `newstitem` 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](#)⁶⁸⁵.



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](#) ⁸⁴.



Your HTML output will look like this:

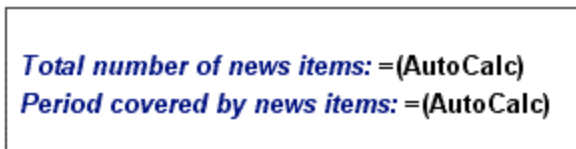


Displaying the period covered by news items

The period covered by all the news items together can be obtained by getting the date of the earliest news item and the date of the latest news item. This can be achieved with XPath expressions like those given below. The first expression below outputs the contents of the `date` node. The second expression is a refinement, outputting just the month and year values in the `date` node. You can use either of these.

- `concat(min(//date), ' to ', max(//date)).`
- `concat(month-from-date(min(//date)), '/', year-from-date(min(//date)), ' to ', month-from-date(max(//date)), '/', year-from-date(max(//date)))`

In the design, insert the static text and Auto-Calculation as shown in the screenshot below. Apply whatever local styling you like.



The HTML preview will look something like this:



After you are done, save the file.

3.6 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.

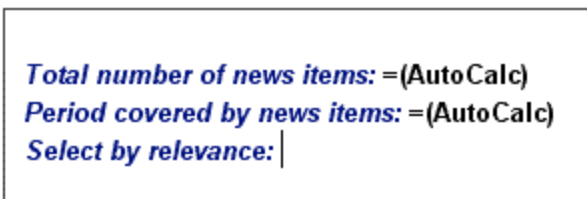
We will proceed as follows:

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

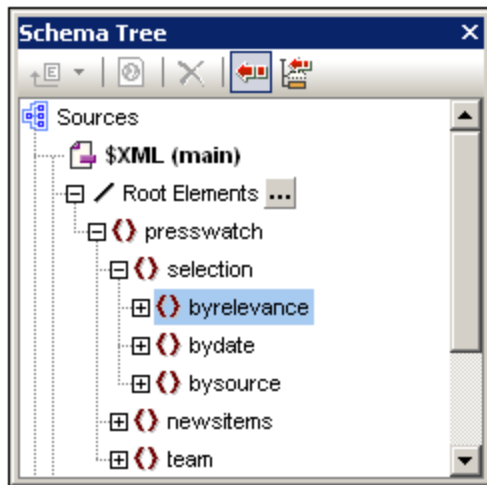
In the XML document, the node that will contain the user selection is `/presswatch/selection/byrelevance`. This is the node you will create as the combo box. Do this as follows:

1. Insert the static text "Select by relevance: " at the head of the document and just below the [second Auto-Calculation](#)⁸⁶ (screenshot below).

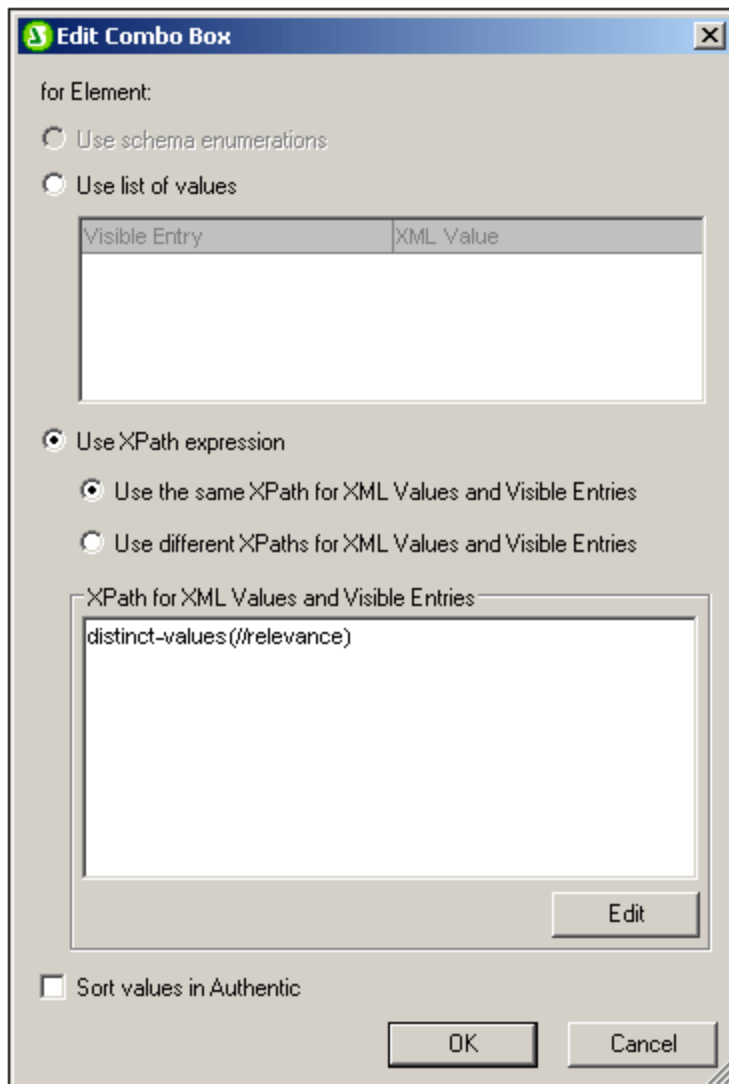


Total number of news items: =(AutoCalc)
Period covered by news items: =(AutoCalc)
Select by relevance: |

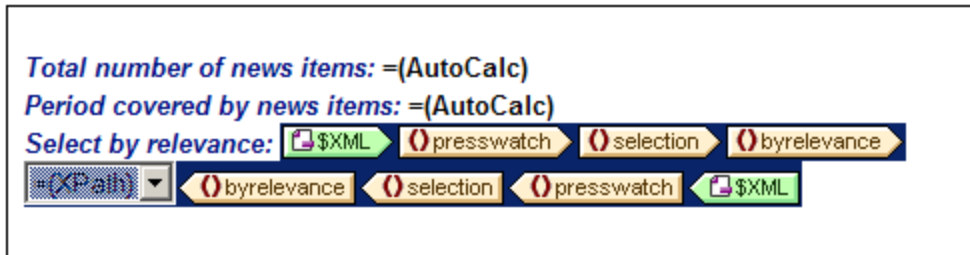
2. Drag the `byrelevance` node from the [Schema Tree sidebar](#)⁴⁴ (screenshot below), and drop it after the newly entered static text.



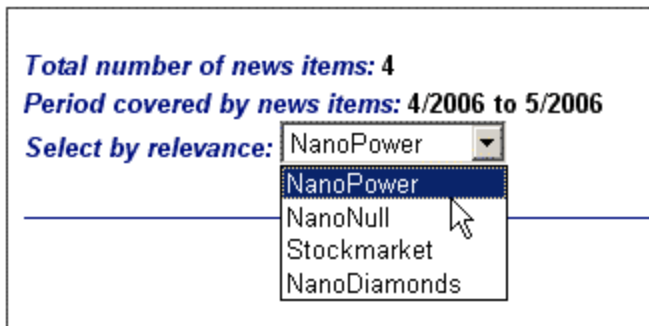
3. In the context menu that appears, select Create Combo Box. This pops up the dialog shown below.



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



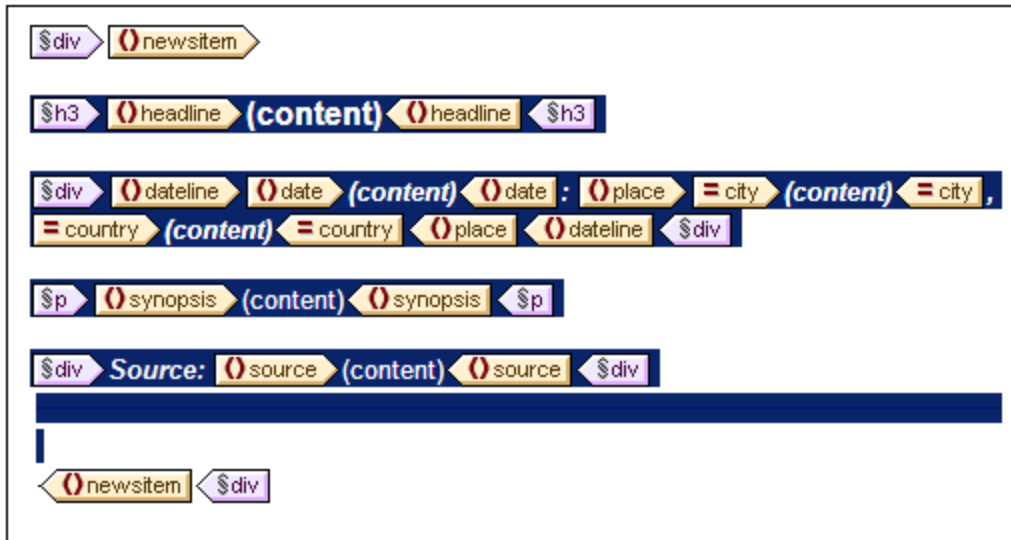
- Switch to [Authentic View](#)³⁵. When you click the dropdown arrow of the combo box, notice that the list contains the unique values of all `relevance` nodes (*screenshot below*). Check this against the XML document. This is a dynamic listing that will be augmented each time a new `relevance` value is added to the XML document.



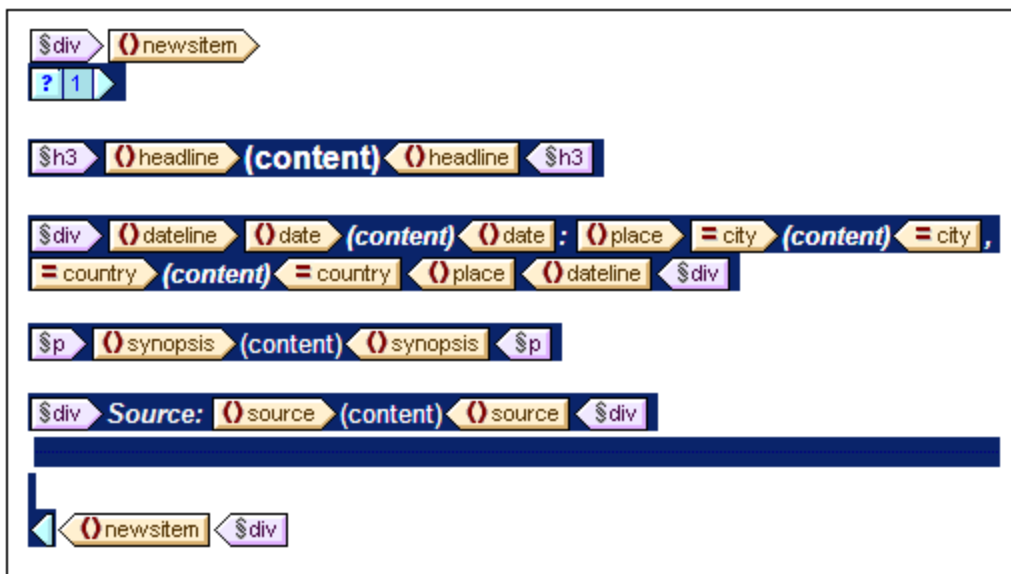
Inserting a condition to display news items having the selected `relevance`

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:

- 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**⁷⁹⁰. This pops up the **Edit XPath Expression dialog**⁶⁸⁵.
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*).

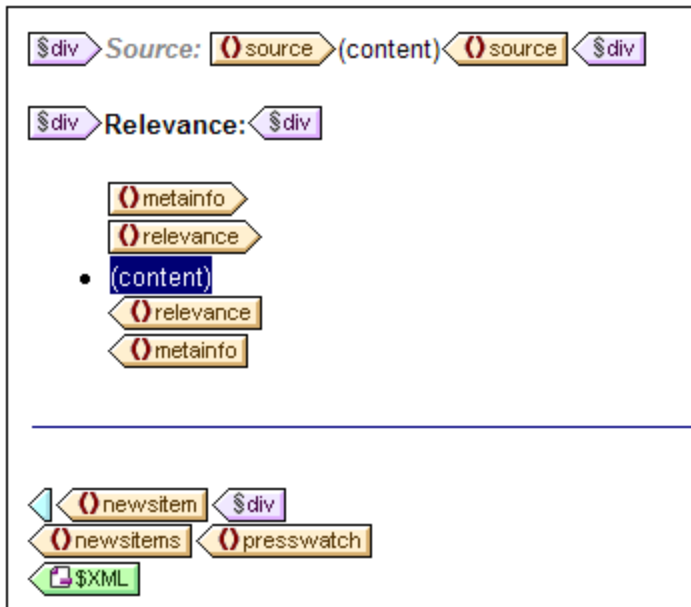


Note that there is a single branch in this condition. News items for which the condition test evaluates to true are displayed, those for which the condition test does not evaluate to true are not displayed. The condition in this case, therefore, works as a filter. Later in this section, you will add 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](#)⁴⁴ 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:

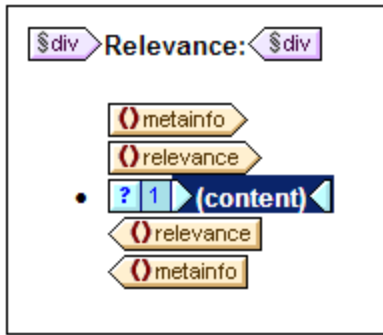


Now, in Authentic View, check the results for different selections of `relevance`; use the combo box to change the selection.

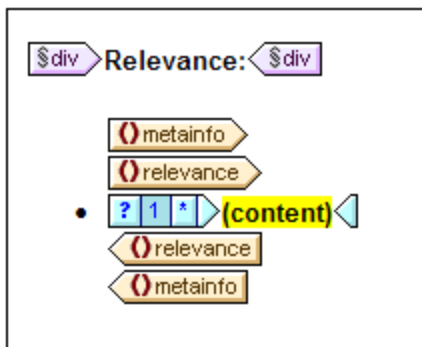
Making the selected `relevance` element bold

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](#)⁶⁸⁵ 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.



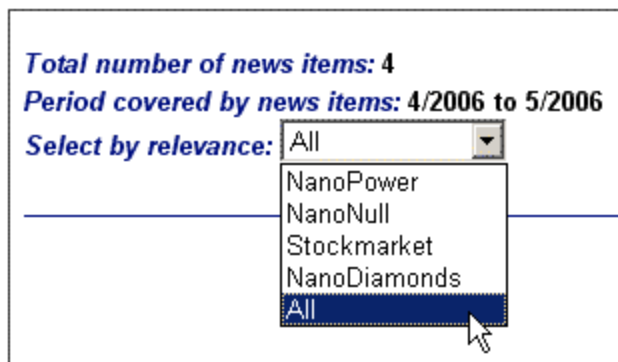
7. Notice that the contents of the *Otherwise* branch are a copy of the first branch; the `contents` placeholder is bold and has a yellow background. Remove this formatting (bold and background-color) from the `contents` placeholder.

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](#)¹⁷⁷ 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*).



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](#)⁶⁸⁵ 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.

3.7 Using Global Templates and Rest-of-Contents

[Global templates](#)²⁷ 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.

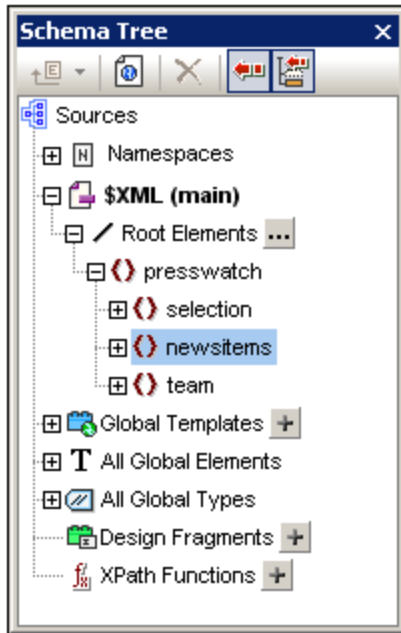
There are two mechanisms that are used to invoke a global template from the main template:

- 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](#)²⁷ `member`.

Inserting the rest-of-contents instruction

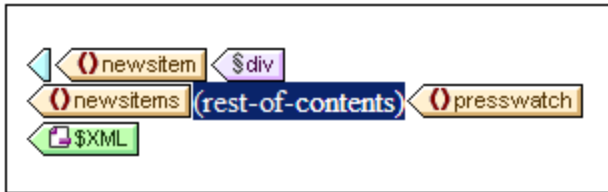
The broad structure of the schema is shown in the screenshot below.



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](#)⁷⁶⁴. 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@nanonull.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](#)⁴⁴.
2. In the context menu that pops up, select **Make / Remove Global Template**. A global template for `selection` is created (*screenshot below*).



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*).

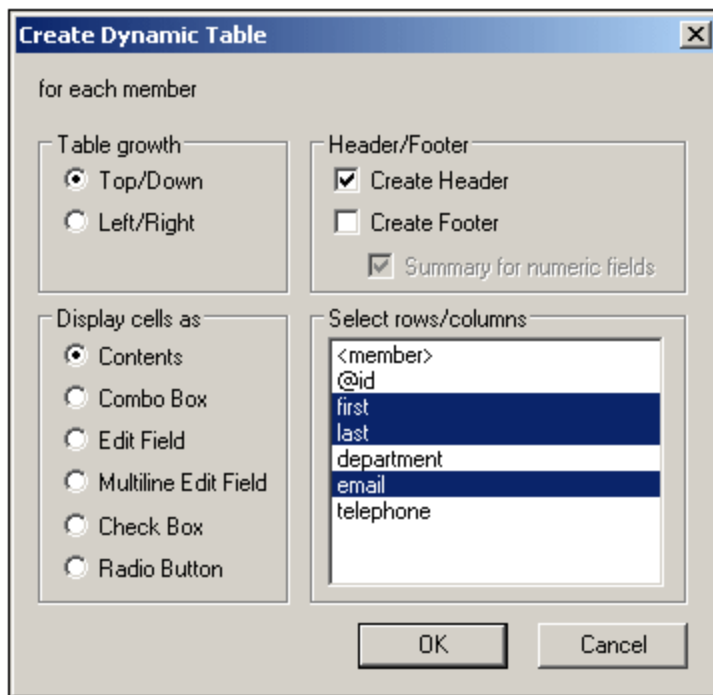
```
AndrewBentincka.bentinck@nanonull.comNadiaEdwardsn.e
```

Since the global template for `selection` is empty, the child elements of `selection` are not processed.

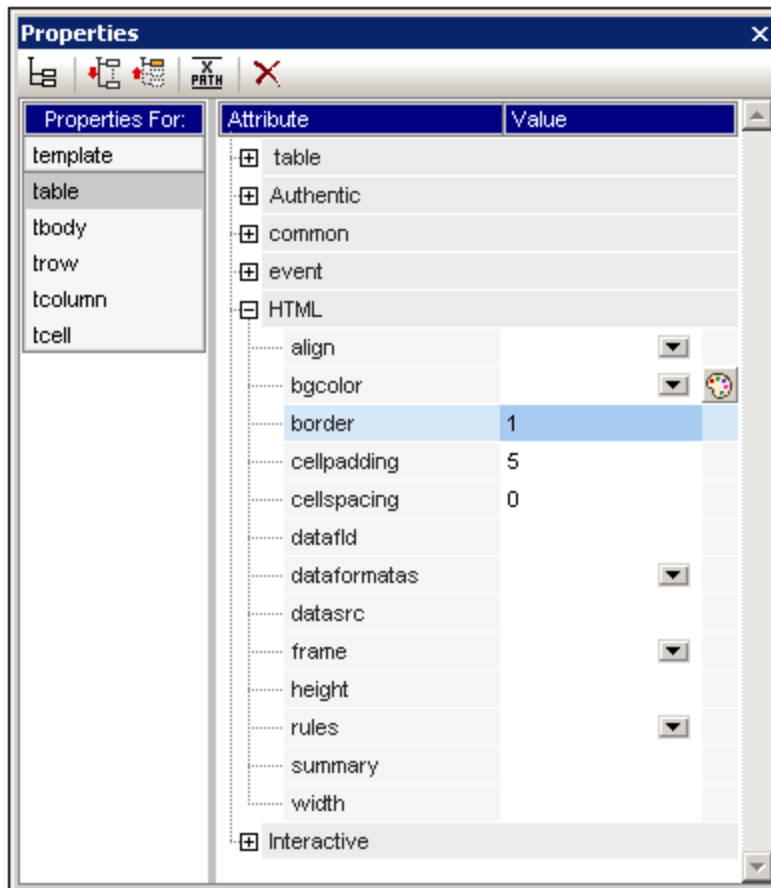
Creating a global template for `team/member`

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*).



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](#)⁵⁵, with `table` selected in the Properties For column, specify table properties as shown in the screenshot below.



- 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 `tr` selected in the Styles For column of the Styles sidebar, specifying a value for the `background-color` property (`color` group). You can also edit the headers, which are strings of static text. Also, if the `content` placeholder of the `team` element is still present in the global template, delete it.

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

First	Last	Email
Andrew	Bentinck	a.bentinck@nanonull.com
Nadia	Edwards	n.edwards@nanonull.com
John	Edwards	j.edwards@nanonull.com
Janet	Ashe	j.ashe@nanonull.com

3.8 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](#)⁶⁶ of the document ([main template](#)⁶⁸ and [global templates](#)⁹⁷).
- How to insert [dynamic](#)⁶⁸ and [static](#)⁷⁵ content in the design, using a variety of dynamic and static SPS components..
- How to use [CSS styles](#)⁶⁸, in [external stylesheets](#)⁸², in [global style rules](#)⁸², and in [local style rules](#)⁸⁴.
- How to use [Auto-Calculations](#)⁸⁶ to derive additional information from the available XML data.
- How to use [conditions](#)⁹⁰ to filter the XML data and how to obtain different outputs depending on values in the XML data.
- How to use [global templates](#)⁹⁸ and [rest-of-contents](#)⁹⁷.

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

- [SPS File: Content](#)¹²⁴
- [SPS File: Structure](#)²⁰⁰
- [SPS File: Advanced Features](#)²⁶⁹
- [SPS File: Presentation](#)³⁴⁹
- [SPS File: Additional Functionality](#)⁴¹⁵

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, Text output. The XSLT stylesheets can be generated using the [File | Save Generated Files](#)⁷³⁴ command or via the [command line](#)⁶⁶². Try generating XSLT stylesheets from `MyQuickStart.sps` and then using these stylesheets to transform `QuickStart.xml`.

4 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, Text, and RTF output.

In this way, the SPS can be used to enable XML document editing and to generate HTML, Text, and RTF 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](#)⁴⁴ 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](#)⁴². Alternatively, a new SPS can be created directly with a schema via the **File | New** command.
2. [Assign a Working XML File](#)⁴⁴ to the SPS. The [Working XML File](#)²⁸ provides the XML data processed by the SPS when generating Authentic View and output previews. The [Working XML File](#)²⁸ is assigned in the [Design Overview sidebar](#)⁴². The Working XML File enables you to preview output in StyleVision.
3. [Select the required XSLT version](#)¹⁰⁵. In order to generate Text output, the XSLT version must be XSLT 2.0 or XSLT 3.0
4. Select the [Internet Explorer Compatibility](#)¹⁰⁶ to match the installed Internet Explorer version.
5. The SPS document is designed in [Design View](#)³³ using the various design components available to the designer. The [design process](#)¹⁰⁴ consists of creating a document structure and defining [presentation properties](#)³⁴⁹. If [print output](#)³⁹² is required, then additional [print formatting properties](#)³⁹² 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](#)¹¹¹ are required, these are [generated](#)¹¹¹.
8. If required, assign a Template XML File. The [Template XML File](#)¹⁰³ 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](#)¹⁰⁸ among multiple Authentic View users.

4.1 SPS and Sources

Creating a new SPS file

To create a new SPS document, select an option from under the [File | New \(Ctrl+N\)](#)⁷¹⁴ command or click the **New Design** icon  in the [Standard toolbar](#)⁷¹². 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](#)¹⁰³
- [Working XML File](#)¹⁰³
- [Template XML File](#)¹⁰³

These source file assignments are made in the [Design Overview sidebar](#)⁴². How to make the assignments is described in the section, [Design Overview](#)⁴². The significant points about each type of source file are given below.

Schema sources

A schema source file must be assigned to an SPS so that a structure for the design document can be created. Schema sources are assigned in the [Design Overview sidebar](#)⁴². 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, or a user-defined schema. For each schema, one optional [Working XML File](#)¹⁰³ and one optional [Template XML File](#)¹⁰³ 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

An SPS can, optionally, have a [Working XML File](#)²⁸ associated with it. The function of the [Working XML File](#)²⁸ 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](#)²⁸ is assigned in the [Design Overview sidebar](#)⁴².

Template XML File

An SPS can have a [Template XML File](#)²⁸ optionally associated with it. The function of the [Template XML File](#)²⁸ is to provide the starting data of the new XML document that is created each time that SPS is opened in the [Authentic View](#)²⁴ of a product other than StyleVision. The [Template XML File](#)²⁸ must be valid according to the schema with which it is associated. It is assigned in the [Design Overview sidebar](#)⁴².

4.2 Creating the Design

In the SPS design, you specify:

1. [What content](#)¹⁰⁴ (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](#)¹⁰⁴; and
3. [What presentation \(formatting\) properties](#)¹⁰⁴ 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 to which the SPS is applied. Content from the [XML document](#)²⁸ is included in the SPS by dragging the required XML data node from the relevant schema tree in the [Schema Tree sidebar](#)⁴⁴ 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](#)²⁴ 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](#)⁴⁴ 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](#)²⁷⁰.
5. For the HTML output, [JavaScript functions](#)⁴⁶⁷ can be used to generate content.




Structure of output

In the SPS design, the [structure of the output](#)²⁴⁴ can be controlled by using either: (i) a procedural approach, in which the output structure is specified in an [entry-level template](#)²⁴⁴ (StyleVision's [main template](#)²⁴⁴) and can be independent of the structure of the XML document; (ii) a declarative approach, in which [template rules are declared for various nodes](#)²⁴⁴ (StyleVision's [global templates](#)²⁴⁴), 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](#)²⁴⁴ and [global templates](#)²⁴⁴ to obtain the desired structure for the output document. The use of [Modular SPSs](#)²³⁰ and [Design Fragments](#)²⁵⁵ 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](#)³⁵⁰. Specifying presentation properties is described in detail in the section, [Presentation Procedures](#)³⁴⁹.

4.3 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:  or  or . 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.

Note: In order to generate Text output, the XSLT version must be XSLT 2.0 or XSLT 3.0

XSLT transformations

XSLT transformations in StyleVision are used: (i) to generate [output views](#)³⁶ in the interface; and (ii) to [generate and save output files](#)¹¹¹ (HTML, Text, and RTF) from [within the interface](#)⁷³⁴ 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](#)²⁷⁰ and [Conditional Templates](#)²⁸⁰—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.

4.4 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](#)¹⁷⁰.
- 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](#)¹⁴⁰.

Design View and IE versions

You can set up Design View for a specific IE version by specifying, in the [Properties](#)⁷³⁸ 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](#)¹²⁷ or as [user-defined elements](#)¹⁴⁰. 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](#)⁷³⁸ for details.

Compatibility of older SPS designs with IE9

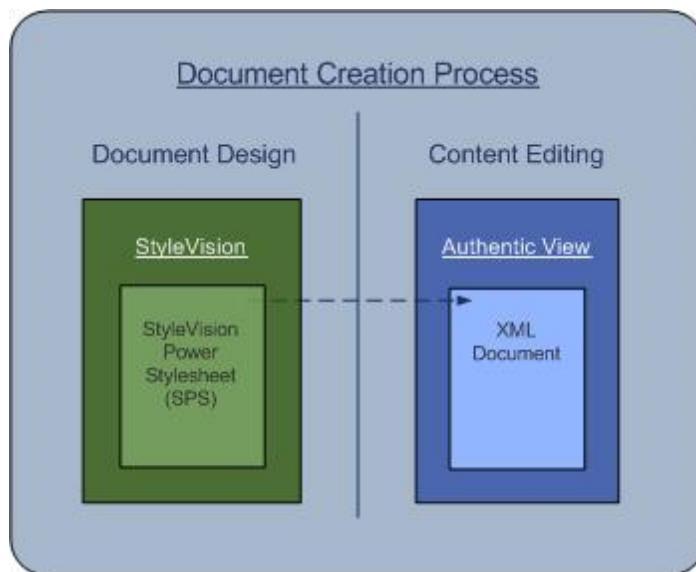
If you open an SPS design that has been created for an older IE version, and if the newer IE9 version or higher is installed on the StyleVision machine, then StyleVision will detect the newer version and ask in a dialog whether you wish to change the compatibility to IE9-compatibility. Changing to the 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](#)⁷³⁸ dialog, the IE compatibility back to that of the previously selected IE version.

4.5 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](#)²⁴, 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](#)²⁸) 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.

Required		Optional	
<h3>NCA XML Invoice</h3> <p>add documentID</p> <p>Invoice Type: <i>(Proforma, Final, Debit Note, Credit Note)</i> <input type="text"/></p> <p>Status: <i>(Draft, Final, Ammended)</i> <input type="text"/></p>			<p>This is a template for the NCA Invoice used to define a detailed list of goods shipped or services rendered, with an account of costs. Information created could consist of parties associated with the transaction, type of goods shipped or method of shipment.</p>
<h4>General Information</h4> <p>References and other general information pertaining to the contract and this document.</p>			
<p>Date of Issue: <i>(yyy-mm-dd)</i> <input type="text"/></p> <p>Document Creator Identifier: <input type="text"/></p> <p>Docuement Number: <input type="text"/> Invoice</p> <p>Number: <input type="text"/></p>	<p>add contractExtension add documentVersion add contractType Contract Id</p> <p>TransactionNumber add buyerContractIdentifier add sellerContractIdentifier add brokerContract</p>		

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

4.6 Synchronizing StyleVision and Authentic

Each new release of StyleVision contains features that add to the power and capability of [Authentic View](#)²⁴. 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](#)²⁴ 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](#)²⁴ 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](#)²⁴ 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.

4.7 Generated Files

In StyleVision, XSLT stylesheets and output files can be generated using the [File | Save Generated Files](#)⁷³⁴ 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](#)⁶⁶⁵.

The following files can be generated from StyleVision:

- XSLT stylesheets based on the SPS design. Separate XSLT stylesheets are generated for HTML, Text, and RTF output.
- Output files, generated by processing the [Working XML File](#)²⁸ 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](#)³⁶.

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.

Altova website: [XML reporting](#)

Output documents

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.

Altova website: [XML to HTML](#)

- **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](#)⁵⁵ and the [Design Tree sidebar](#)⁴⁸. (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.

- **Text output and stylesheets:** Since the plain text format does not use formatting, style properties in the design are not passed to the text 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](#)⁷³⁴ 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.

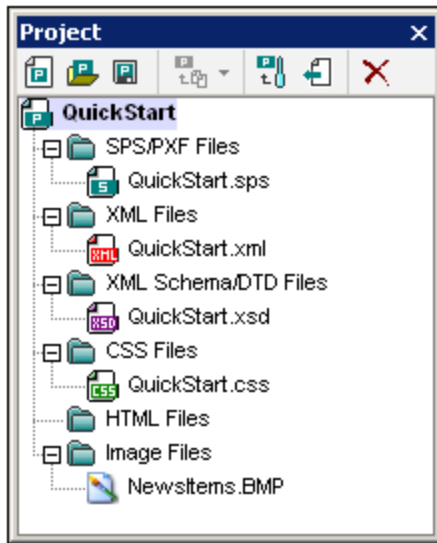
Text output

Since Text output generates plain text, note the following points:

- Style and page layout properties that are defined in the design will not be applied to the Text output.
- In order to generate Text output, the XSLT version selected for the SPS must be XSLT 2.0 or XSLT 3.0.

4.8 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.



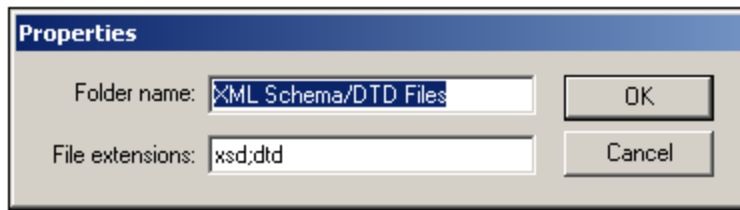
Creating and saving a project

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**.

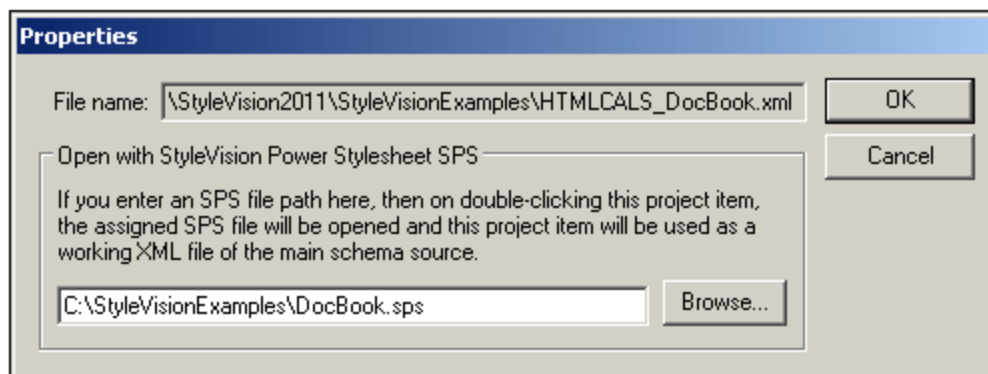


In the Properties dialog, you can edit the folder name and the file type extensions for that folder (each file type extension must be separated by a semi-colon). Project folder names can also be edited by selecting the folder in the Project sidebar, pressing **F2**, and editing the name. When a folder 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](#)⁴¹⁶ 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](#)⁴²⁷.

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.)
- *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](#)⁸³⁹ 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.)
- *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](#)⁸³⁹ 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

- When you hover over an image file that has been placed in a project folder, a preview of the image is displayed (.png, .jpeg, .gif, .bmp, .tiff, and .ico formats). Double-click the image to open it in the system's default image viewer/editor application.
- The context menu command *Open* opens the image file in the default image viewer/editing application.
- The context menu command *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.)

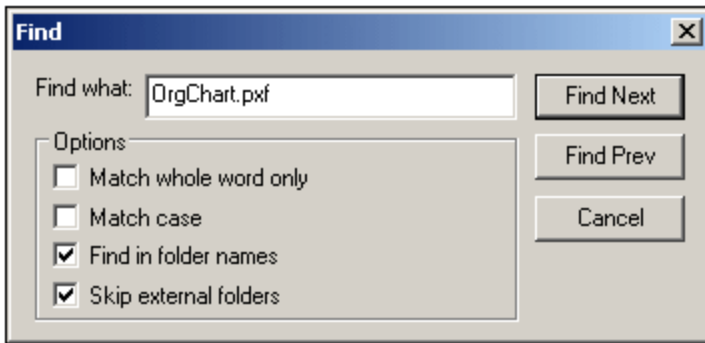
All file types

- *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](#)⁷⁵⁶ 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.

4.9 Catalogs in StyleVision

The XML catalog mechanism enables files to be retrieved from local folders, thus increasing the overall processing speed, as well as improving the portability of documents—since only the catalog file URIs then need to be changed. See the section [How Catalogs Work](#)¹¹⁸ for details.

Altova's XML products use a catalog mechanism to quickly access and load commonly used files, such as DTDs and XML Schemas. This catalog mechanism can be customized and extended by the user, and it is described in the sections [Catalog Structure in StyleVision](#)¹¹⁹ and [Customizing your Catalogs](#)¹²⁰. The section [Variables for Windows System Locations](#)¹²² list Windows variables for common system locations. These variables can be used in catalog files to locate commonly used folders.

This section is organized into the following sub-sections:

- [How Catalogs Work](#)¹¹⁸
- [Catalog Structure in StyleVision](#)¹¹⁹
- [Customizing your Catalogs](#)¹²⁰
- [Variables for Windows System Locations](#)¹²²

For more information on catalogs, see the [XML Catalogs specification](#).

4.9.1 How Catalogs Work

Catalogs can be used to redirect both DTDs and XML Schemas. While the concept behind the mechanisms of both cases is the same, the details are different and are explained below.

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, its 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 map the `PUBLIC` identifier 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, then the corresponding URL in the catalog file will be looked up and the schema will be read from this location. So, for example, if the following SVG file is opened in StyleVision:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg width="20" height="20" xml:space="preserve">
  <g style="fill:red; stroke:#000000">
    <rect x="0" y="0" width="15" height="15"/>
    <rect x="5" y="5" width="15" height="15"/>
  </g>
</svg>
```

The catalog is searched for the `PUBLIC` identifier of this SVG file. Let's say the catalog file contains the following entry:

```
<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. As a result, the lookup for the SVG DTD is redirected to the URL `schemas/svg/svg11.dtd` (which is relative to the catalog file). This is a local file that will be used as the DTD for the SVG file. If there is no mapping for the `Public` ID in the catalog, then the URL in the XML document will be used (in the SVG file example above, this is the Internet URL: `http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd`).

XML Schemas

In StyleVision, you can also use catalogs with **XML Schemas**. In the XML instance file, the reference to the schema will occur in the `xsi:schemaLocation` attribute of the XML document's top-level element. For example,

```
xsi:schemaLocation="http://www.xmlspy.com/schemas/orgchart OrgChart.xsd"
```

The value of the `xsi:schemaLocation` attribute has two parts: a namespace part (green above) and a URI part (highlighted). The namespace part is used in the catalog to map to the alternative resource. For example, the following catalog entry redirects the schema reference above to a schema at an alternative location.

```
<uri name="http://www.xmlspy.com/schemas/orgchart" uri="C:\MySchemas\OrgChart.xsd"/>
```

Normally, the URI part of the `xsi:schemaLocation` attribute's value 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 must 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 to be valid.

4.9.2 Catalog Structure in StyleVision

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 of which is referenced in a `nextCatalog` element. These catalog files are looked up and the URIs in them are resolved according to their mappings.

Listing of RootCatalog.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"/>
  <!-- Include all catalogs under common schemas folder on the first directory level -->
  <nextCatalog spy:recurseFrom="%CommonSchemasFolder%" catalog="catalog.xml"
spy:depth="1"/>
```

```

<nextCatalog spy:recurseFrom="%ApplicationWritableDataFolder%/pkgs/.cache"
catalog="remapping.xml" spy:depth="0"/>
<nextCatalog catalog="CoreCatalog.xml"/>
</catalog>

```

The listing above references a custom catalog (named `CustomCatalog.xml`) and a set of catalogs that locate commonly used schemas (such as W3C XML Schemas and the SVG schema).

- `CustomCatalog.xml` is located in your Personal Folder (located via the variable `%PersonalFolder%`). It is a skeleton file in which you can create your own mappings. You can add mappings to `CustomCatalog.xml` for any schema you require that is not addressed by the catalog files in the Common Schemas Folder. Do this by using the supported elements of the OASIS catalog mechanism (see *next section*).
- The Common Schemas Folder (located via the variable `%CommonSchemasFolder%`) contains a set of commonly used schemas. Inside each of these schema folders is a `catalog.xml` file that maps public and/or system identifiers to URIs that point to locally saved copies of the respective schemas.
- `CoreCatalog.xml` is located in the StyleVision application folder, and is used to locate schemas and stylesheets used by StyleVision-specific processes, such as StyleVision Power Stylesheets which are stylesheets used to generate Altova's Authentic View of XML documents.

Location variables

The variables that are used in `RootCatalog.xml` (listing above) have the following values:

<code>%PersonalFolder%</code>	Personal folder of the current user, for example <code>C:\Users\<name>\Documents</name></code>
<code>%CommonSchemasFolder%</code>	<code>C:\ProgramData\Altova\Common2024\Schemas</code>
<code>%ApplicationWritableDataFolder%</code>	<code>C:\ProgramData\Altova</code>

Location of catalog files and schemas

Note the locations of the various catalog files.

- `RootCatalog.xml` and `CoreCatalog.xml` are in the StyleVision application folder.
- `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 Common Schemas Folder.

4.9.3 Customizing Your Catalogs

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.

- `<public publicId="PublicID of Resource" uri="URL of local file"/>`
- `<system systemId="SystemID of Resource" uri="URL of local file"/>`
- `<uri name="filename" uri="URL of file identified by filename"/>`

- `<rewriteURI uriStartString="StartString of URI to rewrite" rewritePrefix="String to replace StartString"/>`
- `<rewriteSystem systemIdStartString="StartString of SystemID" rewritePrefix="Replacement string to locate resource locally"/>`

Note the following points:

- 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.
- A URI can be mapped to another URI using the `uri` element.
- The `rewriteURI` and `rewriteSystem` elements enable the rewriting of the starting part of a URI or system identifier, respectively. This allows the start of a filepath to be replaced and consequently enables the targeting of another directory. For more information on these elements, see the [XML Catalogs specification](#).

From release 2014 onwards, StyleVision adheres closely to the [XML Catalogs specification \(OASIS Standard V1.1.7 October 2005\)](#) specification. This specification strictly separates external-identifier look-ups (those with a Public ID or System ID) from URI look-ups (URIs that are not Public IDs or System IDs). Namespace URIs must therefore be considered simply URIs—not Public IDs or System IDs—and must be used as URI look-ups rather than external-identifier look-ups. In StyleVision versions prior to version 2014, schema namespace URIs were translated through `<public>` mappings. From version 2014 onwards, `<uri>` mappings have to be used.

Prior to v2014: `<public publicID="http://www.MyMapping.com/ref" uri="file:///C:/MyDocs/Catalog/test.xsd"/>`
V-2014 onwards: `<uri name="http://www.MyMapping.com/ref" uri="file:///C:/MyDocs/Catalog/test.xsd"/>`

How StyleVision finds a referenced schema

A schema is referenced in an XML document via the `xsi:schemaLocation` attribute (*shown below*). The value of the `xsi:schemaLocation` attribute has two parts: a namespace part (green) and a URI part (highlighted).

```
xsi:schemaLocation="http://www.xmlspy.com/schemas/orgchart OrgChart.xsd"
```

Given below are the steps, followed sequentially by StyleVision, to find a referenced schema. The schema is loaded at the first successful step.

1. Look up the catalog for the URI part of the `xsi:schemaLocation` value. If a mapping is found, including in `rewriteURI` mappings, use the resulting URI for schema loading.
2. Look up the catalog for the namespace part of the `xsi:schemaLocation` value. If a mapping is found, including in `rewriteURI` mappings, use the resulting URI for schema loading.
3. Use the URI part of the `xsi:schemaLocation` value for schema loading.

XML Schema specifications

XML Schema specification information is built into StyleVision and the validity of XML Schema (`.xsd`) documents is checked against this internal information. In an XML Schema document, therefore, no references should be made to any schema that defines the XML Schema specification.

The `catalog.xml` file in the `%AltovaCommonSchemasFolder%\Schemas\schema` folder contains references to DTDs that implement older XML Schema specifications. You should not validate your XML Schema documents against these schemas. The referenced files are included solely to provide StyleVision with entry helper info for

editing purposes should you wish to create documents according to these older recommendations.

4.9.4 Variables for Windows System Locations

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:

<code>%PersonalFolder%</code>	Full path to the Personal folder of the current user, for example <code>C:\Users\<<name>\Documents</code>
<code>%CommonSchemasFolder%</code>	<code>C:\ProgramData\Altova\Common2024\Schemas</code>
<code>%ApplicationWritableDataFolder%</code>	<code>C:\ProgramData\Altova</code>
<code>%AltovaCommonFolder%</code>	<code>C:\Program Files\Altova\Common2024</code>
<code>%DesktopFolder%</code>	Full path to the Desktop folder of the current user.
<code>%ProgramMenuFolder%</code>	Full path to the Program Menu folder of the current user.
<code>%StartMenuFolder%</code>	Full path to Start Menu folder of the current user.
<code>%StartupFolder%</code>	Full path to Start Up folder of the current user.
<code>%TemplateFolder%</code>	Full path to the Template folder of the current user.
<code>%AdminToolsFolder%</code>	Full path to the file system directory that stores administrative tools of the current user.
<code>%AppDataFolder%</code>	Full path to the Application Data folder of the current user.
<code>%CommonAppDataFolder%</code>	Full path to the file directory containing application data of all users.
<code>%FavoritesFolder%</code>	Full path of the Favorites folder of the current user.
<code>%PersonalFolder%</code>	Full path to the Personal folder of the current user.
<code>%SendToFolder%</code>	Full path to the SendTo folder of the current user.
<code>%FontsFolder%</code>	Full path to the System Fonts folder.
<code>%ProgramFilesFolder%</code>	Full path to the Program Files folder of the current user.
<code>%CommonFilesFolder%</code>	Full path to the Common Files folder of the current user.
<code>%WindowsFolder%</code>	Full path to the Windows folder of the current user.
<code>%SystemFolder%</code>	Full path to the System folder of the current user.
<code>%LocalAppDataFolder%</code>	Full path to the file system directory that serves as the data repository for local (nonroaming) applications.
<code>%MyPicturesFolder%</code>	Full path to the MyPictures folder.

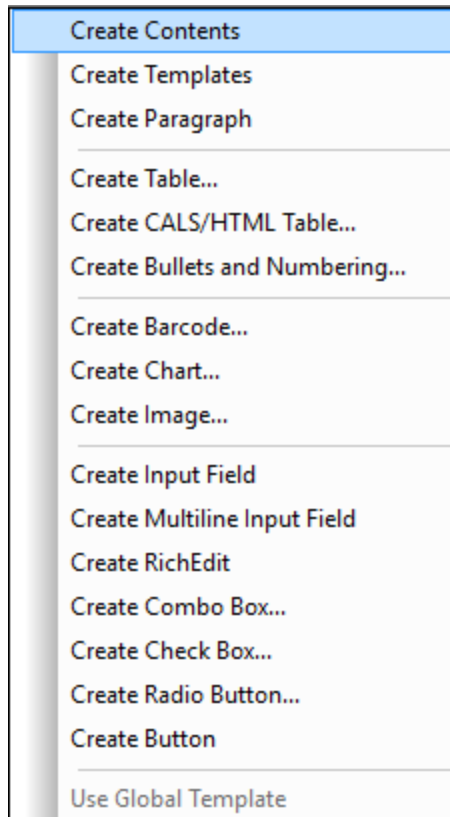
5 SPS Content

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](#)²⁴⁴, [global](#)²⁴⁴, or [named](#)²⁵⁵.

- [Inserting XML Content as Text](#)¹²⁵. 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).
- [Inserting MS Word Content](#)¹³²
- [User-Defined Templates](#)¹³⁷
- [User-Defined Elements, XML Text Blocks](#)¹⁴⁰
- [Working with Tables](#)¹⁴³. 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](#)¹⁶³. 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](#)¹⁶⁸: 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).
- [Using Data-Entry Devices \(or Form Controls\)](#)¹⁷⁴. 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 of input constraints. Individual nodes in the XML document can be created as data-entry devices.
- [Links](#)¹⁸²
- [Barcodes](#)¹⁸³
- [Layout Modules](#)¹⁸⁷
- [The Change-To Feature](#)¹⁹⁷. 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.

5.1 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*).



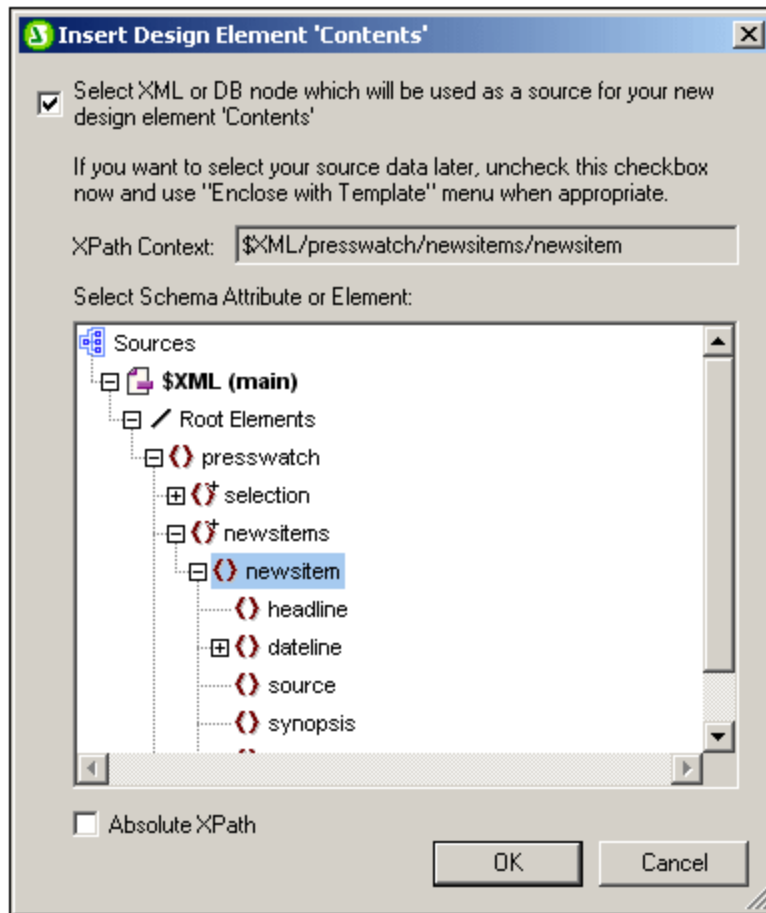
Types of schema nodes

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](#)⁷⁰⁸ 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

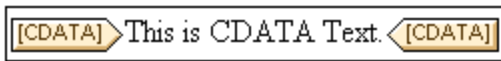
To output the text contents of the node, the node should be created as contents. When a node is created as contents, the node will look something like this in the design document:



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](#)⁸⁰⁶). 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

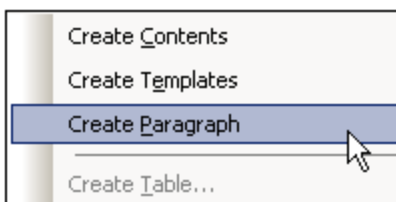
In the sub-sections of this section, we describe other aspects of inserting XML content as text:

- How the text content of a node can be [marked up with a predefined format directly](#)¹²⁷ when the node is inserted.
- How the structure of the source schema determines the [effect of Authentic View usage](#)¹²⁸.
- How descendant nodes not explicitly included within a node can be included for processing. See [Rest-of-Contents](#)¹³⁰.

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.

5.1.1 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](#)⁷⁰⁴ (*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](#). ²⁵² **Enclose With | Special Paragraph** ²⁵².

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.

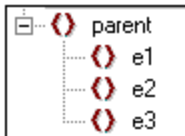
5.1.2 Adding Elements in Authentic View

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?

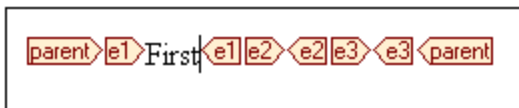
The most important point to bear in mind is that Authentic View follows the structure specified in the underlying schema. In order to ensure that Authentic View implements the schema structure correctly there are a few design rules you should keep in mind. These are explained below.

Unambiguous content model

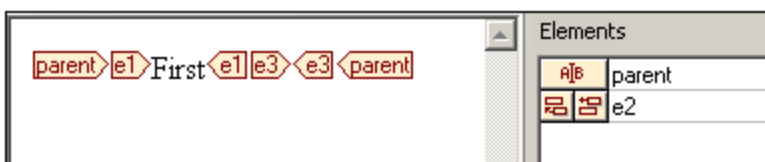
A content model is considered unambiguous when it consists of a single sequence (with `maxOccurs=1`) of child elements (with no choices, groups, substitutions, etc). In such cases, when the element is added, the sequence of child elements is unambiguously known, and they are automatically added. In the 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.

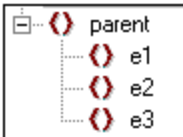
Ambiguous content model

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.

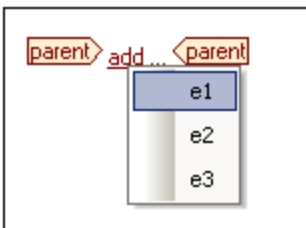
We consider a content model of an element `parent`, which consists of a single sequence of mandatory child elements. This content model is similar to the unambiguous content model discussed above, with one difference: the single sequence is optional, which makes the content model ambiguous—because the 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.



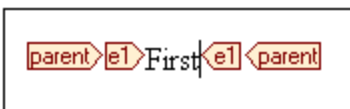
The Authentic View of the `parent` element will look like this (since the sequence is optional):



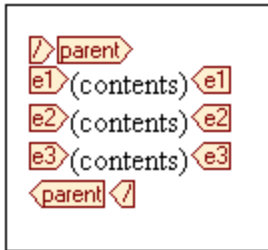
Clicking `add...` pops up a menu of the three child elements:



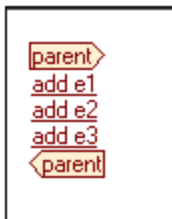
If you select one of these elements, it will be inserted (*screenshot below*), but since Authentic View cannot disambiguate the sequence it does not insert any of the remaining two elements, nor does it offer you the opportunity of inserting them:



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:



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.



5.1.3 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](#)²⁴⁴ 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.

5.2 Inserting MS Word Content

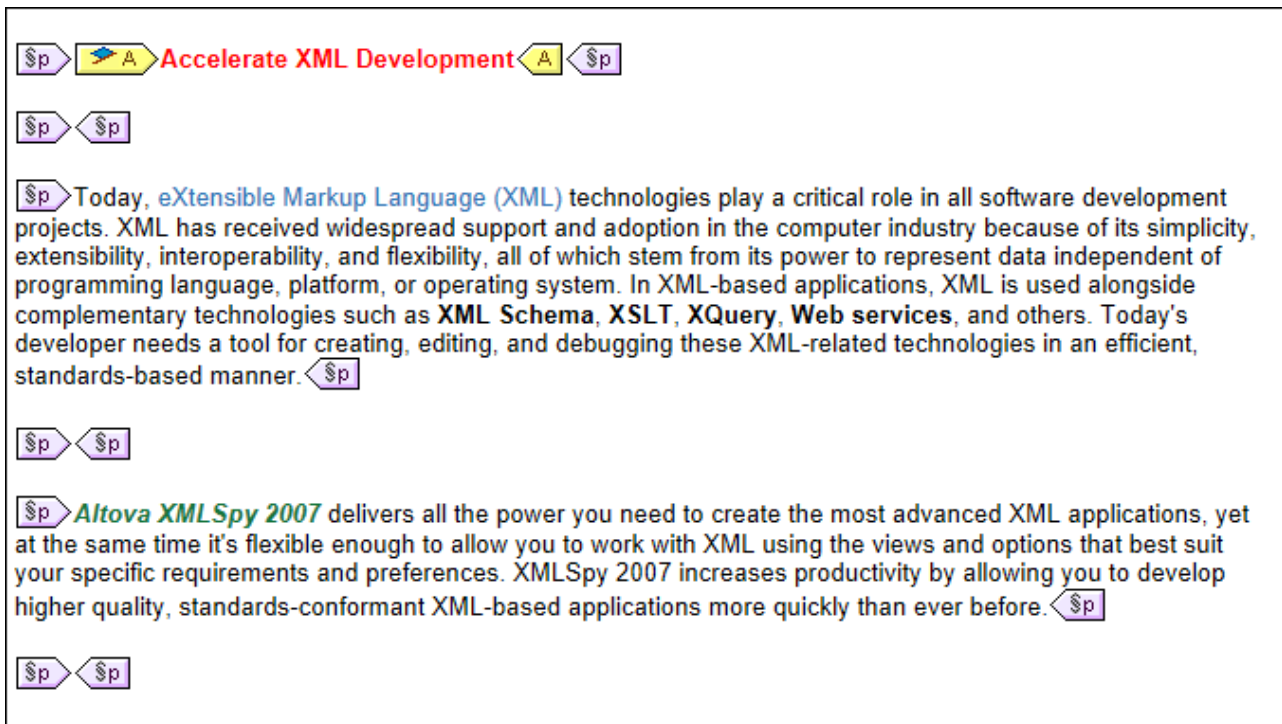
If Microsoft Word 2007+ is installed on your machine, then content can be pasted from Word documents into the design as **static content**. The Word content will be inserted within suitably corresponding design components, and text formatting properties will be carried over from the Word content. For example, text content that is in a Word paragraph block will be inserted within a [Paragraph component](#)¹²⁷, and the formatting of the text will be preserved (see *screenshots below*).

Accelerate XML Development

Today, [eXtensible Markup Language \(XML\)](#) technologies play a critical role in all software development projects. XML has received widespread support and adoption in the computer industry because of its simplicity, extensibility, interoperability, and flexibility, all of which stem from its power to represent data independent of programming language, platform, or operating system. In XML-based applications, XML is used alongside complementary technologies such as **XML Schema**, **XSLT**, **XQuery**, **Web services**, and others. Today's developer needs a tool for creating, editing, and debugging these XML-related technologies in an efficient, standards-based manner.

Altova XMLSpy 2007 delivers all the power you need to create the most advanced XML applications, yet at the same time it's flexible enough to allow you to work with XML using the views and options that best suit your specific requirements and preferences. XMLSpy 2007 increases productivity by allowing you to develop higher quality, standards-conformant XML-based applications more quickly than ever before.

Word content.



Word content pasted into a design. A suitable paragraph format has been applied and text formatting has been preserved.

Note: In addition to Word content, *any content that can be pasted into a Word document* can also be pasted into a StyleVision design. This includes **MS Excel tables** and **HTML page content**.

Note: To create an SPS that contains static content from an entire Word document, create a new SPS with the [File | New | New from Word 2007+](#)⁷¹⁴ command.

Supported Word features

The following Word structures and formats are supported when Word content is copy-pasted into a design:

- Formatted text
 - *Different fonts, size, weights, style, text-decoration, etc.*
 - *Color*
 - *Background color*
 - *Border around text*
- Paragraphs
- Page breaks
- Horizontal line
- Hyperlinks
- Bookmarks
- Tables
 - *Rowspans, colspans*
 - *Formatted/rich content*
 - *Nested tables*
 - *Headers, footers*

- Lists, sublists
 - *Bulleted: different styles*
 - *Enumerated: different styles*
- Images

5.3 Inserting MS Excel Content

If Microsoft Excel 2007+ is installed on your machine, then content can be pasted from Excel documents into the design as **static content**. The Excel content will be inserted as static tables and other suitably corresponding design components. Formatting properties will be preserved (see *screenshots below*). Each Excel sheet is inserted as a separate static table.

	A	B
1	09.03	Euclid's Elements
2	11.09	English Phrasal Verbs in Use
3	08.86	Code Book
4	09.80	Foundations and Fundamental Concepts of Mathematics
5	18.38	Style
6	08.72	The English Language
7	18.36	History of Mathematics
8	06.09	QED
9	07.89	Fowler's Modern English Usage
10	05.30	Oxford Guide to Plain English
11	21.88	Rediscover Grammar
12	11.97	How to Solve It
13	21.54	Advanced Learner's Grammar
14	19.02	Macmillan English Grammar in Context
15	22.57	Oxford Style Manual
16	16.44	Proofreading
17	20.51	Taschenbuch Mathematischer Formeln und Moderner Verfahren
18	97.50	Oxford Companion to the Book

Excel sheet.

Sheet1	
▶	09.03 Euclid's Elements
▶	11.09 English Phrasal Verbs in Use
▶	08.86 Code Book
▶	09.80 Foundations and Fundamental Concepts of Mathematics
▶	18.38 Style
▶	08.72 The English Language
▶	18.36 History of Mathematics
▶	06.09 QED
▶	07.89 Fowler's Modern English Usage
▶	05.30 Oxford Guide to Plain English
▶	21.88 Rediscover Grammar
▶	11.97 How to Solve It
▶	21.54 Advanced Learner's Grammar
▶	19.02 Macmillan English Grammar in Context
▶	22.57 Oxford Style Manual
▶	16.44 Proofreading
▶	Taschenbuch Mathematischer Formeln und Moderner 20.51 Verfahren
▶	97.50 Oxford Companion to the Book

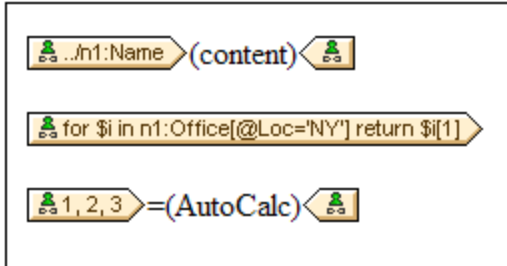
Excel content imported into a design as a static table with text formatting preserved.

Note: In addition to Excel content, *any content that can be pasted into an Excel document* can also be pasted into a StyleVision design. This includes **MS Word content** and **HTML page content**.

Note: To create an SPS that contains static content from an entire Excel document, create a new SPS with the [File | New | New from Excel 2007+](#)⁷¹⁴ command.

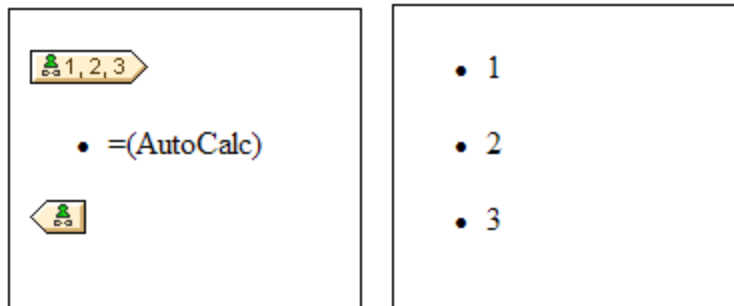
5.4 User-Defined Templates

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.

Brackets	Underlying XSLT Mechanism	Effect
No	<pre><xsl:for-each select="Org"> <xsl:for-each select="Office"> <xsl:for-each select="Dept"> ... </xsl:for-each> </xsl:for-each> </xsl:for-each></pre>	Each <code>Office</code> element has its own <code>Dept</code> population. So grouping and sorting can be done within each <code>Office</code> .
Yes	<pre><xsl:for-each select="/Org/Office/Dept"> ... </xsl:for-each></pre>	The <code>Dept</code> population extends over all <code>Office</code> elements and across <code>Org</code> .

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:

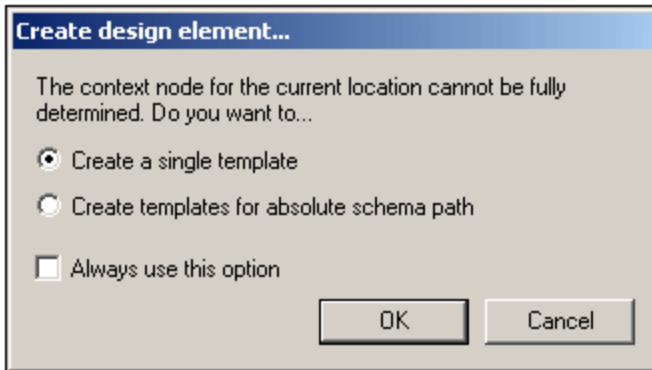
1. Click the **Insert User-Defined Template** icon in the Insert Design Elements toolbar and then click the design location where you wish to insert the template. Alternatively, right-click the design location where you wish to insert the template and, from the context menu that appears, select the **Insert User-Defined Template** command.
2. In the [Edit XPath Expression](#)⁶⁸⁵ 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](#)²⁵².

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](#)⁸³⁹.

5.5 User-Defined Elements, XML Text Blocks

[User-Defined Elements](#)¹⁴⁰ and [User-Defined XML Text Blocks](#)¹⁴¹ 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.

5.5.1 User-Defined Elements

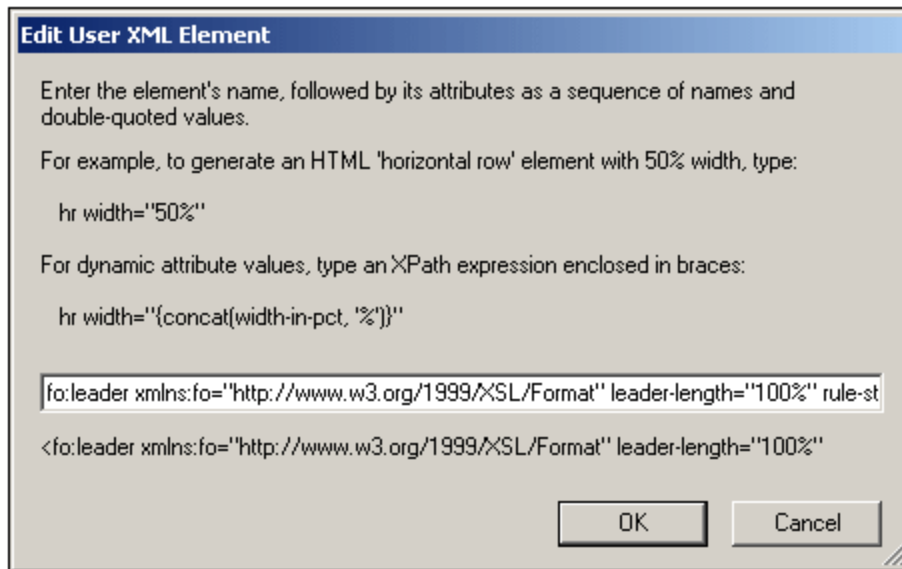
User-Defined Elements are elements that you can generate in the output without these elements needing to be in any of the schema sources of the SPS. This means that an element from any namespace (HTML or XSL-FO for example) can be inserted at any location in the design. SPS design elements can then be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

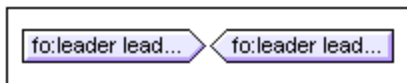
The mechanism for using User-Defined Elements is as follows:

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.



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.

5.5.2 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.

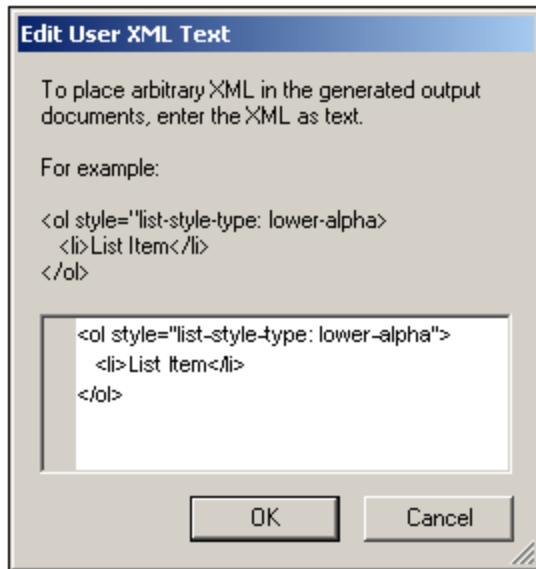
The usefulness of this feature is that it provides the stylesheet designer a mechanism with which to insert XSLT fragments and customized code in the design. For example, an ActiveX object can be inserted within an HTML `SCRIPT` element.

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.



- 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.

5.6 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.

Name	Phone
John Merrimack	6517890
Joe Concord	6402387

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:

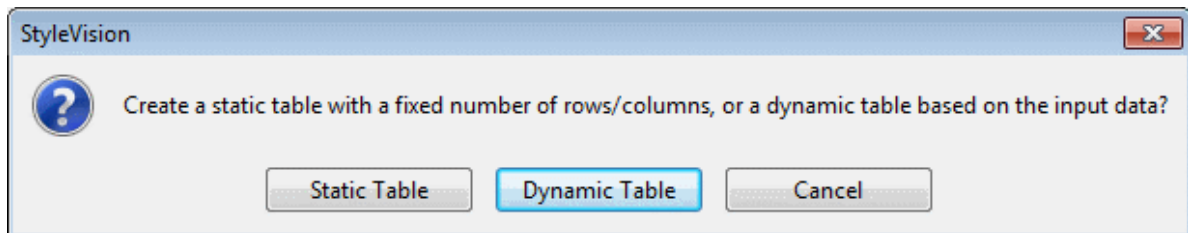
- **The structure** of an SPS table is defined in the SPS. The structure of a CALS/HTML table on the other hand is specified in the schema and must follow that of the CALS/HTML table model; the element names in the schema may, however, be different than those in the CALS or HTML table models.
- **Colspans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

5.6.1 Static Tables

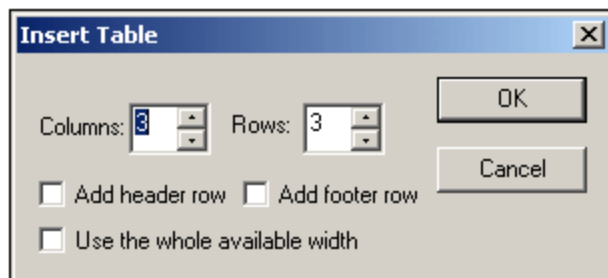
To create a static table, do the following:

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*).



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.



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

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.

Person	Telephone		Fax	
	Office	Home	Office	Home

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.

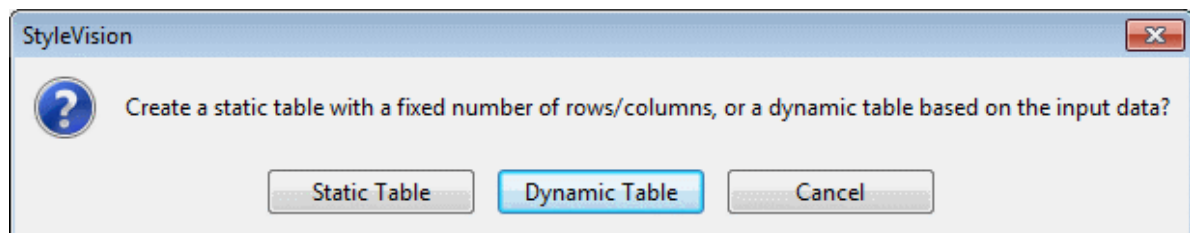
Toolbar table editing icons

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](#)⁽¹⁵⁸⁾ are not formatted in this way.

5.6.2 Dynamic Tables

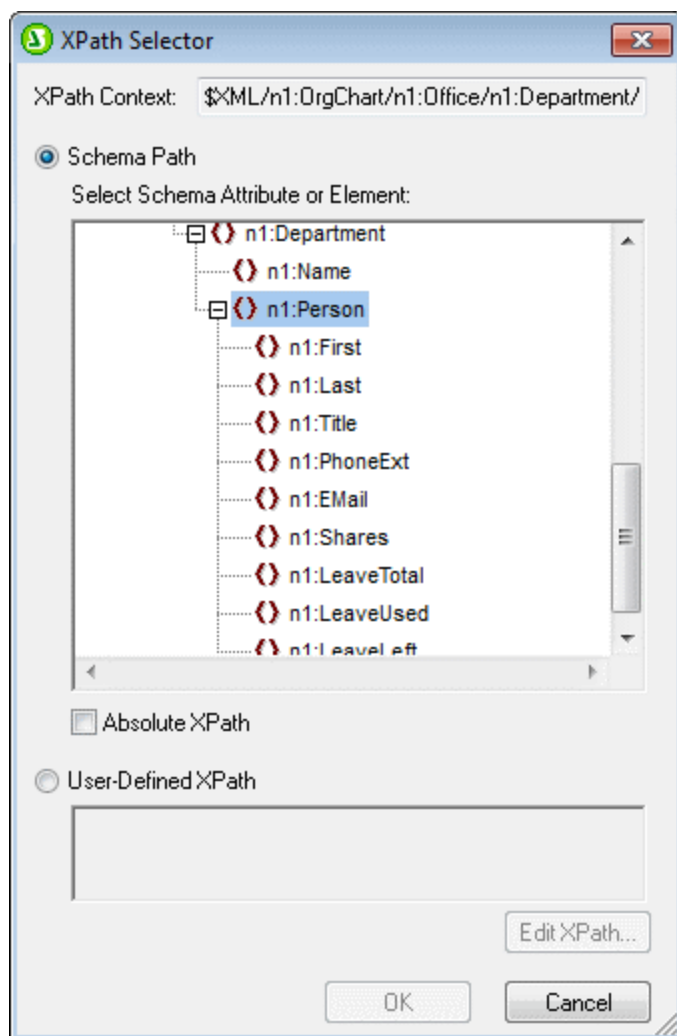
To insert a dynamic table, do the following:

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.



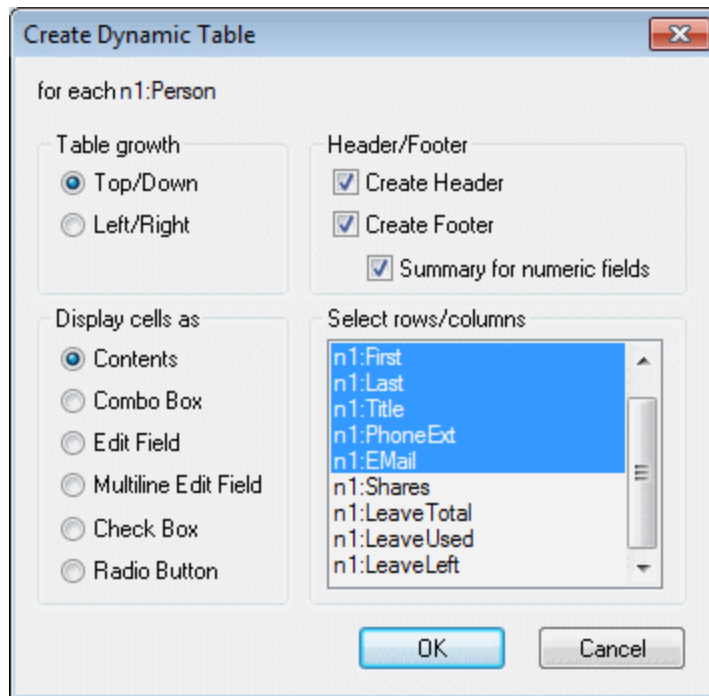
Click **Dynamic Table**.

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 XPath 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:

name	street	city	state	zip
<code><ipo:name></code>	<code><ipo:street></code>	<code><ipo:city></code>	<code><ipo:state></code>	<code><ipo:zip></code>
(contents)	(contents)	(contents)	(contents)	(contents)
<code></ipo:name></code>	<code></ipo:street></code>	<code></ipo:city></code>	<code></ipo:state></code>	<code></ipo:zip></code>

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

name	<code>ipo:name</code> (contents) <code>ipo:name</code>
street	<code>ipo:street</code> (contents) <code>ipo:street</code>
city	<code>ipo:city</code> (contents) <code>ipo:city</code>
state	<code>ipo:state</code> (contents) <code>ipo:state</code>
zip	<code>ipo:zip</code> (contents) <code>ipo:zip</code>

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. 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

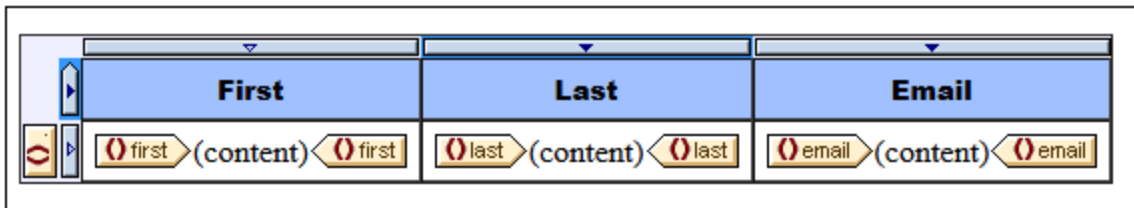
You can also create dynamic tables on elements inside global templates. The process works in the same way as for Main Template elements (described above). The important point to note is that, in a global template, a dynamic table can only be created for **descendant elements** of the global template 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.

5.6.3 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](#)⁶⁸⁵, 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.

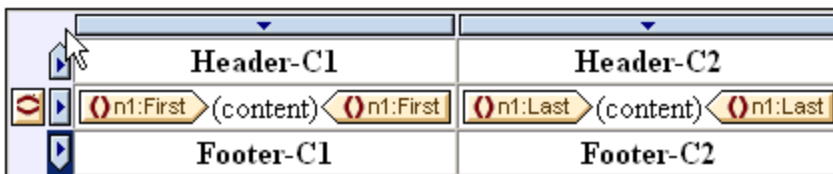
To edit an already created condition, right click the respective design component and select **Edit Conditional Processing**. In the [Edit XPath Expression dialog](#)⁶⁸⁵ that pops up, edit the XPath expression that tests the truth of the condition.

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**.

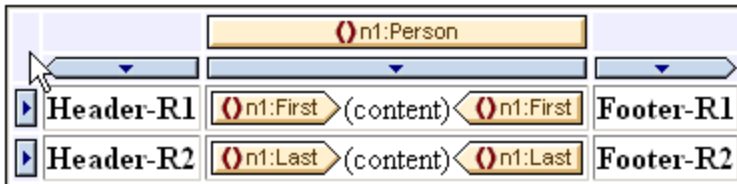
5.6.4 Tables in Design View

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*).



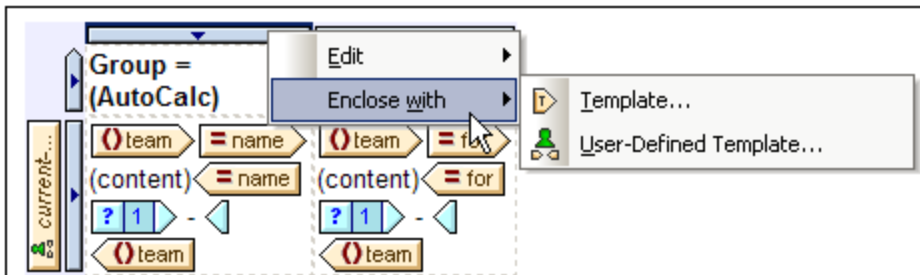
After a column or row or table has been selected, styles and/or properties can be set for the selection in the Styles and Properties Windows.

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](#)²⁴⁸. 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](#)²⁴⁸ also allows the use of interesting template-match results within the row or column (via Auto-Calculations, for example).

5.6.5 Table Formatting

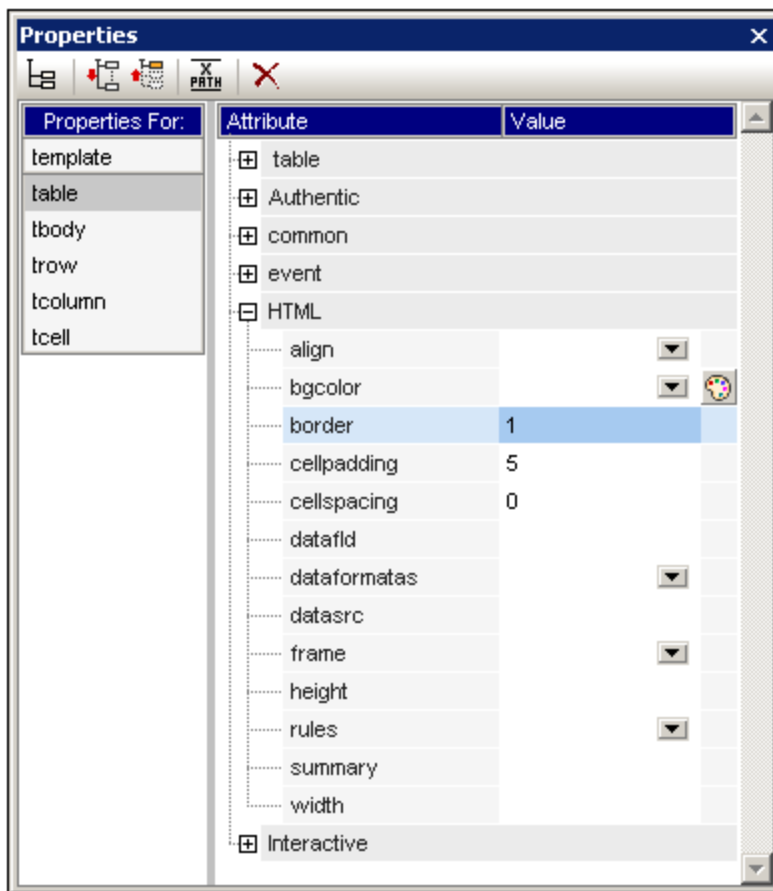
Static and dynamic tables can be formatted using:

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

Note: For Text output, tables will be generated using a [user-selected character as the cell separator](#)⁷³⁸. The default character is the semicolon. It can be changed in the XSD/XSLTtab of the [Properties dialog](#)⁷³⁸.

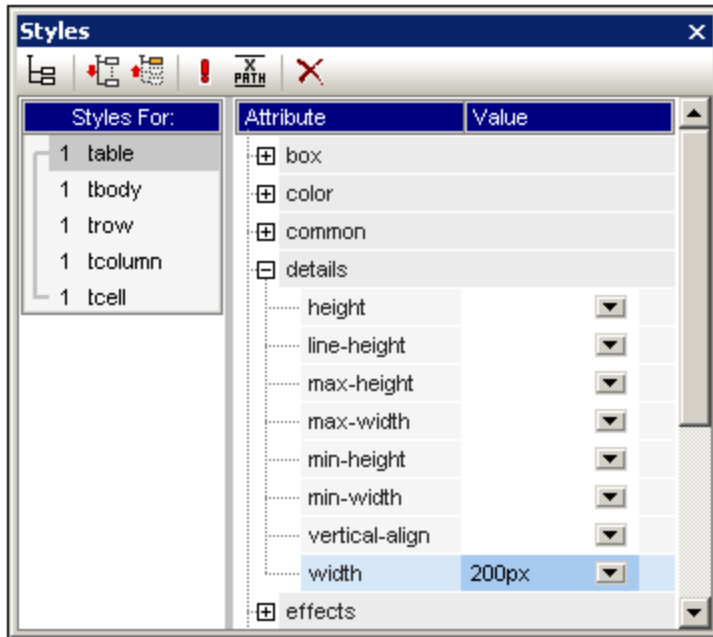
Properties sidebar

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

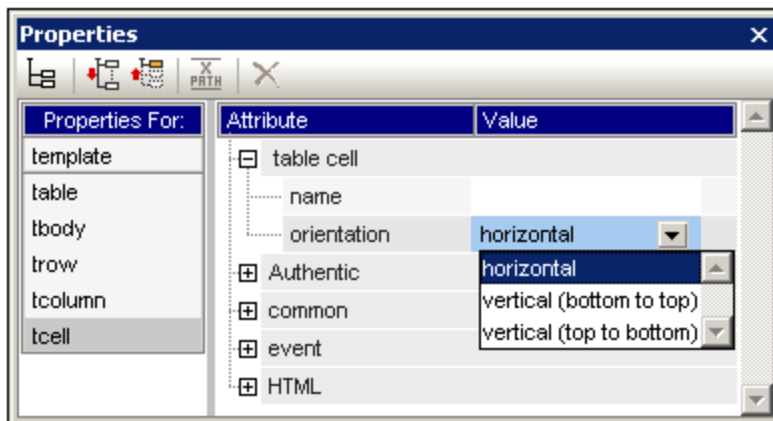
The CSS table formatting properties are available in the Styles sidebar (*screenshot below*). CSS properties are available for the table component and its sub-components (body, row, column, and cell).



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 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](#)¹⁹².

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

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](#)¹⁴⁰. 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](#)⁶⁸⁵ 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 of your PDF output, note that the XSL:FO specification uses the `hyphenate` attribute of the `fo:block` element to do this. So, to apply hyphenation, you must explicitly, in the FO document, set the `hyphenate` property of the respective `fo:block` elements to `true`.

5.6.6 Row and Column Display

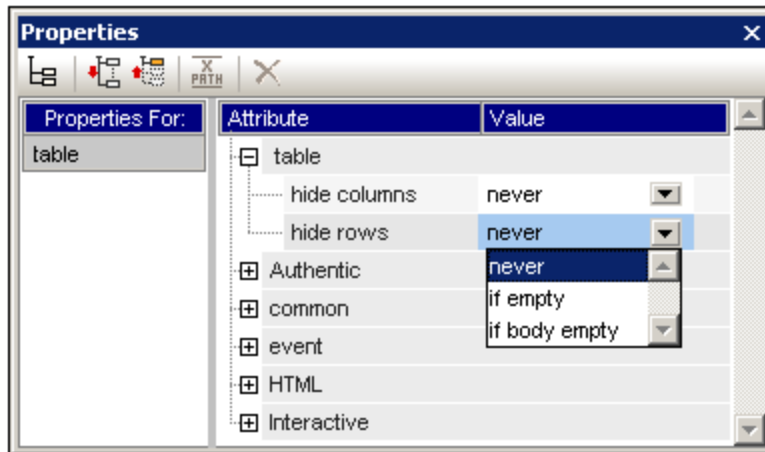
For tables, the following row and column display options are available in the **HTML output only**. These features are **not supported in Authentic View** and they require XSLT 2.0 or XSLT 3.0 to be selected as the XSLT version of the SPS.

- 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.

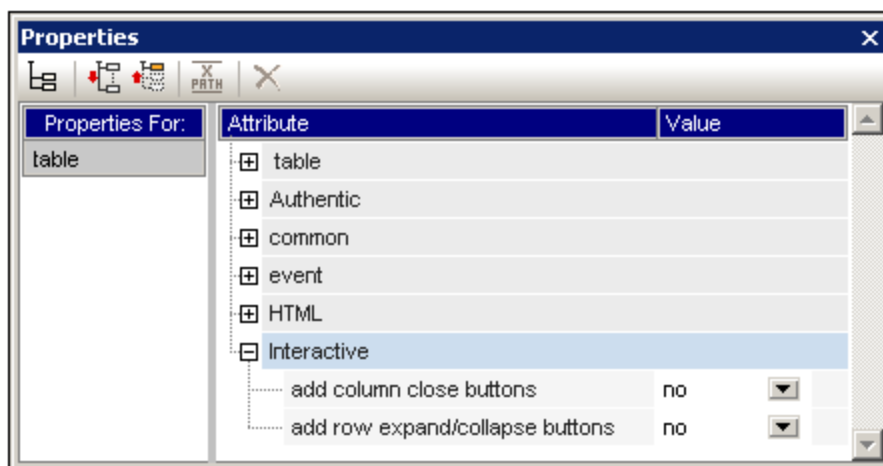
User interaction to hide columns expand/collapse rows

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.

Balance Sheet (in Millions) ⁺	2004-09-30 ^x	2004-07-01 - 2004-09-30 ^x	2003-12-31 ^x	2004-01-01 ^x
[-] Assets, Total	€ 21.49		€ 24.02	
[-] Current Assets, Total	€ 10.65		€ 12.32	
[-] Non Current Assets, Total	€ 10.85		€ 11.7	
[-] Liabilities and Equity, Total	€ 21.49		€ 24.02	
[-] Liabilities, Total	€ 8.9		€ 10.79	
Minority Interests				
[-] Equity, Total	€ 12.59		€ 13.23	
[-] Issued Capital and Reserves	€ 12.59		€ 13.23	

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).

5.6.7 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](#) ¹⁵⁹
2. [Specify formatting styles for the table](#) ¹⁶¹
3. [Insert the CALS/HTML table in the SPS design](#) ¹⁶¹

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:

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

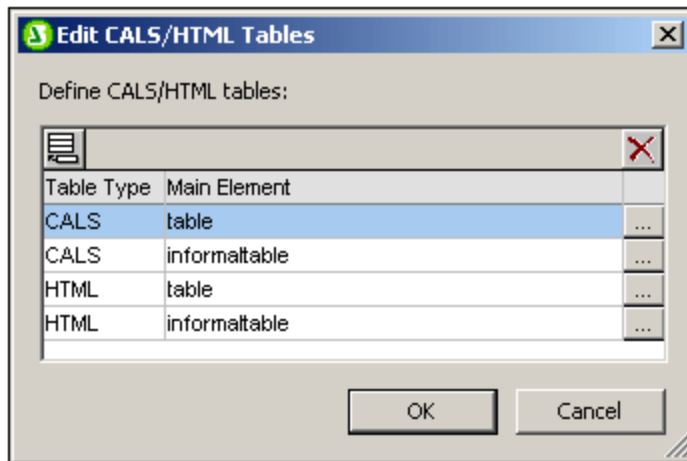
Alternatively, the XML data structure could have a structure corresponding to the HTML table model but different element names than in the HTML table model. For example:

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

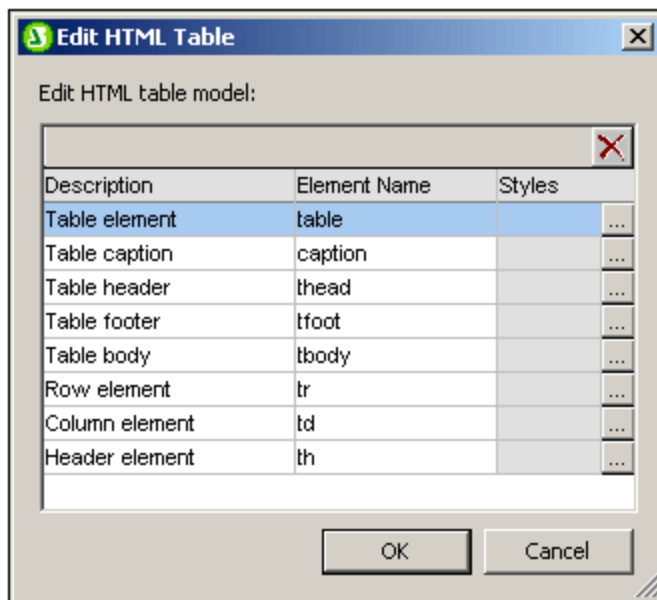
This table structure, which is defined in the XML document, can be used to directly generate a table in the various output formats. To do this you need to define this XML data structure as a CALS or HTML table. If the XML data structure is not defined as a CALS or HTML (the default), the elements in 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. (For information about the CALS table model, see the [CALS table model at OASIS](#). For an example of a `table` element having an HTML table structure, open `HTMLTable1.sps`, which is in the `Basics` folder of the `Examples` project folder (in the Project window of the GUI).) So, if you wish to enable an element in your 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.

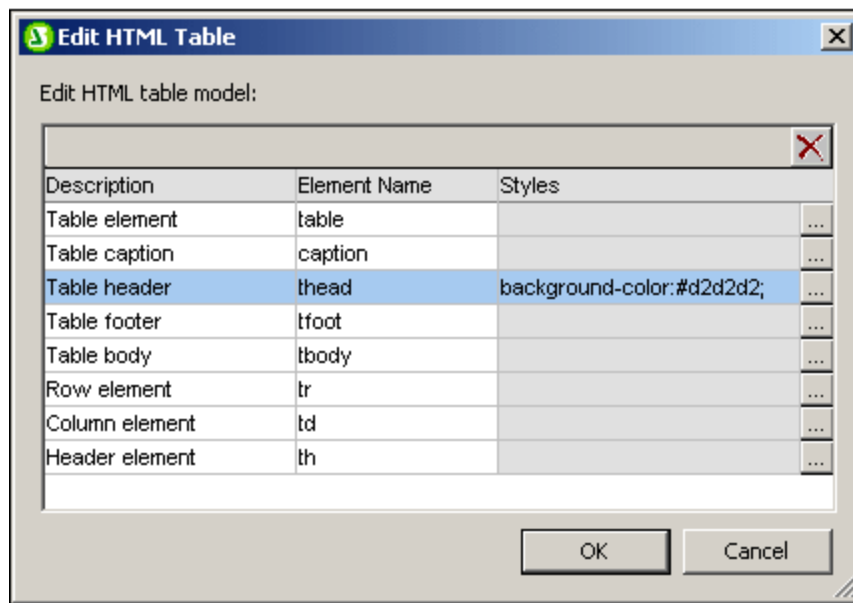
If a CALS/HTML table has been defined and the XML data structure is [correctly inserted](#)¹⁶¹ as a CALS/HTML table, then the data structure will be sent to the output as a table. To **remove a CALS/HTML table**

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).



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](#)⁵⁴ 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.

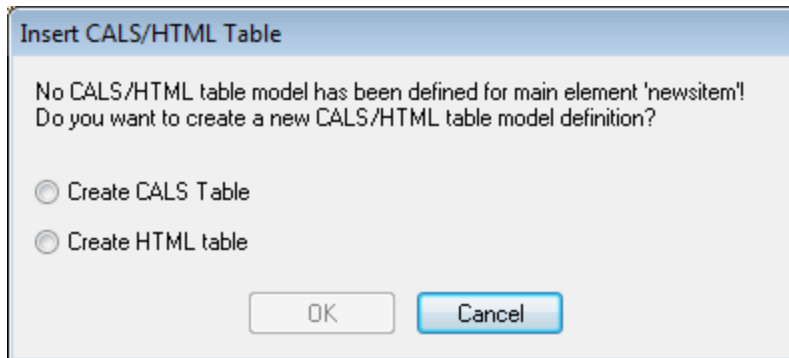
To remove a style that has been assigned to an element in the CALS/HTML table definition, select that element (for example in the screenshot above the `thead` element has been selected) and click the **Delete** button. The styles for that element will be removed.

Inserting a CALS/HTML table in the design

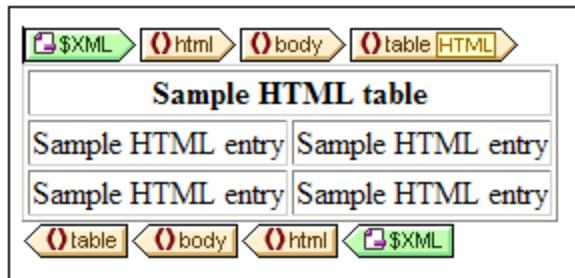
A CALS/HTML table structure can be inserted in the design in two ways:

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](#)¹⁵⁹, the Insert CALS/HTML Tables dialog (*screenshot below*) pops up and you can define the element as a CALS or HTML table.



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*).



Global templates of table elements

If [global templates](#)²⁴⁴ 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 the `Examples` project folder (in the Project window of the GUI).

5.7 Lists

There are two types of lists that can be created in the SPS:

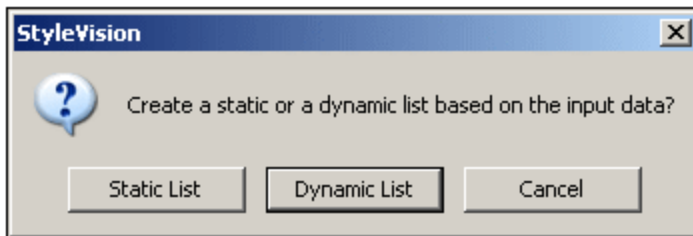
- [Static lists](#)¹⁶³, which are lists, the contents of which are entered directly in the SPS. The list structure is not dynamically derived from the structure of the XML document.
- [Dynamic lists](#)¹⁶⁵, which are lists that derive their structure and contents dynamically from the XML document.

How to create these two list types are described in detail in the sub-sections of this section.

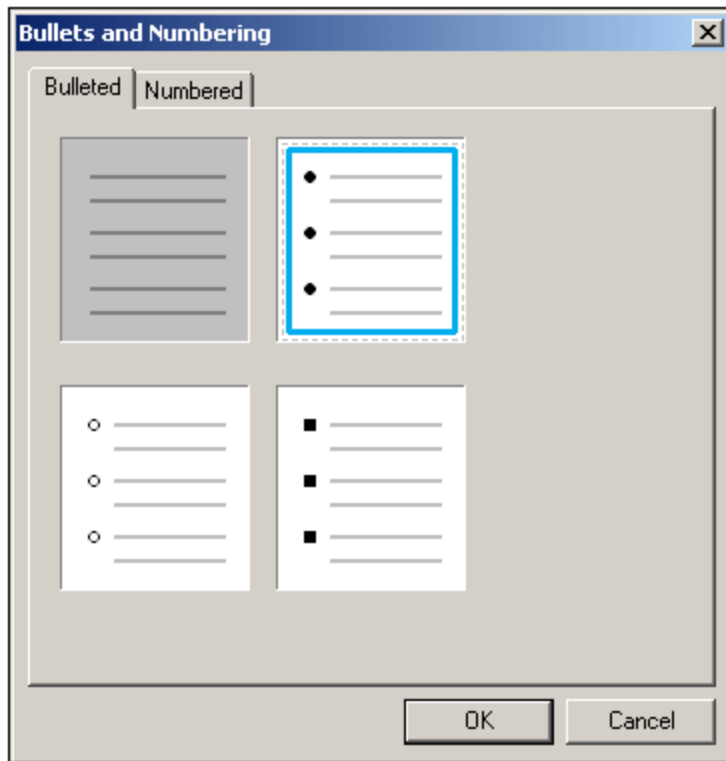
5.7.1 Static Lists

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](#)⁷⁷³ menu command (or click the Bullets and Numbering icon in the [Insert Design Elements toolbar](#)⁷⁰⁸). 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](#)⁷⁷³ 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](#)⁷⁰⁴. 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](#)⁷⁸⁹, 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](#)⁷⁸⁹ command can also be accessed via the context menu.

Lists in Text output

When a list is generated in Text output, each list item is indicated by a minus symbol at the start of the list item.

5.7.2 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.

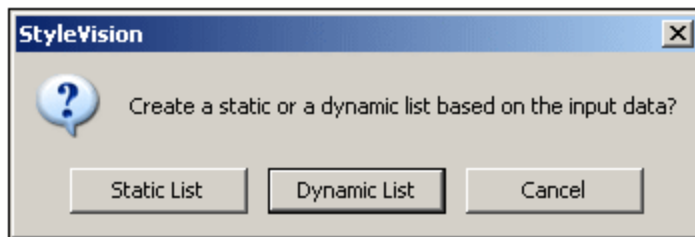
General usage mechanism

- 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.

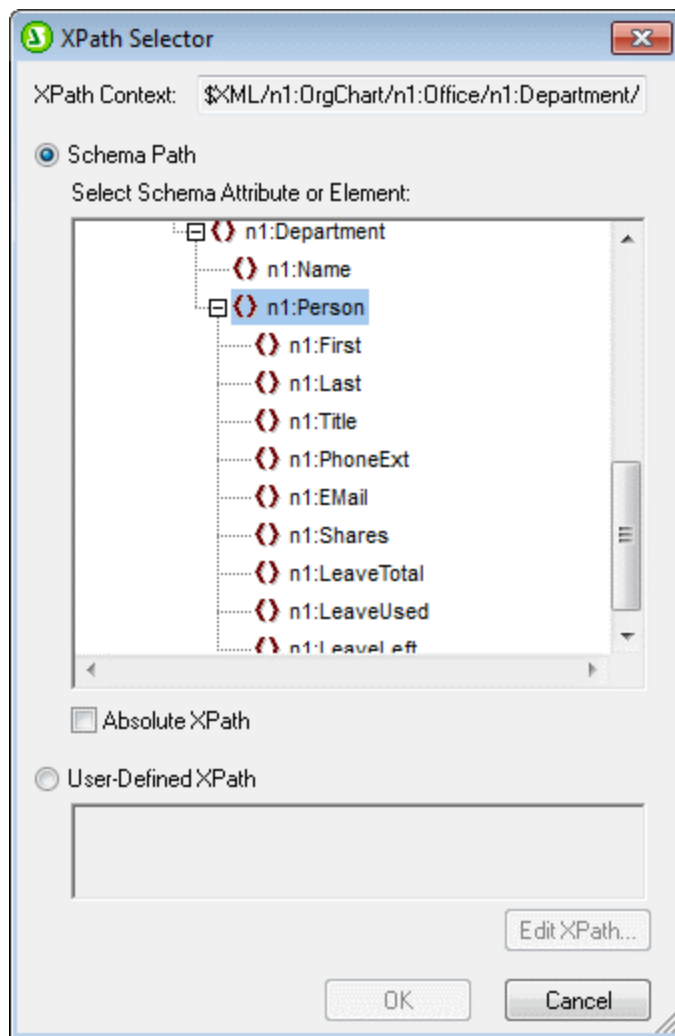
Creating a dynamic list

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](#) ⁷⁷³ menu command. This pops up a dialog asking whether you wish to create a static list or dynamic list (*screenshot below*).

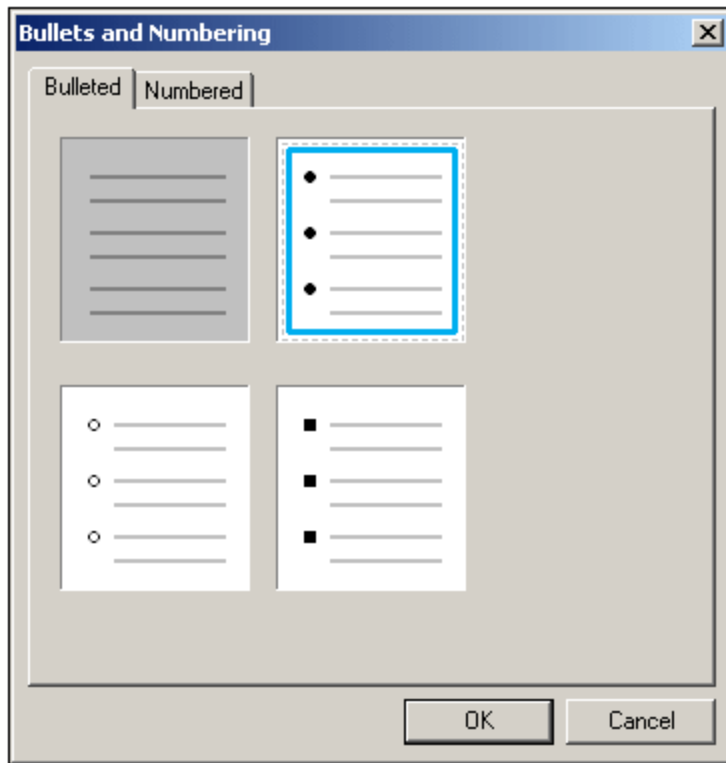


2. Click **Dynamic List**. This pops up the XPath Selector dialog (*screenshot below*).
3. In the XPath Selector dialog, notice that the XPath Context is the context of the insertion location, and that it cannot be changed in the dialog. Select the node that is to be created as the dynamic list. In the 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 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.



Lists in Text output

When a list is generated in Text output, each list item is indicated by a minus symbol at the start of the list item.

5.8 Graphics

There are two ways in which graphics are used in an SPS:

- As [images in the design document](#)¹⁶⁸, and
- As Authentic View toolbar icons for applying markup to the XML document ([custom toolbar buttons](#)⁸⁰⁰).

Note: Graphics are not rendered in Text output.

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](#)¹⁶⁸. What type of images are supported in the various outputs are listed in the section [Image Types and Output](#)¹⁷⁰. The section [Reference | Autnetic Menu | Custom toolbar buttons](#)⁸⁰⁰ describes how toolbar icons for Authentic View can be defined.

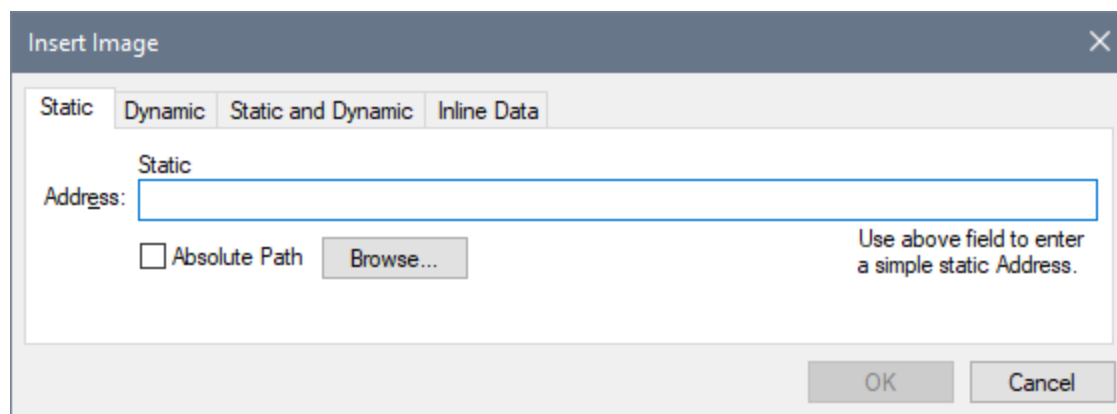
Image properties

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.

5.8.1 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](#)⁷⁷⁰ 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.

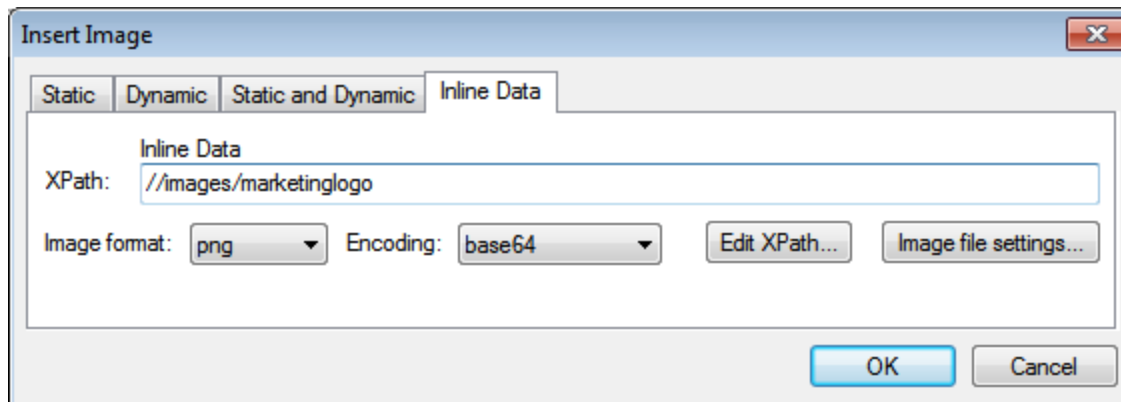
Inserting an image file

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](#)¹⁷⁰), 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](#)⁴⁴⁰. 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](#)⁴⁴⁰.
- 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*).



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.

If you wish to embed an inline image in HTML output, select the *Image Embedding with Data URI Scheme* option in the [Properties dialog](#)⁷³⁸ of the SPS.

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](#)⁷³⁸, 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.
- 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.

5.8.2 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.

Image Type	Authentic	HTML	RTF	PDF	Word 2007+
JPEG	Yes	Yes	Yes	Yes	Yes
GIF	Yes	Yes	Yes	Yes	Yes

PNG	Yes	Yes	Yes	Yes	Yes
BMP	Yes	Yes	Yes	Yes	Yes
TIFF	Yes*	Yes*	Yes	Yes	Yes
SVG	Yes*	Yes*	No	Yes	No
JPEG XR	Yes	Yes	No	No	No

* 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 JIMI 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\\My
  Documents\Altova\StyleVision2024\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](#)⁵⁴. 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 and `auto` 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

In the RTF output, images can either be embedded (when XSLT 2.0 or XSLT 3.0 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](#)⁵⁵). For information about how paths are resolved, see the section [Image URIs](#)¹⁶⁸.

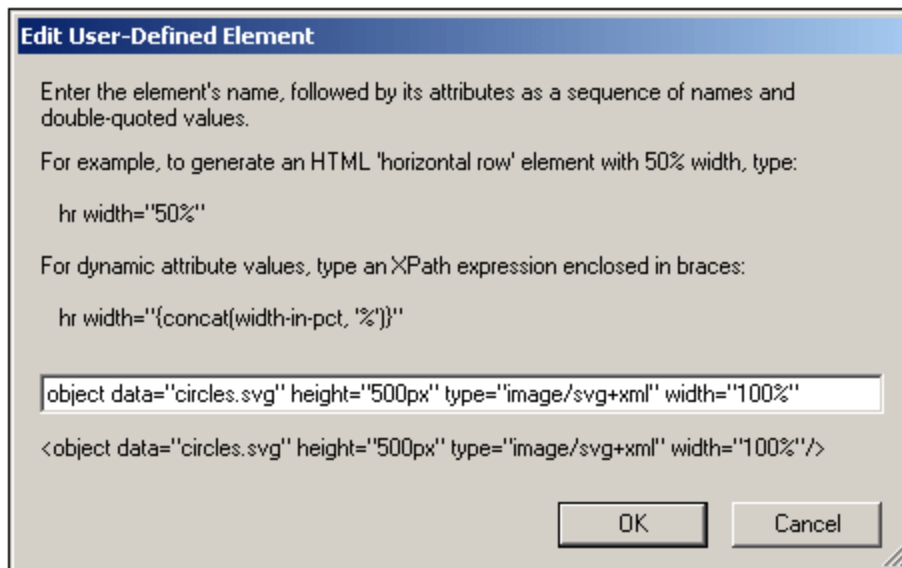
Note: The RTF format supports embedded images only for EMF, JPG, and PNG files.

SVG images in HTML

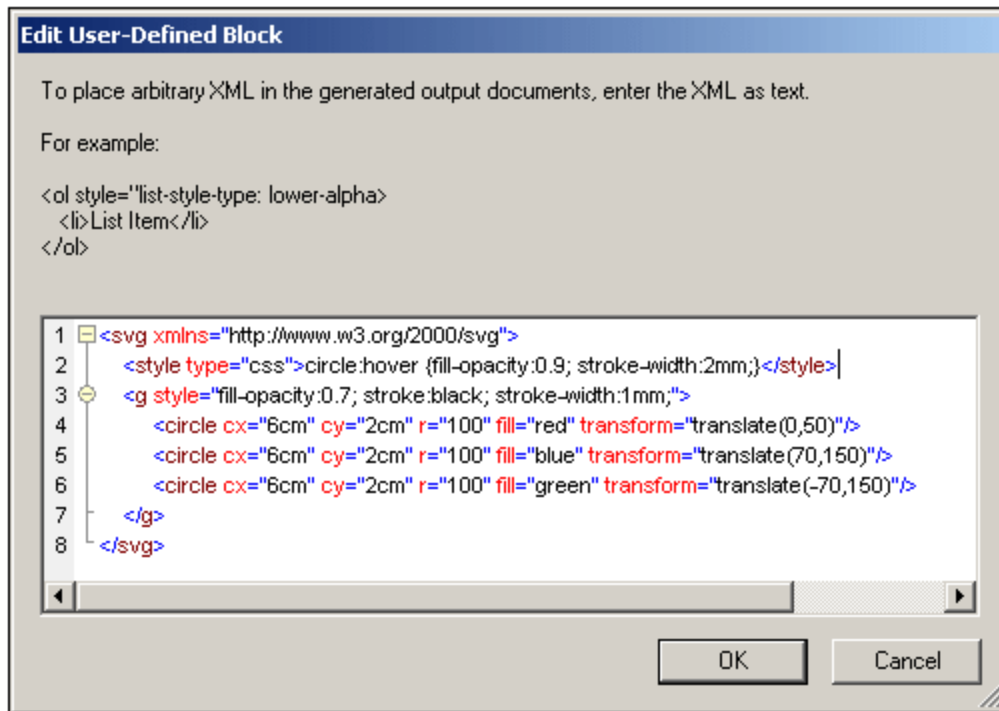
When an external SVG file with code for mouse events is used as an image, the SVG file is rendered within the image and will no longer be interactive. This limitation can be overcome by using the external SVG image file as an object or by including the SVG code fragment as a User-Defined XML Block.

Given below are the three ways in which SVG images can be included in a web page.

1. External SVG [inserted as image](#)¹⁶⁸: This generates an `` in the generated HTML file, and interactivity will be lost.
2. External SVG inserted as an object via the [User-Defined Element feature](#)¹⁴⁰ (see screenshot below). Be sure to insert the `type` attribute correctly: `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](#)¹⁴¹. 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.



5.8.3 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](#)²⁹: C:\Documents and Settings\

- 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 | Autnetic Menu | Custom toolbar buttons](#)⁶⁰⁰ 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.

5.9 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, Text, and RTF 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 (i) an explanation of how data is entered in the XML document for each device and (ii) how the data-entry devices will be output in plain text format.

Data-Entry Device	Data in XML File	Output to Plain Text Format
Input Field (Text Box)	Text entered by user	Text entered by user
Multiline Input Field	Text entered by user	Text entered by user
Combo box	User selection is mapped to a value.	Selected value
Check box	User selection is mapped to a value.	[] (unselected); [x] (selected)
Radio button	User selection is mapped to a value.	() (unselected); (o) (selected)
Button	User selection is mapped to a value.	Button text

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**.

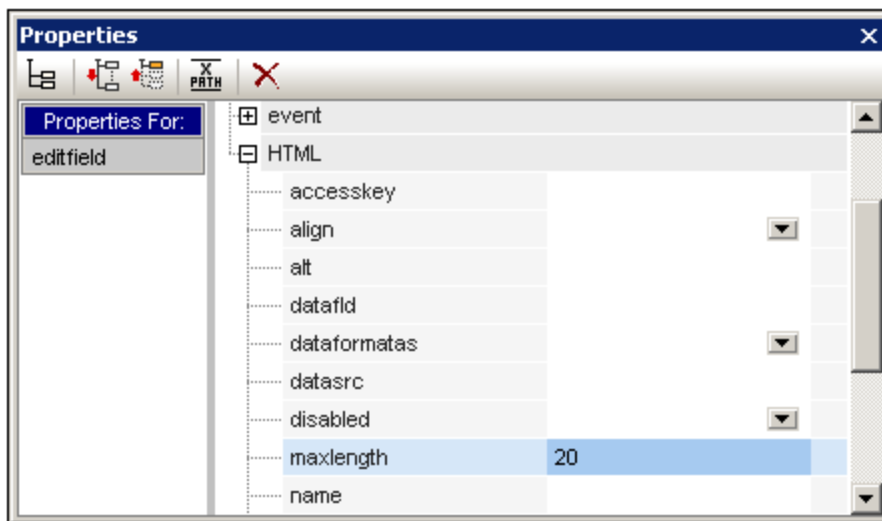
- In the HTML and RTF output, the entry selected by the user is displayed in the output. Changing the value of a data-entry device in the HTML document does not change the text value in either the XML document or HTML document.
- In the case of some data-entry devices, such as check boxes, where the device cannot correctly be rendered in print, an alternative rendition is implemented.

5.9.1 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*).

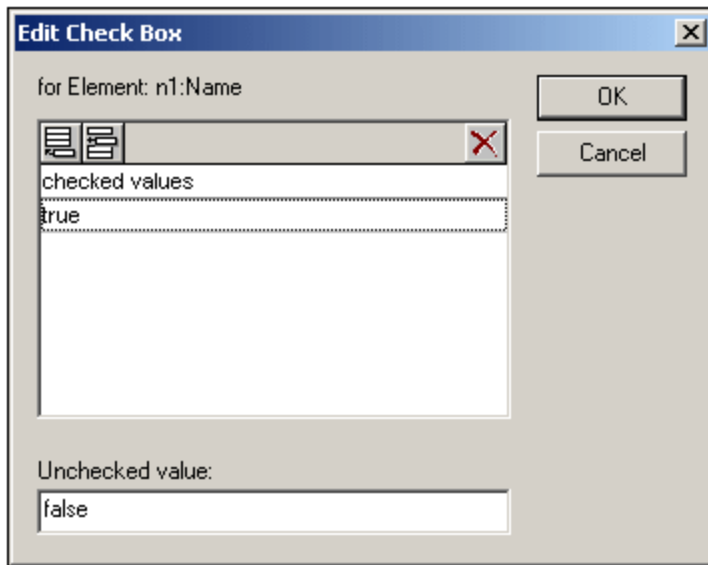


For example, with the input field selected, in the Properties window select `editfield`, select the *HTML* group of properties and the `maxlength` property. Then double-click in the Value field of `maxlength` and enter a value.

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.

5.9.2 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).

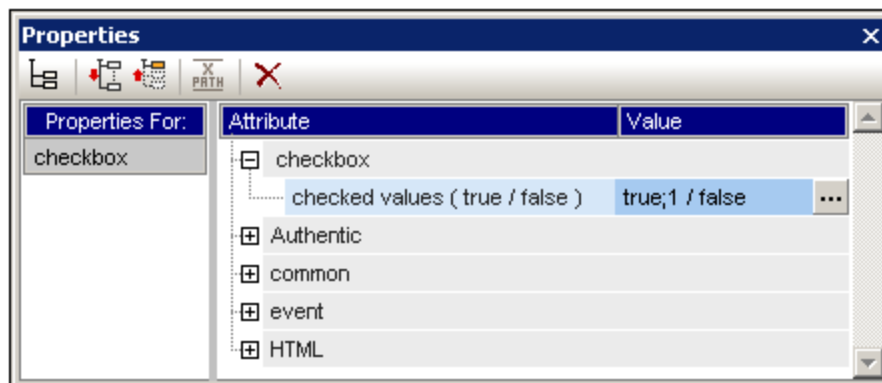
Note: When a new `Name` (or check box) element is created in Authentic View, its XML value is empty (it is not the Unchecked Value). The Unchecked Value is entered only after the check box has first been checked, and then unchecked. To have a default value in a node, create a Template XML file that contains the default value.

Note: Check boxes in Text output are displayed as square brackets: `[]` for unselected check boxes; `[x]` for selected check boxes.

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.

5.9.3 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:

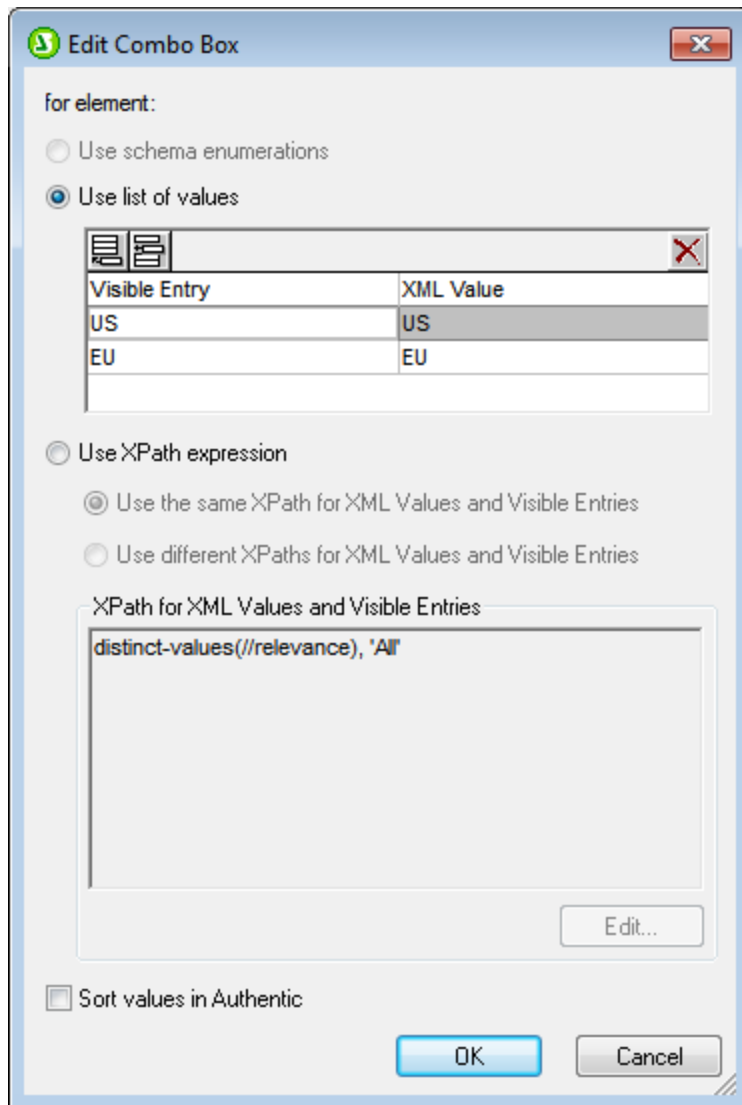
1. Select the combo box in the design.
2. In the Properties sidebar, select the combo box item and then the *combo box* group of properties (see *screenshot below*).



3. Click the Edit button  of the the `content origin` property. This pops up the Edit Combo Box dialog.

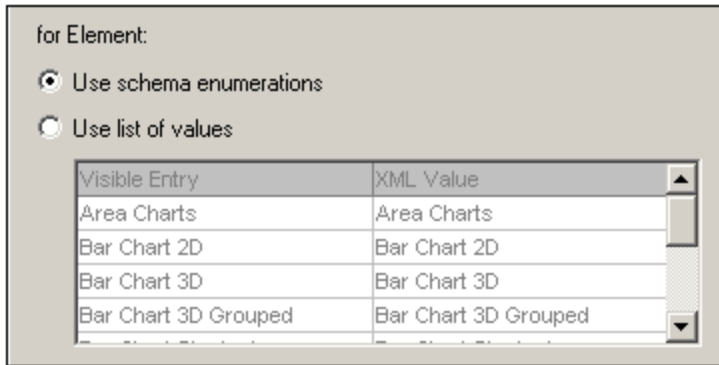
Using the Edit Combo Box dialog

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](#)⁵⁹⁴.
- Combo boxes in Text output displayed the selected value.

5.9.4 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](#)⁴³⁵, 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.

Note: Radio buttons in Text output are displayed as parentheses: for unselected radio buttons; for selected radio buttons. Buttons generate only the button text in Text output.

5.10 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](#)³³⁹, describes how to create static and dynamic bookmarks in the document and how to link to bookmarks as well as to external documents.

Note: Links are not rendered in Text output.

5.11 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](#)¹⁸³ and specify its [properties](#)¹⁸⁵.



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.

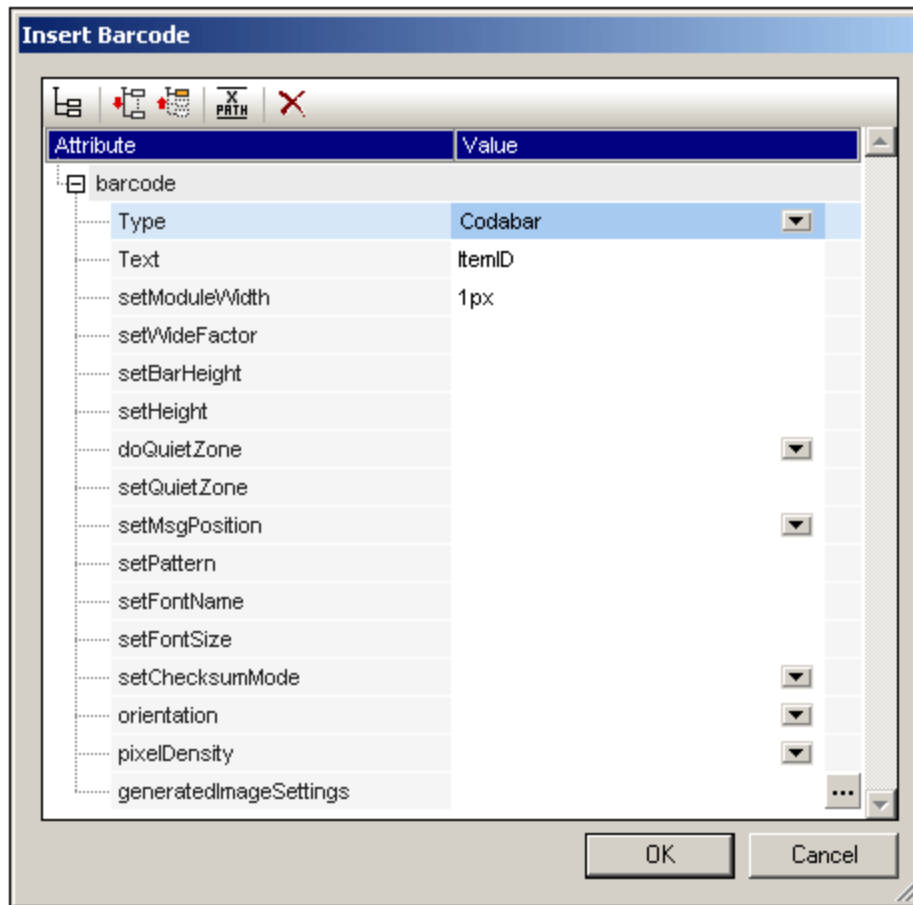
Important: For barcodes to be generated in the output, you **must use Altova's XSLT processor** to generate the output. This is because the barcodes in an SPS are generated by calling special Java extension functions that are not part of the XSLT standard. Altova's XSLT processors support these specific extension functions, whereas other XSLT processors very probably do not. As a result, barcodes will not be generated if processed with a non-Altova XSLT processor. The Altova XSLT processor is packaged with StyleVision, and is automatically called when you generate output via the **Generate** commands in the **File** menu. Alternatively, you can use [RaptorXML Server](#), which is a Altova's standalone XSLT processor.

Note: Barcodes are not rendered in Text output.

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*).



- 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](#)¹⁸⁵. 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](#)⁵⁸ 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](#)⁶⁸⁵. The XPath expression will be evaluated within the current context node.
- 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

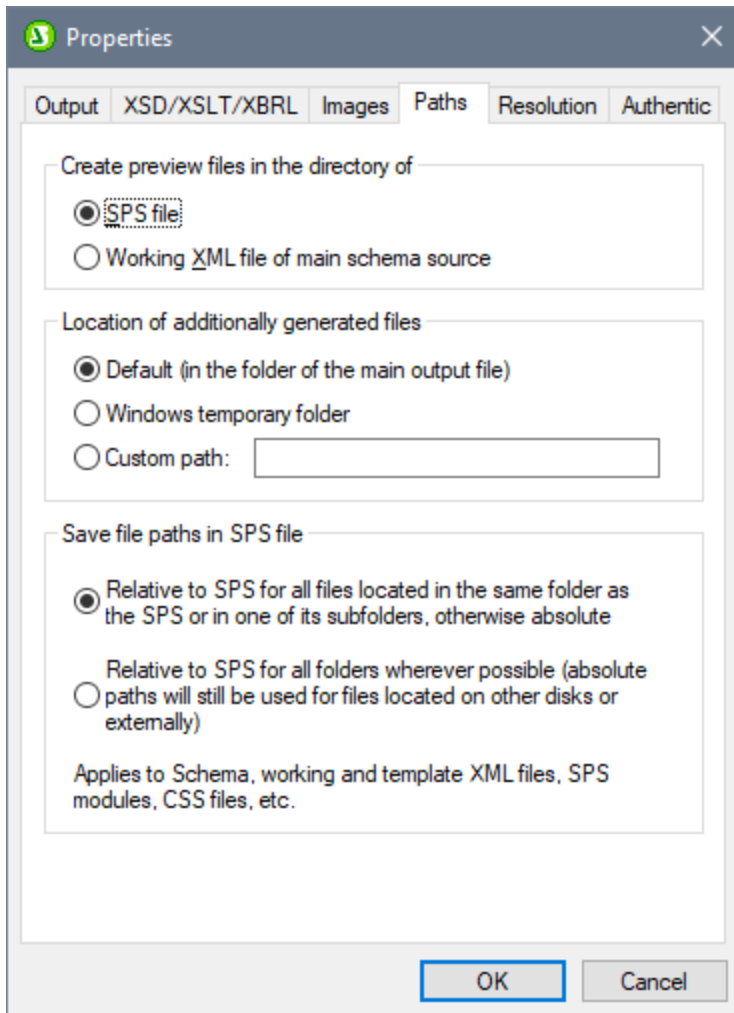
Patterns are used to make the input message string more readable in the barcode. In the pattern, each character of the input message text is represented by the underscore "_". Any other characters included in the pattern are inserted at the corresponding locations in the output message text. The backslash "\" 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.

Input message text	Pattern	Output message text
123456	_ _ _	12 34 56

15032011094655	__ \\ __ \\ __ __ : __ : __ UTC	15\03\2011 09:46:55 UTC
15-03-2011	__ # / __ # / __ __	15/03/2011

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*).

5.12 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](#)¹⁸⁷. 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](#)¹⁹⁰ 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](#)¹⁹⁴ can be drawn, formatted, positioned and moved to the front or back of the stack of layout objects (Layout Boxes and other Lines).

Note: Layout Modules are not rendered in Text output.

Form-based designs

When you [create a new SPS](#)⁷¹⁴ 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.

5.12.1 Layout Containers

A Layout Container has the following properties:

- It can be [inserted](#)¹⁸⁸ 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](#)¹⁸⁸ section below for details.
- A [layout grid](#)¹⁸⁸ and a [zoom feature](#)¹⁸⁹ make the positioning of objects in the Layout Container easier.
- It can have [style properties](#)¹⁸⁹, such as borders, background colors, font-properties for the whole container, etc.
- It can [contain Layout Boxes and Lines](#)¹⁸⁹, but no other design element. (All design elements must be placed within Layout Boxes.)
- It can contain a [blueprint](#)¹⁹⁰, 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

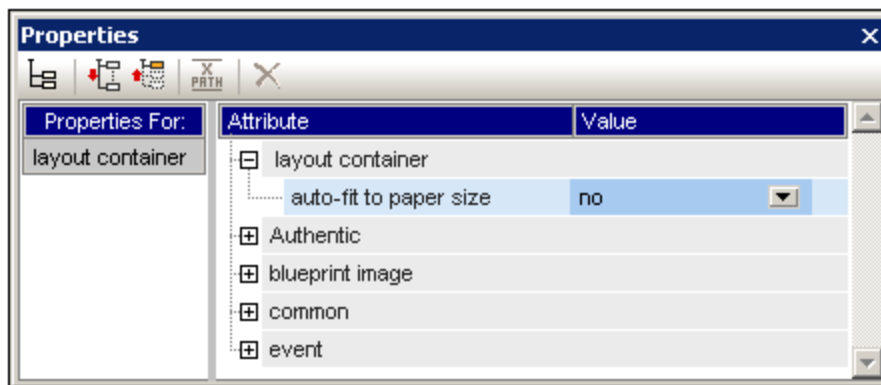
To insert a Layout Container, click the **Insert Layout Container** icon in the [Insert Design Elements toolbar](#)⁷⁰⁸ and click the location in the design where the Layout Container is to be inserted. A dialog appears asking whether you wish to auto-fit the Layout Container to the page. If you click **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](#)⁸³⁹) 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.

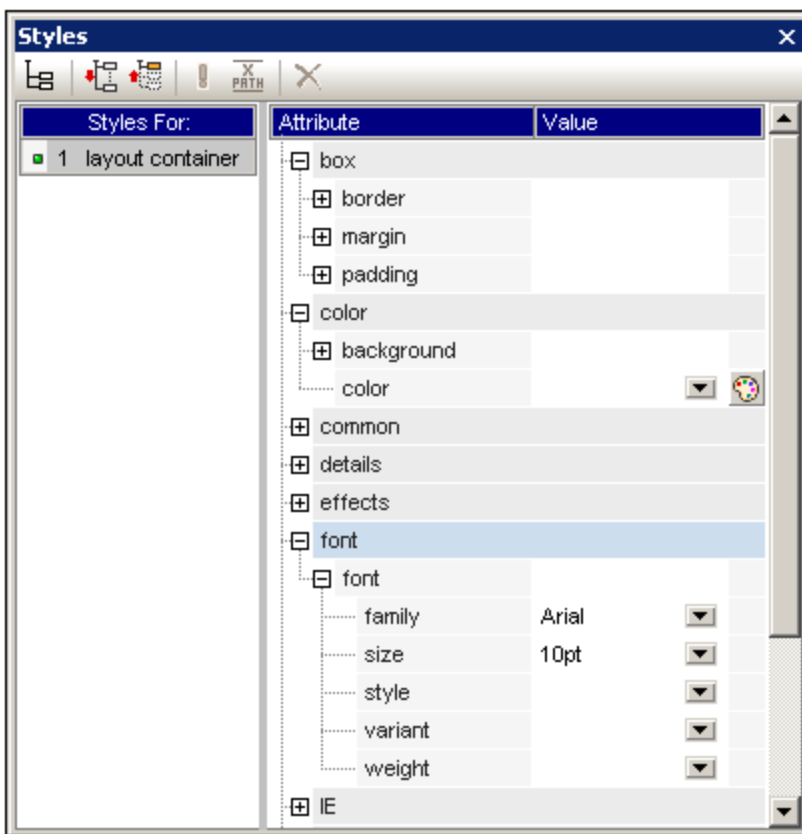
Zooming

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 properties.



The style properties of a Layout Container are set in the *Layout Container* styles in the Styles sidebar (screenshot above).

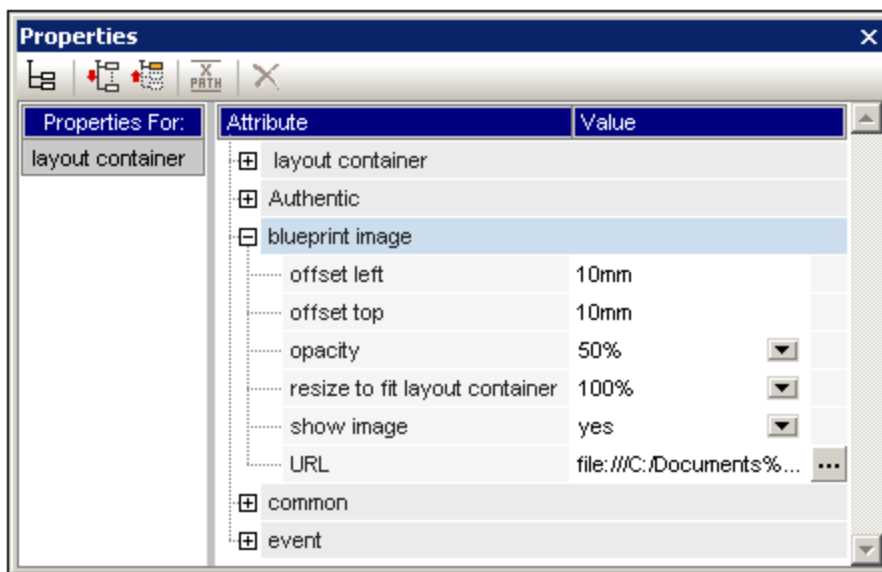
Layout Container contents

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.

5.12.2 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 sub-sections:

- [Inserting Layout Boxes](#) ¹⁹¹
- [Selecting and moving Layout Boxes](#) ¹⁹¹
- [Modifying the size of the Layout Box](#) ¹⁹¹
- [Defining Layout Box style properties](#) ¹⁹²

- [Inserting content in the Layout Box](#) ¹⁹²
- [Stacking order of Layout Boxes](#) ¹⁹³

Inserting a Layout Box

A Layout Box can be inserted only in a [Layout Container](#) ¹⁸⁷. To add a Layout Box, first click the Insert Layout Box icon in the [Insert Design Elements](#) ⁷⁰⁸ 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.

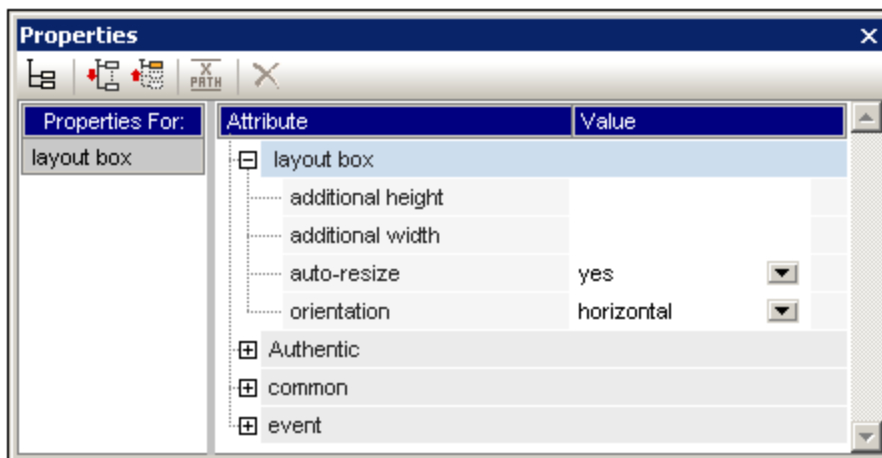
Selecting and moving a Layout Box

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](#) ¹⁹¹. 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.

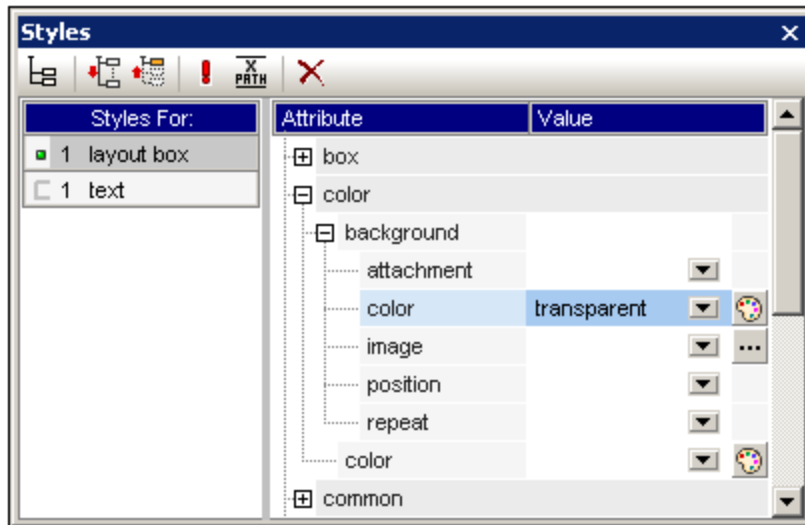


The *Additional Height* and *Additional Width* properties give the lengths that are additional to the optimal dimensions as determined by auto-resizing. The additional lengths are obtained when a Layout Box is manually resized. Conversely, by changing the values of these two properties, the size of the 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.

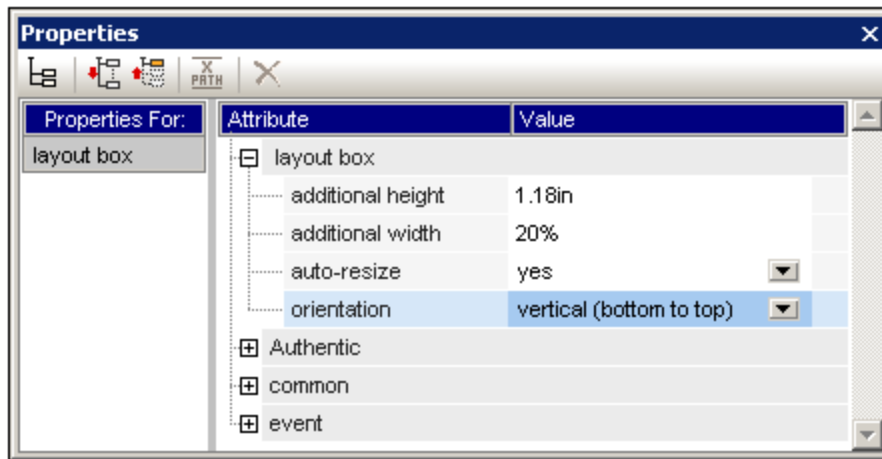


Note: The *background-color* value of `transparent` can be selected in the dropdown list of the property's combo box (it is not available in the color palette). The significance of this value in a situation where the Layout Box is part of a stack is explained below.

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](#)¹⁸⁷ nor a [Layout Line](#)¹⁹⁴ 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.

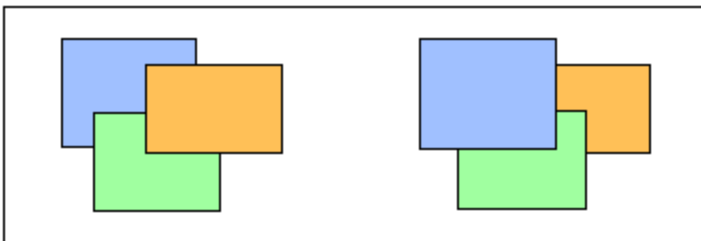


Note the following points:

- 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](#) ¹⁵³.

Stacking order of Layout Boxes

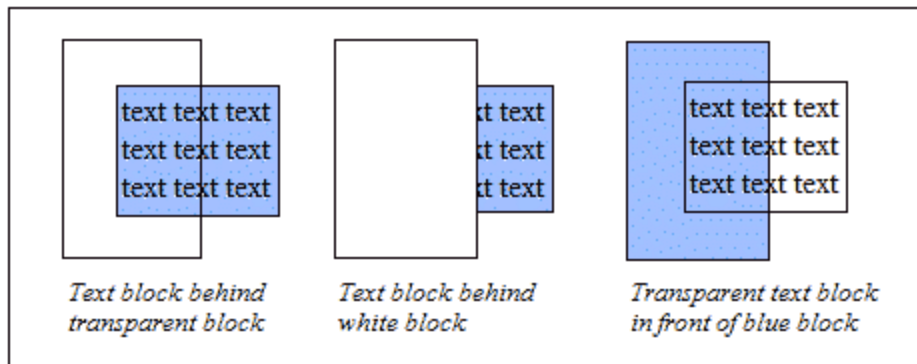
Layout Boxes can be placed one over the other. When one Layout Box is placed on top of another, then, if it is opaque, it hides that part of the Layout Box which it covers. This behavior can be extended to a stack of several Layout Boxes. In such a stack, only the topmost Layout Box will be fully visible; the others will be partially or fully covered.



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.



Note: [Layout Lines](#)¹⁹⁴ can also be added to a stack of Layout Boxes, and each Line can be moved relative to other items in the stack.

5.12.3 Lines

Lines can be [inserted in a Layout Container](#)¹⁹⁴ (but not in Layout Boxes), then [selected, re-sized and moved](#)¹⁹⁴ around within the Layout Container, [assigned properties](#)¹⁹⁵, and [moved backwards and forwards in a stack of layout items](#)¹⁹⁵ 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](#)⁷⁰⁸ 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.

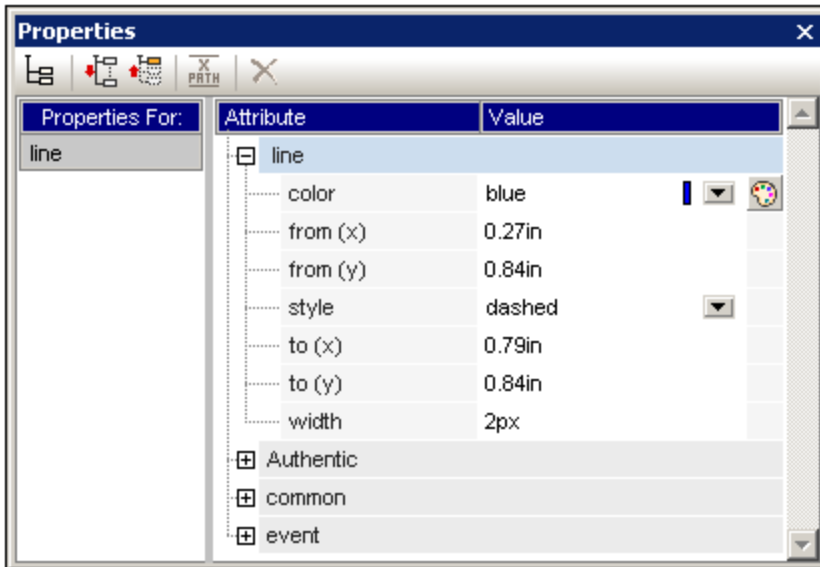
Selecting, moving, and sizing a Line

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.

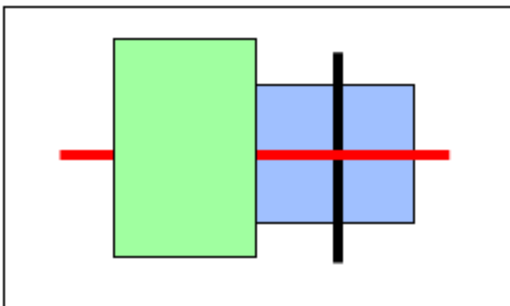


The following Line properties can be edited in the Properties sidebar:

- **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.

5.13 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

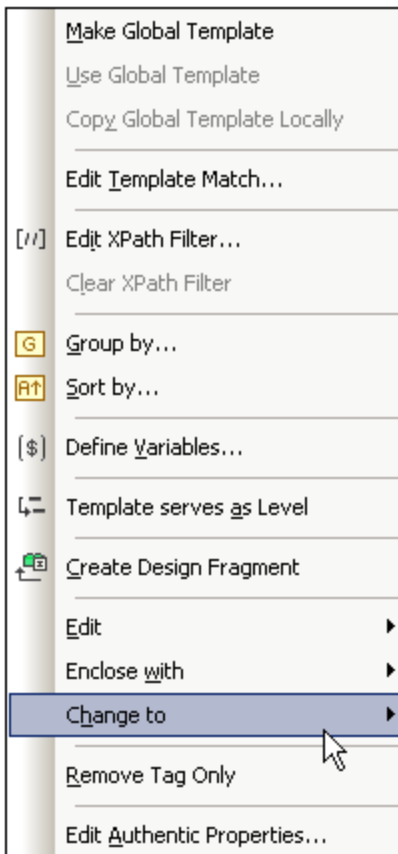
Either a node or its contents can be changed. In the image below left, the node is selected. In the image at right, the node's contents are selected.



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.

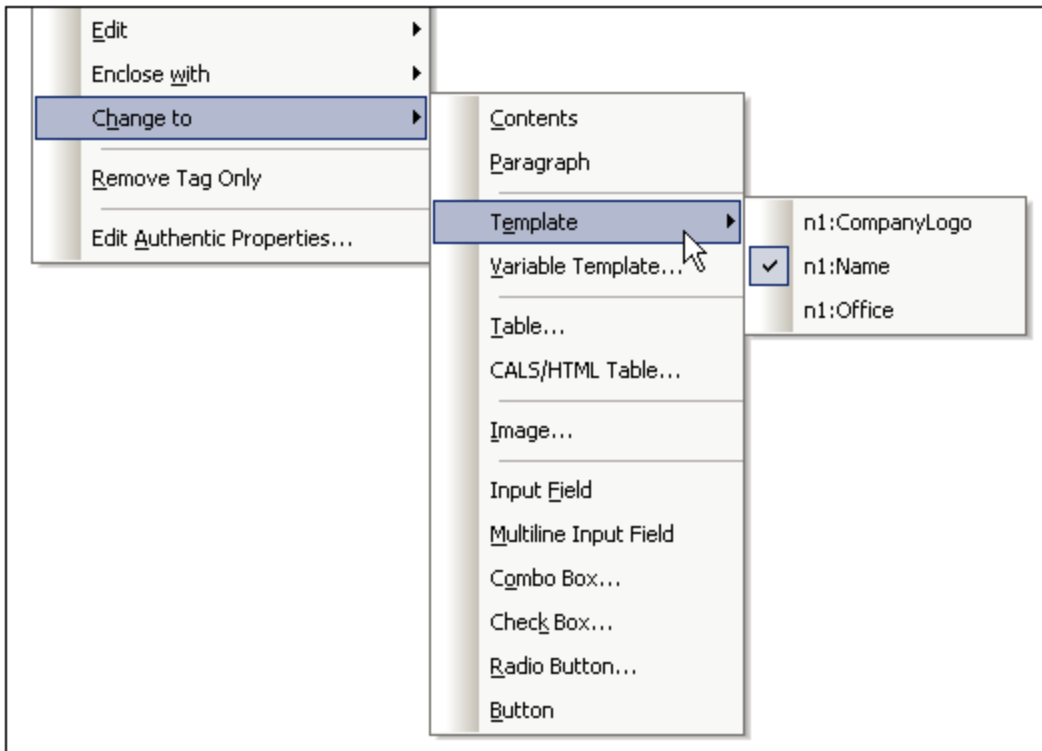
The Change-To command

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 of 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.

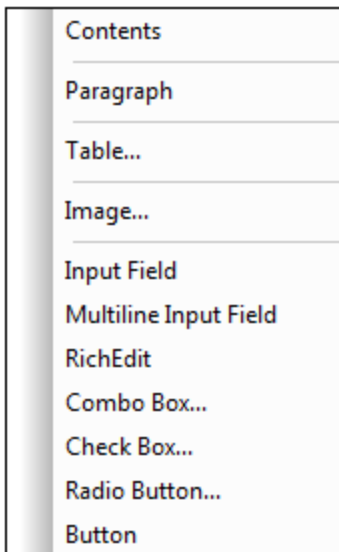


If the selected node has a content model that does not match that described in the template, there will be structural inconsistencies. Such inconsistencies are errors and are indicated with red strikethroughs in the tags of nodes that are invalid.

You can also change the template-match to match, not a node, but a [variable template](#)²⁵¹.

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.

6 SPS Structure

The structure of an SPS document is both input- as well as output-driven, and it is controlled by:

- [Schema sources](#) ²⁰²
- [Modular SPSs](#) ²³⁰
- [Templates and Design Fragments](#) ²⁴⁴

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).

First	Last	Title (sorted by)
Loby	Matise	Accounting Manager
Frank	Further	Accounts Receivable
Vernon	Callaby	Office Manager

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](#) ²⁰².

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](#) ²³⁰.

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](#) ²⁴⁴.

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](#)²⁵⁵.

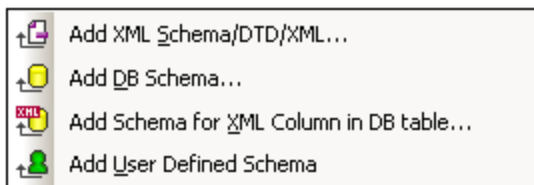
6.1 Schema Sources

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](#)⁴², 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.




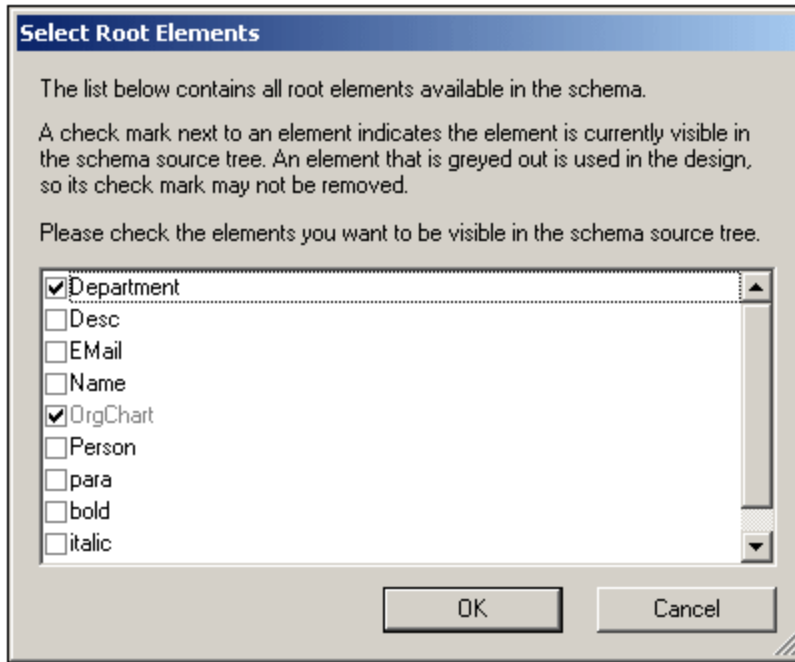
The schema source can be selected from a file, from a DB, or be user-defined. An important point to consider is whether you will be using global templates, and whether elements you wish to create as global templates are defined as global elements in the schema. When adding a DTD from file, remember that 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: 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](#)²⁷, then multiple root elements ([document elements](#)²⁷) 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](#)²⁷), 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](#)²⁷) 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.

6.1.1 DTDs and XML Schemas

An SPS can be based on an XML Schema or DTD. An XML Schema or DTD can be created as a schema source in one of the following ways:

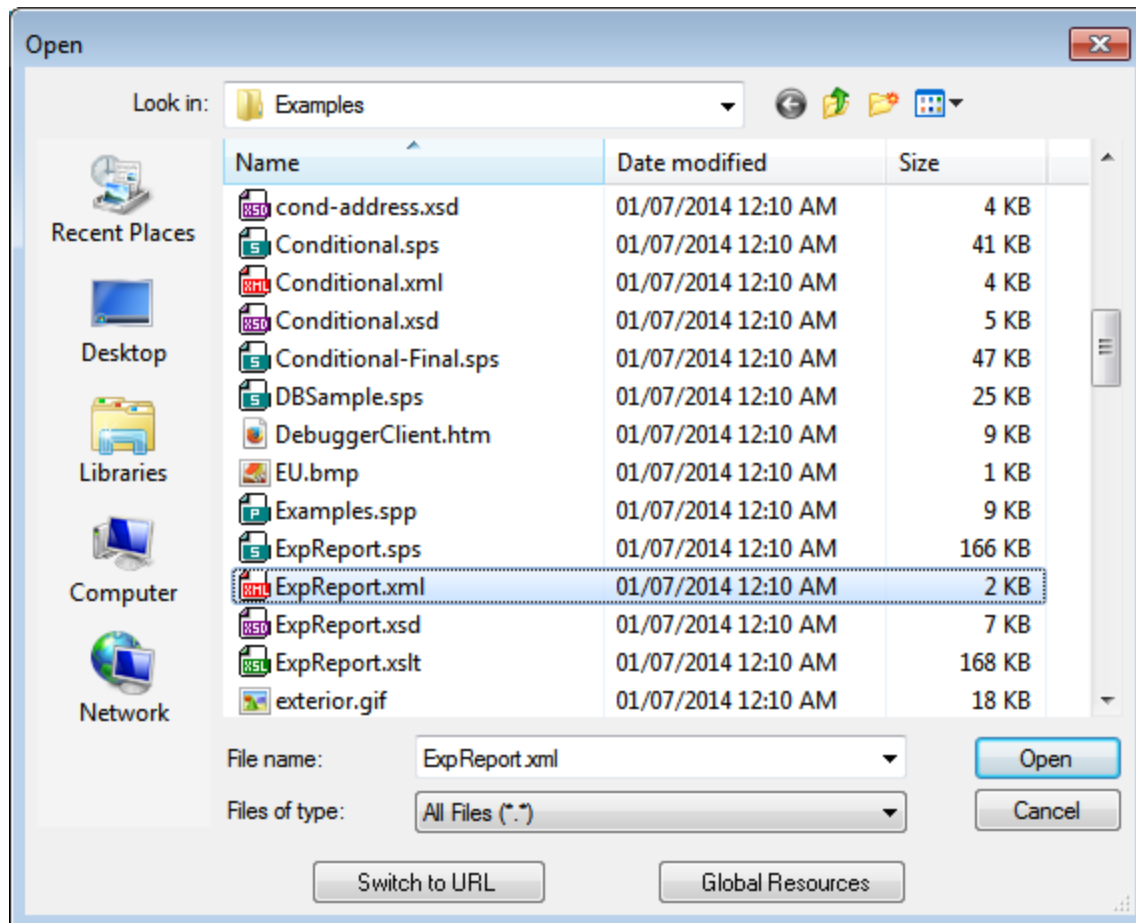
- The XML Schema or DTD is created as a schema source directly when the SPS is created (**File | New | New from XML Schema / DTD / XML**).
- The XML Schema or DTD is added to an empty SPS (in the [Design Overview sidebar](#)⁴²).

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 **Global Resource** to go to one of these selection processes.

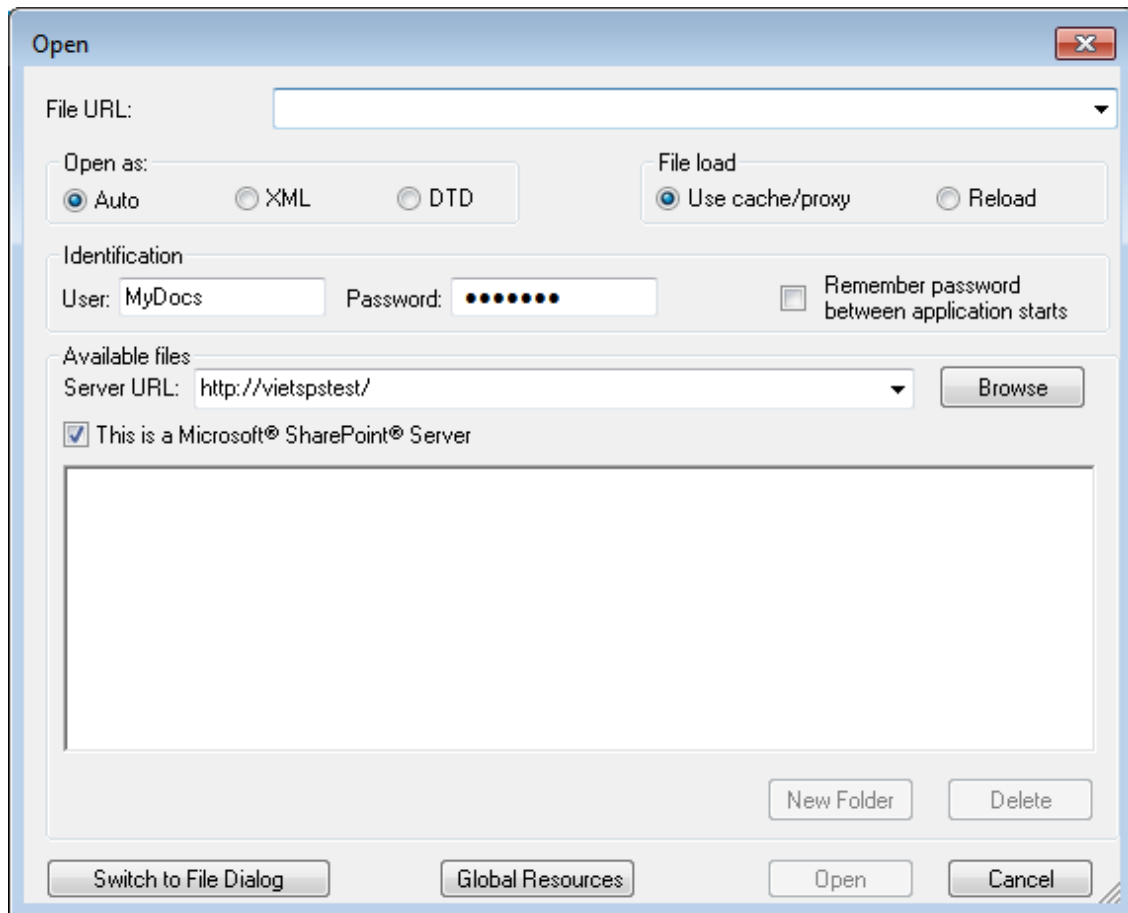


Selecting files via URLs

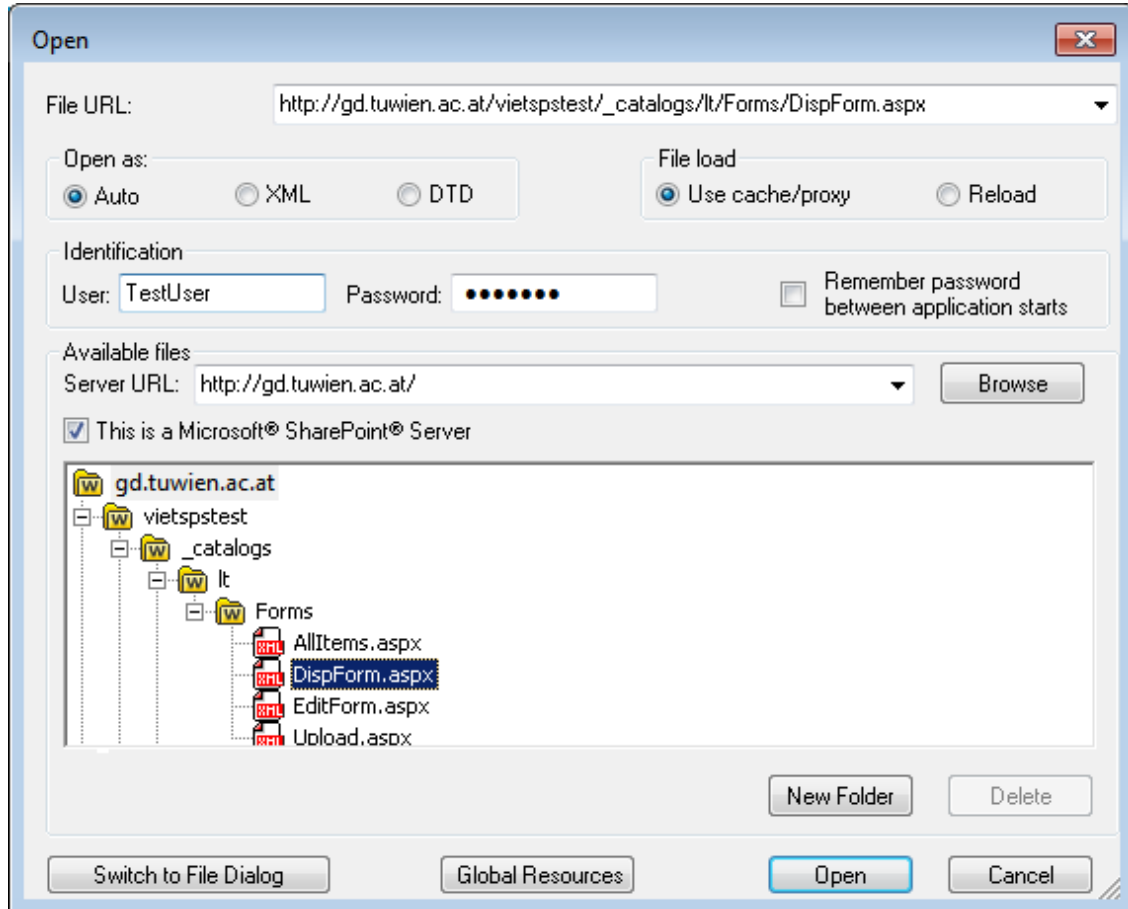
To select a file via a URL (either for opening or saving), do the following:

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.

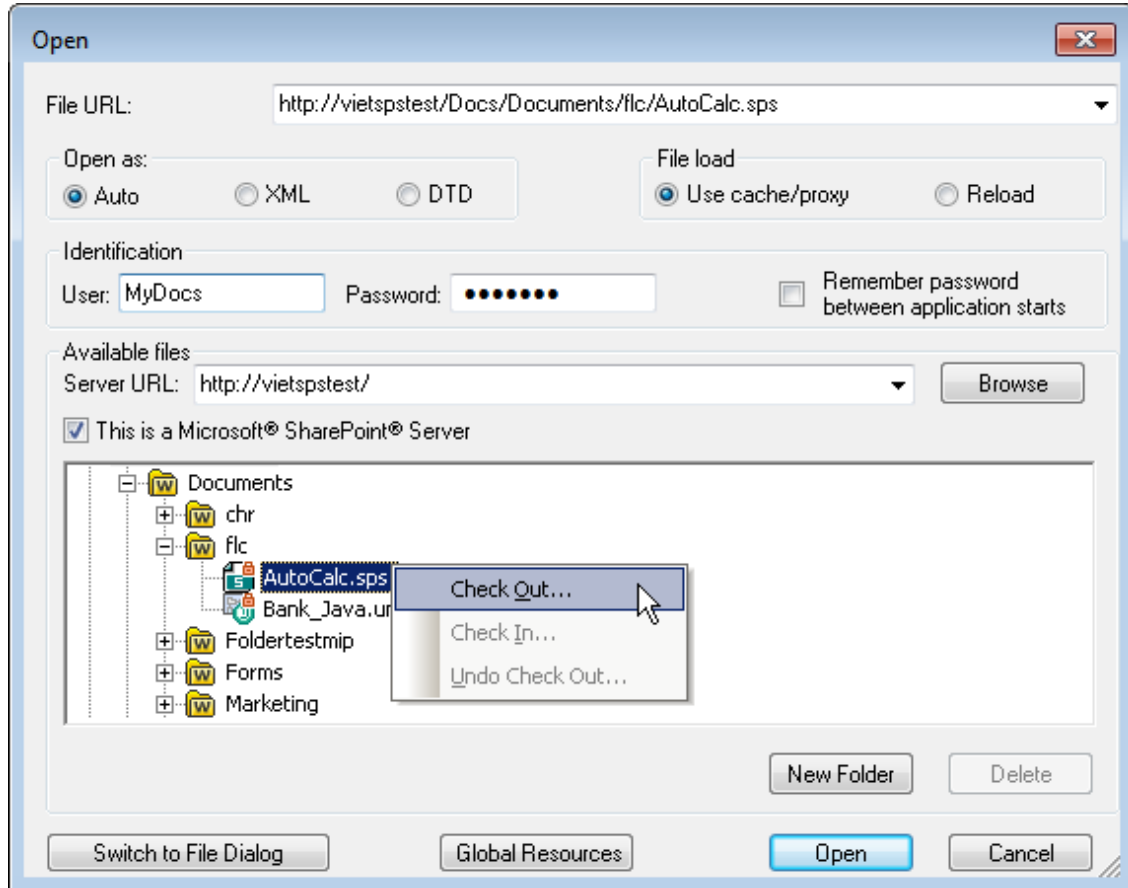
Note the following:

- 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.

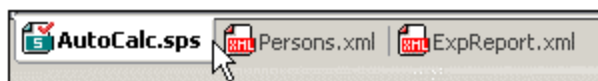


Right-clicking a file pops up a context menu containing commands available for that file (screenshot above).

- 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 **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](#)⁴²⁷. For a general description of Global Resources, see the [Global Resources](#)⁴¹⁶ 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`.

6.1.2 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 (in the [Design Overview sidebar](#)⁴²).

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](#)⁴⁴, 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](#)⁴⁹². 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](#)⁵⁷⁵. 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.

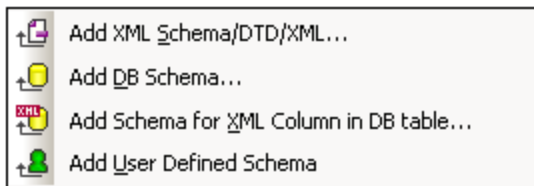
After the schema source appears in the Schema Tree window, you can start designing the SPS. A temporary XML File is created each time an output preview tab is clicked. The XML file is structured according to the generated XML Schema and contains data from the selected DB tables.

6.1.3 User-Defined Schemas

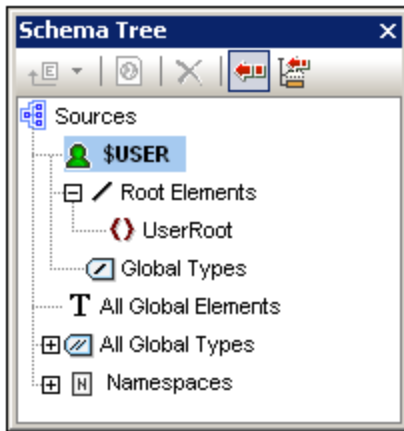
You can quickly create a user-defined schema in the [Schema Tree sidebar](#)⁴⁴. This is useful if you have an XML document that is not based on any schema and you wish to create an SPS for this XML document.


To add and create a user-defined schema, in the Schema Tree sidebar, do the following:

1. Click **File | New | New (empty)**. In the [Design Overview sidebar](#)⁴², click the **Add New Source** command (under the Sources heading), and select **Add User-Defined Schema** (*screenshot below*).

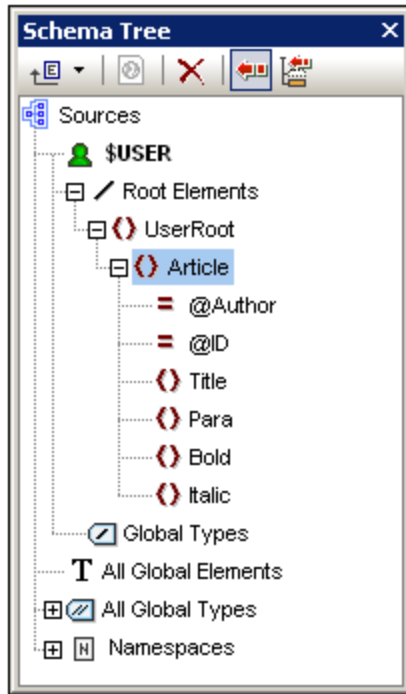



The new schema is created and is indicated with the parameter `$USER` (*screenshot below*).

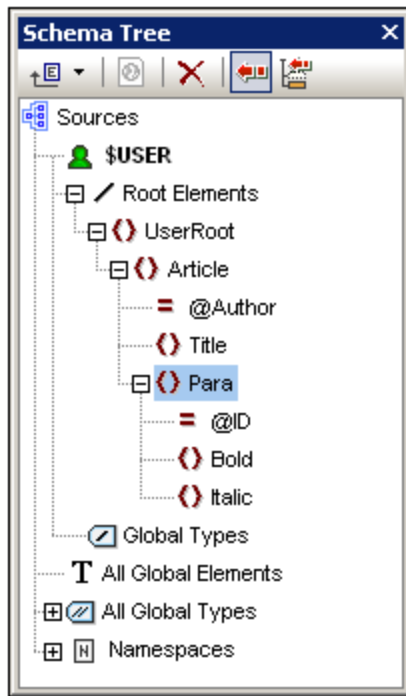


2. In the Root Elements tree, there is a single [root element \(document element\)](#)²⁷ called `UserRoot`.
3. Double-click `UserRoot` and rename it to match the [document element](#)²⁷ of the XML document for which you are building this schema.
4. To assign a child element or an attribute to the document element, select the document element (`UserRoot`), and click, respectively, (i) the drop-Append New Element icon  in the toolbar of the [Schema Tree sidebar](#)⁴⁴; 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  appears, drop the dragged node. It will be created as a "child" of `Para` (screenshot below).



- 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  in the toolbar. Alternatively, select **Remove** from the context menu.

6.1.4 Schema Manager

XML Schema Manager is an Altova tool that provides a centralized way to install and manage XML schemas (DTDs for XML and XML Schemas) for use across all Altova's XML-Schema-aware applications, including StyleVision.

- On Windows, Schema Manager has a graphical user interface (*screenshot below*) and is also available at the command line. (Altova's desktop applications are available on Windows only; *see list below*.)
- On Linux and macOS, Schema Manager is available at the command line only. (Altova's server applications are available on Windows, Linux, and macOS; *see list below*.)



Altova applications that operate with Schema Manager

Desktop applications (Windows only)	Server applications (Windows, Linux, macOS)
XMLSpy (all editions)	RaptorXML Server, RaptorXML+XBRL Server

MapForce (all editions)	StyleVision Server
StyleVision (all editions)	
Authentic Desktop Enterprise Edition	

Installation and de-installation of Schema Manager

Schema Manager is installed automatically when you first install a new version of Altova Mission Kit or of any of Altova's XML-schema-aware applications (see *table above*).

Likewise, it is removed automatically when you uninstall the last Altova XML-schema-aware application from your computer.

Schema Manager features

Schema Manager provides the following features:

- Shows XML schemas installed on your computer and checks whether new versions are available for download.
- Downloads newer versions of XML schemas independently of the Altova product release cycle. (Altova stores schemas online, and you can download them via Schema Manager.)
- Install or uninstall any of the multiple versions of a given schema (or all versions if necessary).
- An XML schema may have dependencies on other schemas. When you install or uninstall a particular schema, Schema Manager informs you about dependent schemas and will automatically install or remove them as well.
- Schema Manager uses the [XML catalog](#) mechanism to map schema references to local files. In the case of large XML schemas, processing will therefore be faster than if the schemas were at a remote location.
- All major schemas are available via Schema Manager and are regularly updated for the latest versions. This provides you with a convenient single resource for managing all your schemas and making them readily available to all of Altova's XML-schema-aware applications.
- Changes made in Schema Manager take effect for all Altova products installed on that machine.
- In an Altova product, if you attempt to validate on a schema that is not installed but which is available via Schema Manager, then installation is triggered automatically. However, if the schema package contains namespace mappings, then there will be no automatic installation; in this case, you must start Schema Manager, select the package/s you want to install, and run the installation. If, after installation, your open Altova application does not restart automatically, then you must restart it manually.

How it works

Altova stores all XML schemas used in Altova products online. This repository is updated when new versions of the schemas are released. Schema Manager displays information about the latest available schemas when invoked in both its GUI form as well as on the CLI. You can then install, upgrade or uninstall schemas via Schema Manager.

Schema Manager also installs schemas in one other way. At the Altova website (<https://www.altova.com/schema-manager>) you can select a schema and its dependent schemas that you want to install. The website will prepare a file of type `.altova_xmlschemas` for download that contains information about your schema selection. When you double-click this file or pass it to Schema Manager via the CLI as an argument of the `install` ²²² command, Schema Manager will install the schemas you selected.

Local cache: tracking your schemas

All information about installed schemas is tracked in a centralized cache directory on your computer, located here:

<i>Windows</i>	C:\ProgramData\Altova\pkgs\.cache
<i>Linux</i>	/var/opt/Altova/pkgs\.cache
<i>macOS</i>	/var/Altova/pkgs

This cache directory is updated regularly with the latest status of schemas at Altova's online storage. These updates are carried out at the following times:

- Every time you start Schema Manager.
- When you start StyleVision for the first time on a given calendar day.
- If StyleVision is open for more than 24 hours, the cache is updated every 24 hours.
- You can also update the cache by running the [update](#)²²⁵ command at the command line interface.

The cache therefore enables Schema Manager to continuously track your installed schemas against the schemas available online at the Altova website.

Do not modify the cache manually!

The local cache directory is maintained automatically based on the schemas you install and uninstall. It should not be altered or deleted manually. If you ever need to reset Schema Manager to its original "pristine" state, then, on the command line interface (CLI): (i) run the [reset](#)²²³ command, and (ii) run the [initialize](#)²²¹ command. (Alternatively, run the `reset` command with the `--i` option.)

6.1.4.1 Run Schema Manager

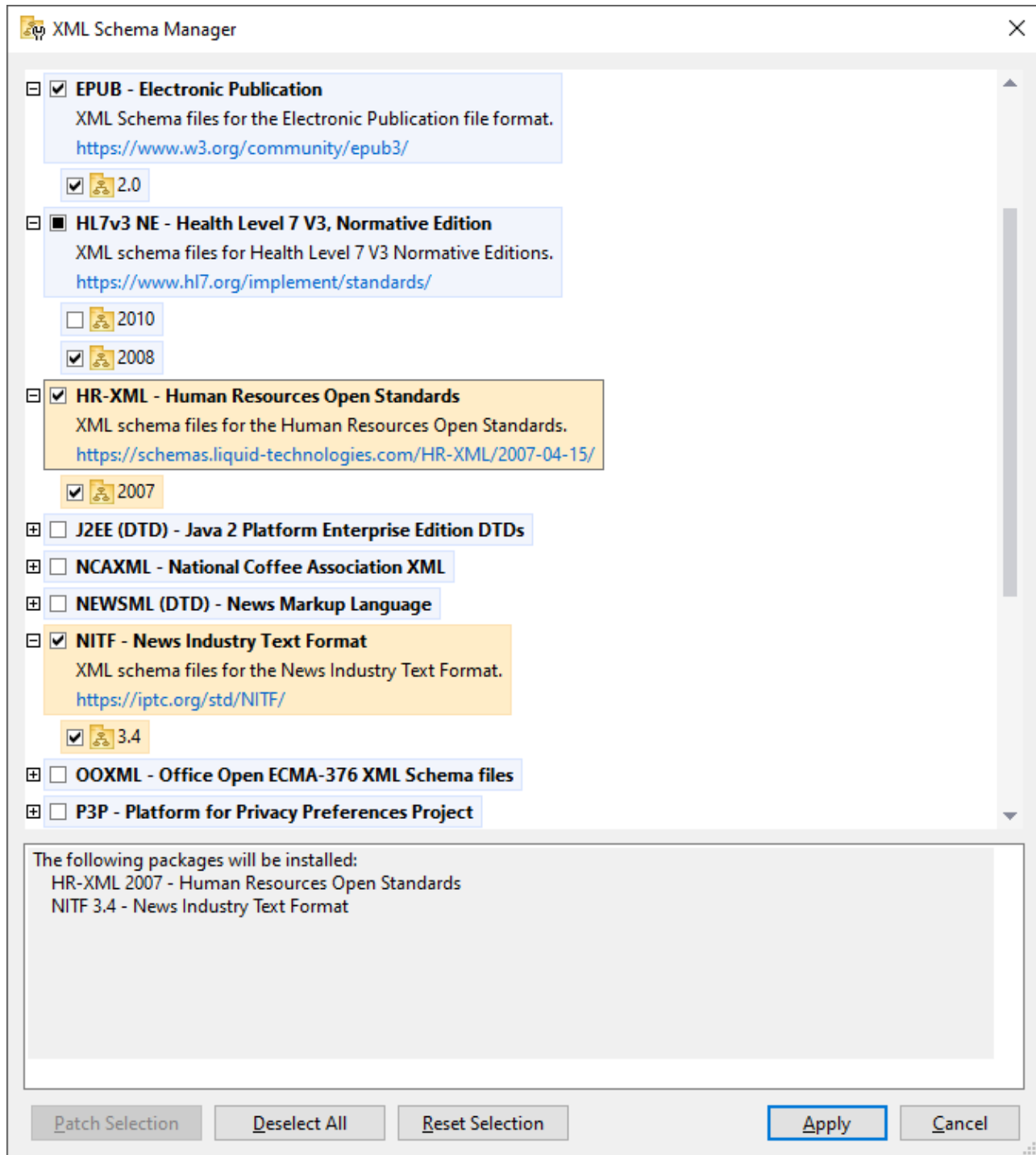
Graphical User Interface

You can access the GUI of Schema Manager in any of the following ways:

- *During the installation of StyleVision:* Towards the end of the installation procedure, select the check box *Invoke Altova XML-Schema Manager* to access the Schema Manager GUI straight away. This will enable you to install schemas during the installation process of your Altova application.
- *After the installation of StyleVision:* After your application has been installed, you can access the Schema Manager GUI at any time, via the menu command **Tools | XML Schema Manager**.
- Via the `.altova_xmlschemas` file downloaded from the [Altova website](#): Double-click the downloaded file to run the Schema Manager GUI, which will be set up to install the schemas you selected (at the website) for installation.

After the Schema Manager GUI (*screenshot below*) has been opened, already installed schemas will be shown selected. If you want to install an additional schema, select it. If you want to uninstall an already installed schema, deselect it. After you have made your selections and/or deselections, you are ready to apply your changes. The schemas that will be installed or uninstalled will be highlighted and a message about the

upcoming changes will be posted to the Messages pane at the bottom of the Schema Manager window (see *screenshot*).



Command line interface

You can run Schema Manager from a command line interface by sending commands to its executable file, `xmlschemamanager.exe`.

The `xmlschemamanager.exe` file is located in the following folder:

- *On Windows:* C:\ProgramData\Altova\SharedBetweenVersions
- *On Linux or macOS (server applications only):* %INSTALLDIR%/bin, where %INSTALLDIR% is the program's installation directory.

You can then use any of the commands listed in the [CLI command reference section](#)²²⁰.

To display help for the commands, run the following:


- *On Windows:* `xmlschemamanager.exe --help`
- *On Linux or macOS (server applications only):* `sudo ./xmlschemamanager --help`

6.1.4.2 Status Categories

Schema Manager categorizes the schemas under its management as follows:

- *Installed schemas.* These are shown in the GUI with their check boxes selected (*in the screenshot below the checked and blue versions of the EPUB and HL7v3 NE schemas are installed schemas*). If all the versions of a schema are selected, then the selection mark is a tick. If at least one version is unselected, then the selection mark is a solid colored square. You can deselect an installed schema to **uninstall** it; (*in the screenshot below, the DocBook DTD is installed and has been deselected, thereby preparing it for de-installation*).
- *Uninstalled available schemas.* These are shown in the GUI with their check boxes unselected. You can select the schemas you want to **install**.



- *Upgradeable schemas* are those which have been revised by their issuers since they were installed. They are indicated in the GUI by a  icon. You can **patch** an installed schema with an available revision.

Points to note

- In the screenshot above, both CBCR schemas are checked. The one with the blue background is already installed. The one with the yellow background is uninstalled and has been selected for installation. Note that the HL7v3 NE 2010 schema is not installed and has not been selected for installation.
- A yellow background means that the schema will be modified in some way when the **Apply** button is clicked. If a schema is unchecked and has a yellow background, it means that it will be uninstalled when the **Apply** button is clicked. In the screenshot above the DocBook DTD has such a status.
- When running Schema Manager from the command line, the `list` ²²² command is used with different options to list different categories of schemas:

<code>xmlschemamanager.exe list</code>	Lists all installed and available schemas; upgradeables are also indicated
<code>xmlschemamanager.exe list -i</code>	Lists installed schemas only; upgradeables are also indicated
<code>xmlschemamanager.exe list -u</code>	Lists upgradeable schemas




Note: On Linux and macOS, use `sudo ./xmlschemamanager list`

6.1.4.3 Patch or Install a Schema

Patch an installed schema

Occasionally, XML schemas may receive patches (upgrades or revisions) from their issuers. When Schema Manager detects that patches are available, these are indicated in the schema listings of Schema Manager and you can install the patches quickly.

In the GUI

Patches are indicated by the  icon. (Also see the previous topic about [status categories](#)²¹⁶.) If patches are available, the **Patch Selection** button will be enabled. Click it to select and prepare all patches for installation. In the GUI, the icon of each schema that will be patched changes from  to , and the Messages pane at the bottom of the dialog lists the patches that will be applied. When you are ready to install the selected patches, click **Apply**. All patches will be applied together. Note that if you deselect a schema marked for patching, you will actually be uninstalling that schema.

On the CLI

To apply a patch at the command line interface:

1. Run the `list -u`²²² command. This lists any schemas for which upgrades are available.
2. Run the `upgrade`²²⁵ command to install all the patches.

Install an available schema

You can install schemas using either the Schema Manager GUI or by sending Schema Manager the install instructions via the command line.

Note: If the current schema references other schemas, the referenced schemas are also installed.

In the GUI

To install schemas using the Schema Manager GUI, select the schemas you want to install and click **Apply**.

You can also select the schemas you want to install at the [Altova website](#) and generate a downloadable `.altova_xmlschemas` file. When you double-click this file, it will open Schema Manager with the schemas you wanted pre-selected. All you will now have to do is click **Apply**.

On the CLI

To install schemas via the command line, run the `install`²²² command:

```
xmlschemamanager.exe install [options] Schema+
```

where `Schema` is the schema (or schemas) you want to install or a `.altova_xmlschemas` file. A schema is referenced by an identifier of format `<name>-<version>`. (The identifiers of schemas are displayed when you run the `list`²²² command.) You can enter as many schemas as you like. For details, see the description of the `install`²²² command.

Note: On Linux or macOS, use the `sudo ./xmlschemamanager` command.

Installing a required schema

When you run an XML-schema-related command in StyleVision and StyleVision discovers that a schema it needs for executing the command is not present or is incomplete, Schema Manager will display information about the missing schema/s. You can then directly install any missing schema via Schema Manager.

In the Schema Manager GUI, you can view all previously installed schemas at any time by running Schema Manager from **Tools | Schema Manager**.

6.1.4.4 Uninstall a Schema, Reset

Uninstall a schema

You can uninstall schemas using either the Schema Manager GUI or by sending Schema Manager the uninstall instructions via the command line.

Note: If the schema you want to uninstall references other schemas, then the referenced schemas are also uninstalled.

In the GUI

To uninstall schemas in the Schema Manager GUI, clear their check boxes and click **Apply**. The selected schemas and their referenced schemas will be uninstalled.

To uninstall all schemas, click **Deselect All** and click **Apply**.

On the CLI

To uninstall schemas via the command line, run the `uninstall`²²⁴ command:

```
xmlschemamanager.exe uninstall [options] Schema+
```

where each `schema` argument is a schema you want to uninstall or a `.altova_xmlschemas` file. A schema is specified by an identifier that has a format of `<name>-<version>`. (The identifiers of schemas are displayed when you run the `list`²²² command.) You can enter as many schemas as you like. For details, see the description of the `uninstall`²²⁴ command.

Note: On Linux or macOS, use the `sudo ./xmlschemamanager` command.

Reset Schema Manager

You can reset Schema Manager. This removes all installed schemas and the cache directory.

- In the GUI, click **Reset Selection**.
- On the CLI, run the `reset`²²³ command.

After running this command, make sure to run the [initialize](#)²²¹ command in order to recreate the cache directory. Alternatively, run the [reset](#)²²³ command with the `-i` option.

Note that [reset -i](#)²²³ restores the original installation of the product, so it is recommended to run the [update](#)²²⁵ command after performing a reset. Alternatively, run the [reset](#)²²³ command with the `-i` and `-u` options.

6.1.4.5 Command Line Interface (CLI)

To call Schema Manager at the command line, you need to know the path of the executable. By default, the Schema Manager executable is installed here:

```
C:\ProgramData\Altova\SharedBetweenVersions\XMLSchemaManager.exe
```

Note: On Linux and macOS systems, once you have changed the directory to that containing the executable, you can call the executable with `sudo ./xmlschemamanager`. The prefix `./` indicates that the executable is in the current directory. The prefix `sudo` indicates that the command must be run with root privileges.

Command line syntax

The general syntax for using the command line is as follows:

```
<exec> -h | --help | --version | <command> [options] [arguments]
```

In the listing above, the vertical bar `|` separates a set of mutually exclusive items. The square brackets `[]` indicate optional items. Essentially, you can type the executable path followed by either `--h`, `--help`, or `--version` options, or by a command. Each command may have options and arguments. The list of commands is described in the following sections.

6.1.4.5.1 help

This command provides contextual help about commands pertaining to Schema Manager executable.

Syntax

```
<exec> help [command]
```

Where `[command]` is an optional argument which specifies any valid command name.

Note the following:

- You can invoke help for a command by typing the command followed by `-h` or `--help`, for example:

```
<exec> list -h
```
- If you type `-h` or `--help` directly after the executable and before a command, you will get general help (not help for the command), for example:

```
<exec> -h list
```

Example

The following command displays help about the `list` command:

```
xmlschemamanager help list
```

6.1.4.5.2 info

This command displays detailed information for each of the schemas supplied as a `Schema` argument. This information for each submitted schema includes the title, version, description, publisher, and any referenced schemas, as well as whether the schema has been installed or not.

Syntax

```
<exec> info [options] Schema+
```

- The `Schema` argument is the name of a schema or a part of a schema's name. (To display a schema's package ID and detailed information about its installation status, you should use the [list](#)²²² command.)
- Use `<exec> info -h` to display help for the command.

Example

The following command displays information about the latest `DocBook-DTD` and `NITF` schemas:

```
xmlschemamanager info doc nitf
```

6.1.4.5.3 initialize

This command initializes the Schema Manager environment. It creates a cache directory where information about all schemas is stored. Initialization is performed automatically the first time a schema-cognizant Altova application is installed. You would not need to run this command under normal circumstances, but you would typically need to run it after executing the `reset` command.

Syntax

```
<exec> initialize | init [options]
```

Options

The `initialize` command takes the following options:

<code>--silent, --s</code>	Display only error messages. The default is <code>false</code> .
<code>--verbose, --v</code>	Display detailed information during execution. The default is <code>false</code> .
<code>--help, --h</code>	Display help for the command.

Example

The following command initializes Schema Manager:

```
xmlschemamanager initialize
```

6.1.4.5.4 install

This command installs one or more schemas.

Syntax

```
<exec> install [options] Schema+
```

To install multiple schemas, add the `schema` argument multiple times.

The `schema` argument is one of the following:

- A schema identifier (having a format of `<name>-<version>`, for example: `cbr-2.0`). To find out the schema identifiers of the schemas you want, run the [list](#)²²² command. You can also use an abbreviated identifier if it is unique, for example `docbook`. If you use an abbreviated identifier, then the latest version of that schema will be installed.
- The path to a `.altova_xmlschemas` file downloaded from the Altova website. For information about these files, see [Introduction to SchemaManager: How It Works](#)²¹¹.

Options

The `install` command takes the following options:

<code>--silent, --s</code>	Display only error messages. The default is <code>false</code> .
<code>--verbose, --v</code>	Display detailed information during execution. The default is <code>false</code> .
<code>--help, --h</code>	Display help for the command.

Example

The following command installs the CBCR 2.0 (Country-By-Country Reporting) schema and the latest DocBook DTD:

```
xmlschemamanager install cbr-2.0 docbook
```

6.1.4.5.5 list

This command lists schemas under the management of Schema Manager. The list displays one of the following

- All available schemas
- Schemas containing in their name the string submitted as a `Schema` argument
- Only installed schemas
- Only schemas that can be upgraded

Syntax

```
<exec> list | ls [options] Schema?
```

If no `Schema` argument is submitted, then all available schemas are listed. Otherwise, schemas are listed as specified by the submitted options (see *example below*). Note that you can submit the `schema` argument multiple times.

Options

The `list` command takes the following options:

<code>--installed, --i</code>	List only installed schemas. The default is <code>false</code> .
<code>--upgradeable, --u</code>	List only schemas where upgrades (patches) are available. The default is <code>false</code> .
<code>--help, --h</code>	Display help for the command.

Examples

- To list all available schemas, run: `xmlschemamanager list`
- To list installed schemas only, run: `xmlschemamanager list -i`
- To list schemas that contain either "doc" or "nitf" in their name, run: `xmlschemamanager list doc nitf`

6.1.4.5.6 reset

This command removes all installed schemas and the cache directory. You will be completely resetting your schema environment. After running this command, be sure to run the [initialize](#)²²¹ command to recreate the cache directory. Alternatively, run the `reset` command with the `-i` option. Since `reset -i` restores the original installation of the product, we recommend that you run the [update](#)²²⁵ command after performing a reset and initialization. Alternatively, run the `reset` command with both the `-i` and `-u` options.

Syntax

```
<exec> reset [options]
```

Options

The `reset` command takes the following options:

<code>--init, --i</code>	Initialize Schema Manager after reset. The default is <code>false</code> .
<code>--update, --u</code>	Updates the list of available schemas in the cache. The default is <code>false</code> .

<code>--silent, --s</code>	Display only error messages. The default is <code>false</code> .
<code>--verbose, --v</code>	Display detailed information during execution. The default is <code>false</code> .
<code>--help, --h</code>	Display help for the command.

Examples

- To reset Schema Manager, run: `xmlschemamanager reset`
- To reset Schema Manager and initialize it, run: `xmlschemamanager reset -i`
- To reset Schema Manager, initialize it, and update its schema list, run: `xmlschemamanager reset -i -u`

6.1.4.5.7 uninstall

This command uninstalls one or more schemas. By default, any schemas referenced by the current one are uninstalled as well. To uninstall just the current schema and keep the referenced schemas, set the option `--k`.

Syntax

```
<exec> uninstall [options] Schema+
```

To uninstall multiple schemas, add the `schema` argument multiple times.

The `schema` argument is one of the following:

- A schema identifier (having a format of `<name>-<version>`, for example: `cbcr-2.0`). To find out the schema identifiers of the schemas that are installed, run the `list -i` ²²² command. You can also use an abbreviated schema name if it is unique, for example `docbook`. If you use an abbreviated name, then all schemas that contain the abbreviation in its name will be uninstalled.
- The path to a `.altova_xmlschemas` file downloaded from the Altova website. For information about these files, see [Introduction to SchemaManager: How It Works](#) ²¹¹.

Options

The `uninstall` command takes the following options:

<code>--keep-references, --k</code>	Set this option to keep referenced schemas. The default is <code>false</code> .
<code>--silent, --s</code>	Display only error messages. The default is <code>false</code> .
<code>--verbose, --v</code>	Display detailed information during execution. The default is <code>false</code> .
<code>--help, --h</code>	Display help for the command.

Example

The following command uninstalls the CBCR 2.0 and EPUB 2.0 schemas and their dependencies:

```
xmlschemamanager uninstall cbcr-2.0 epub-2.0
```


The following command uninstalls the `eba-2.10` schema but not the schemas it references:

```
xmlschemamanager uninstall --k cbc-2.0
```

6.1.4.5.8 update

This command queries the list of schemas available from the online storage and updates the local cache directory. You should not need to run this command unless you have performed a [reset](#)²²³ and [initialize](#)²²¹.

Syntax

```
<exec> update [options]
```

Options

The `update` command takes the following options:

<code>--silent, --s</code>	Display only error messages. The default is <code>false</code> .
<code>--verbose, --v</code>	Display detailed information during execution. The default is <code>false</code> .
<code>--help, --h</code>	Display help for the command.

Example

The following command updates the local cache with the list of latest schemas:

```
xmlschemamanager update
```

6.1.4.5.9 upgrade

This command upgrades all installed schemas that can be upgraded to the latest available *patched* version. You can identify upgradeable schemas by running the [list -u](#)²²² command.

Note: The `upgrade` command removes a deprecated schema if no newer version is available.

Syntax

```
<exec> upgrade [options]
```

Options

The `upgrade` command takes the following options:

<code>--silent, --s</code>	Display only error messages. The default is <code>false</code> .
<code>--verbose, --v</code>	Display detailed information during execution. The default is <code>false</code> .

`--help, --h` Display help for the command.

6.2 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](#)²⁷⁰ that uses the `doc()` function can, therefore, also be used to merge XML data (see *example below*).

Example

The [\(My\) Documents folder](#)²⁹, C:\Documents and Settings\\My Documents\Altova\StyleVision2024\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 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*).

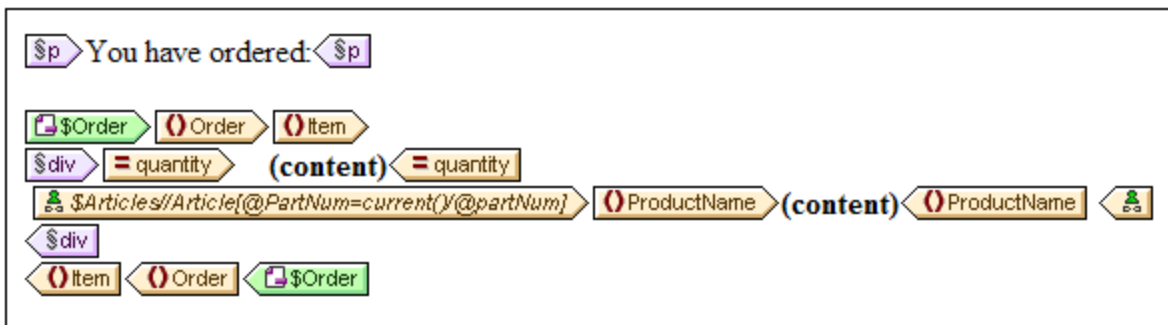
```
<?xml version="1.0" encoding="UTF-8"?>
<Articles xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="MergeArticles.xsd">
  <Article PartNum="833-AA">
    <ProductName>Lapis necklace</ProductName>
    <Price>99.95</Price>
  </Article>
  <Article PartNum="748-OT">
    <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](#)²⁴⁸ 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.

```

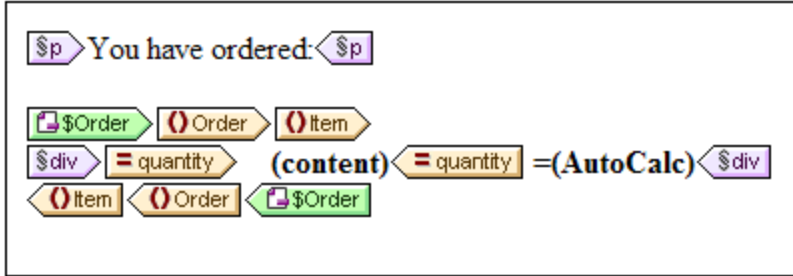
You have ordered:

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](#)²⁷⁰ (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.



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).

6.3 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.

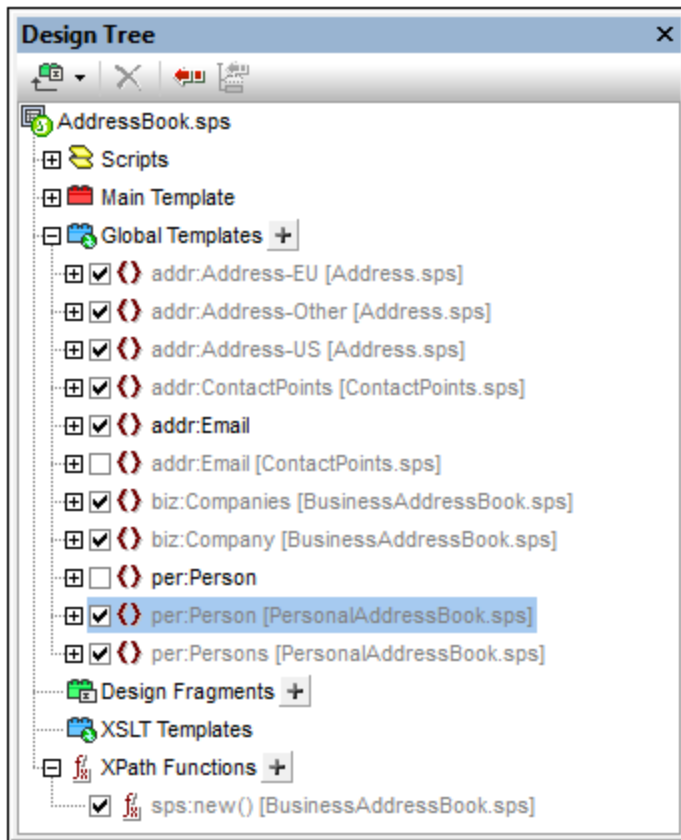
In any given SPS, one or more SPSs can be added as modules. Some types of components (or objects) in these modules are then available to the importing (or referring) SPS.

Available module objects

The section, [Available Module Objects](#)²³¹, 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](#)²³⁴ 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](#)²³⁴.

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.

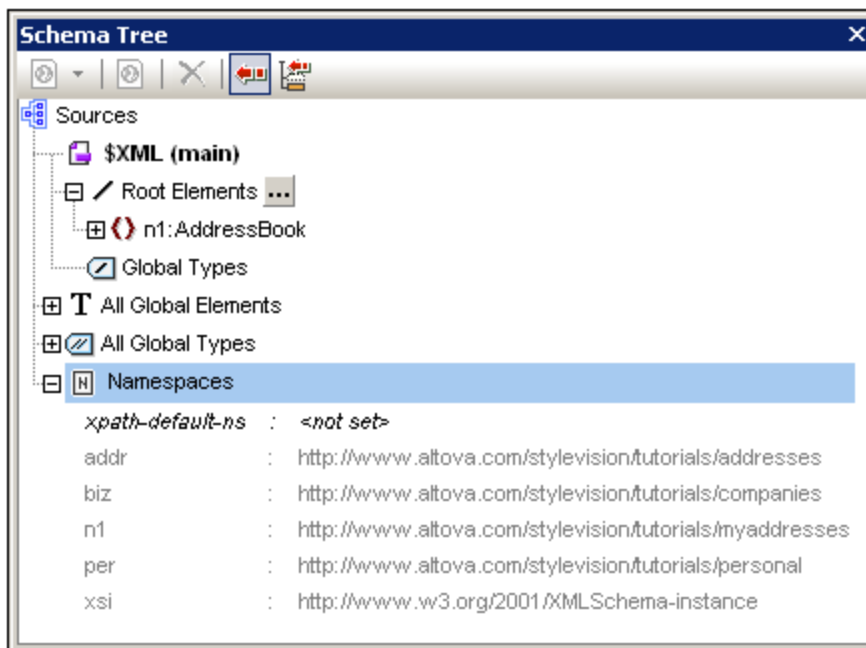
6.3.1 Available Module Objects

This section lists the objects in [added SPS modules](#)²³¹ that are available to the [referring SPS module](#)²³¹. 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](#)²³⁴. 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](#) ²³²
- [Global templates](#) ²³²
- [Design fragments](#) ²³³
- [Added modules](#) ²³³
- [Scripts](#) ²³⁴
- [CSS styles](#) ²³⁴
- [Page layouts](#) ²³⁴
- [Unavailable module objects](#) ²³⁴


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.

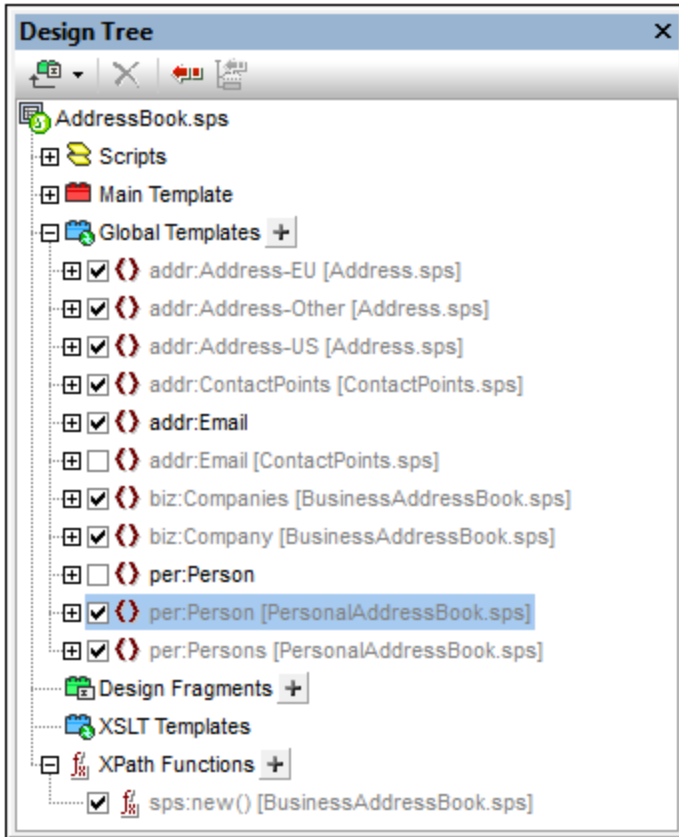


The screenshot above shows the various namespaces in an SPS, together with their prefixes, in the Schema Tree sidebar. These namespaces come from the source schema/s and cannot be edited.

Global templates

The [global templates](#) ²⁴⁴ of the added SPS module are available to the referring SPS module and are displayed in the [Design Tree sidebar](#) ⁴⁸ (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](#)²⁵⁵ in the added SPS module are available to the referring SPS and are displayed in the [Design Tree sidebar](#)⁴⁸ (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](#)⁴².

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](#)⁴⁶⁷. 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](#)³⁶⁵. The CSS files are also listed in the [Design Overview sidebar](#)⁴². 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](#)⁴⁸.

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 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.

6.3.2 Creating a Modular SPS

Creating a modular SPS consists of four broad parts:

1. Design and save the [SPS module to be added](#)²³⁴.
2. [Add the module](#)²³⁵ 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](#)²³⁷ 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](#)²³¹ can only be [global templates](#)²⁴⁴. 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](#)²³¹.

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](#) ²³⁴. 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 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:

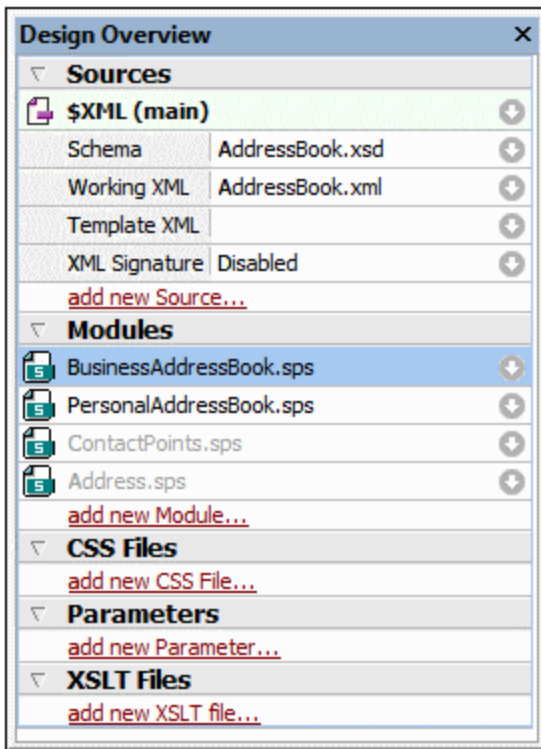
- The main source schema of the referring SPS. In this case, when the SPS is added, the added global templates will be part of the content model of the referring SPS's main schema. The output of these global templates in Authentic View is, therefore, editable.
- A schema which defines a content model 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 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](#) ²³² used since these determine the expanded names of nodes.

You could use a [Working XML File](#) ²⁸ 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](#) ²³⁴.

Adding the SPS module

To add a module to an SPS, in the [Design Overview](#) ⁴² (*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 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.

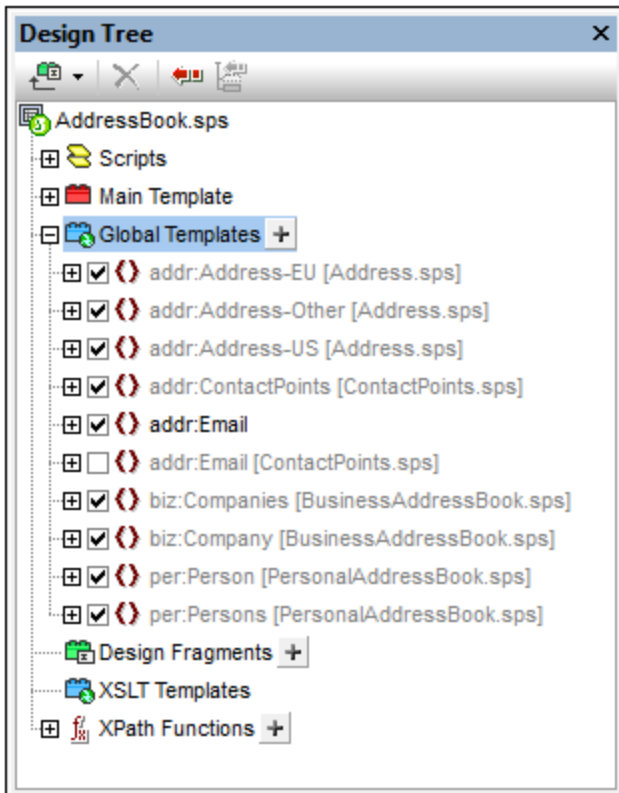
The module order is not significant for resolving conflicts among scripts, global templates, design fragments, and page layout items.

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](#)²³¹, 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](#)²³⁷.

Global templates

When you wish to use a [global template](#)²⁴⁴ 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](#)²⁵⁵.

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](#)⁴⁶⁹.

6.3.3 Example: An Address Book

The [\(My\) Documents folder](#)²⁹, C:\Documents and Settings\\My Documents\Altova\StyleVision2024\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

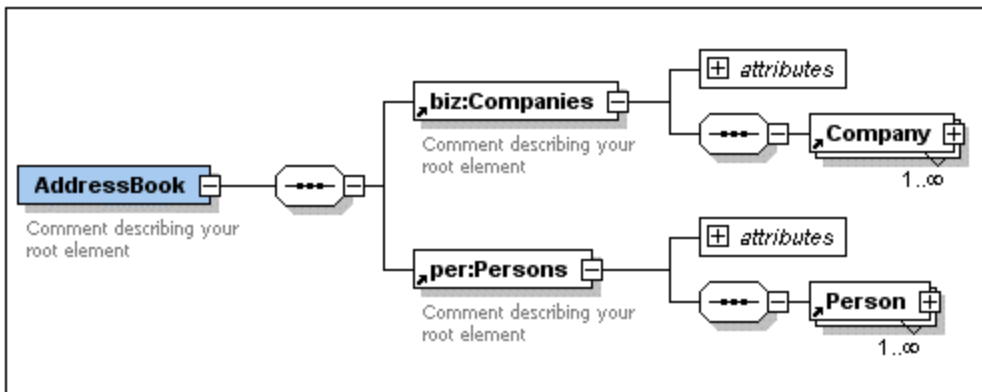
one of three content models, depending on whether the address is in the EU, US, or elsewhere. The output for all three content models is defined in a single SPS. However, they could have been defined in separate SPSs, which would have provided finer granularity. In this case, modularization would provide more flexibility as modules could be re-used more easily.

The description of this project is organized into the following parts:

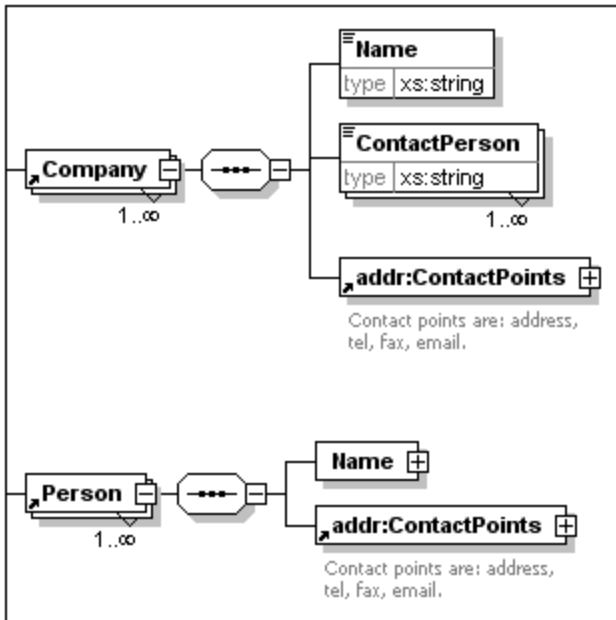
- [The schema files](#) ²³⁹
- [The XML data sources](#) ²⁴⁰
- [The SPS files](#) ²⁴¹

The schema files

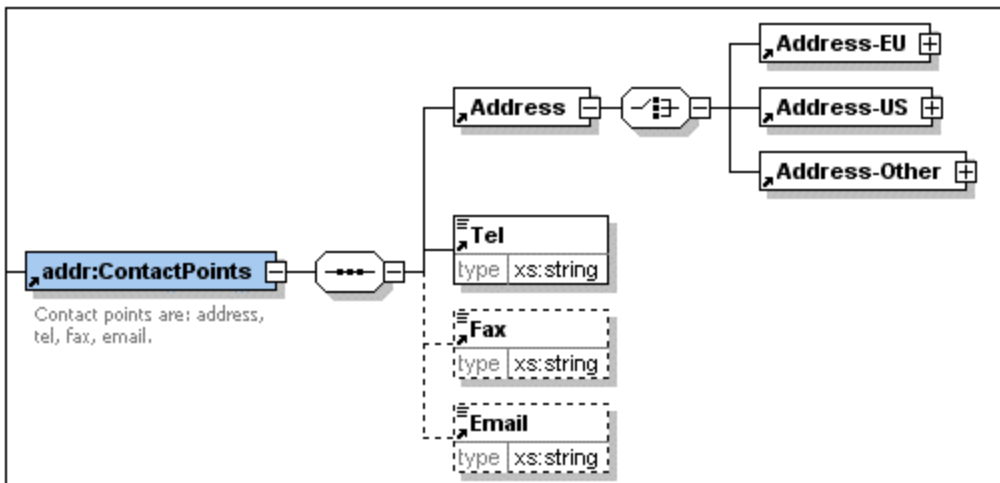
When creating schemas for modular SPSs, the most important thing to bear in mind is to create the elements that you wish to re-use as global elements. The schema for the address book is `AddressBook.xsd`. This schema has been constructed by importing the schemas for the business address book (`BusinessAddressBook.xsd`) 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](#) ²⁸ of the following SPS modules:

- `AddressBook.xml` => `AddressBook.sps`, `ContactPoints.sps`, `Address.sps`
- `BusinessAddressBook.xml` => `BusinessAddressBook.sps`
- `PersonalAddressBook.xml` => `PersonalAddressBook.sps`

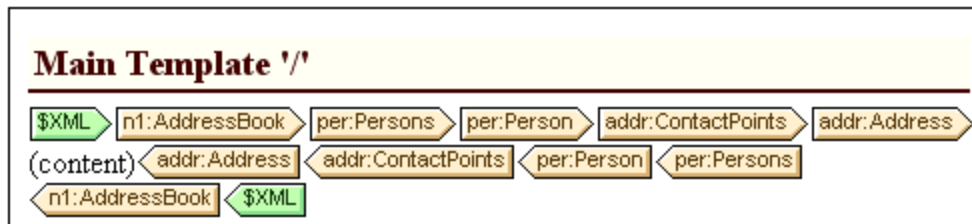
The SPS modules

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.

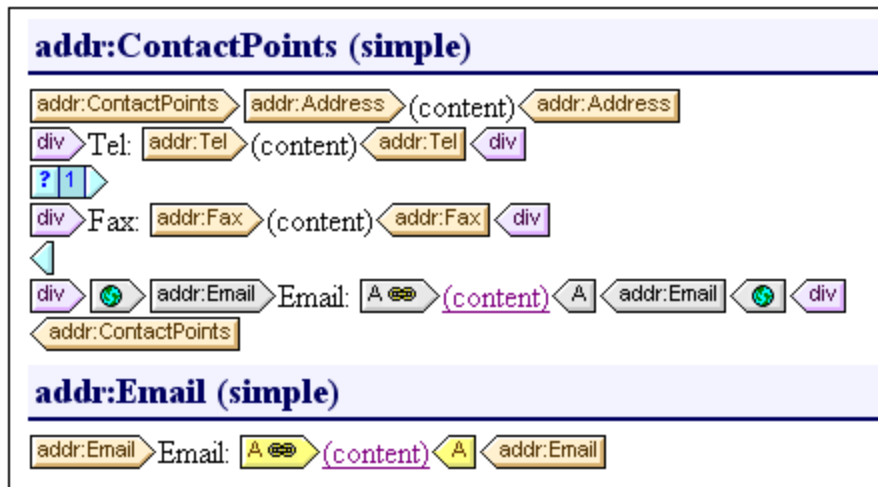


Because only the `Address` element is processed, the output previews show only the output of `Address`. When `Address.sps` is used as a module, the global templates are added and the main template is ignored.

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).



- 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

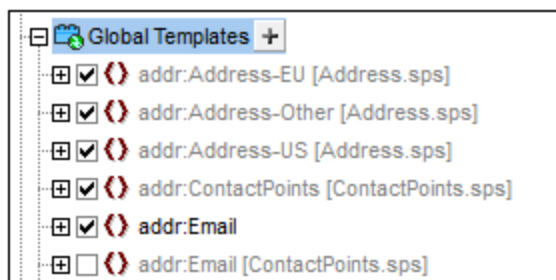
These SPSs each import one module, which in turn imports another. Note that the main template simply applies global templates.

- 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.

6.4 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](#)²⁴⁴ and, optionally, one or more [global templates](#)²⁴⁴. 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](#)²⁵² 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](#)³⁴⁹ 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](#)³⁹².

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](#)⁷¹¹, which are available as [toolbar icons](#)⁷¹¹. These display filters will help you optimize and switch between different displays of your SPS.

6.4.1 Main Template

The [main template](#)²⁷ 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](#)²⁷. 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](#)²⁷ 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](#)²⁷ of the schema. StyleVision offers the option of selecting multiple root elements ([document elements](#)²⁷). This means that within the main template, there can be [local templates](#)²⁷ 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.

6.4.2 Global Templates

A [global template](#)²⁷ can be defined for any node or type in the schema, or for a node specified in an XPath pattern.

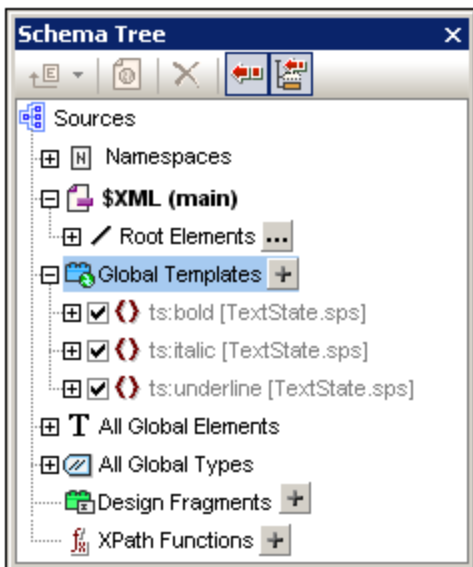
A global template specifies instructions for the selected node or type, and it is invoked by a call from the [main template](#) ²⁷, [design fragments](#) ²⁵⁵, or other global templates. The processing model is similar to that of declarative programming languages, in that a single template is defined 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](#) ²⁷ has been set to reference its global template (done by right-clicking the component in the design and selecting Make Global Template).
- When a [\(contents\)](#) ¹²⁵ or [\(rest-of-contents\)](#) ¹³⁰ is inserted within an element or type in a [local template](#) ²⁷, and the rest of the content of that element or type includes a node or type for which a [global template](#) ²⁷ 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](#) ²⁷ 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 [\(contents\)](#) ¹²⁵ or [\(rest-of-contents\)](#) ¹³⁰ 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

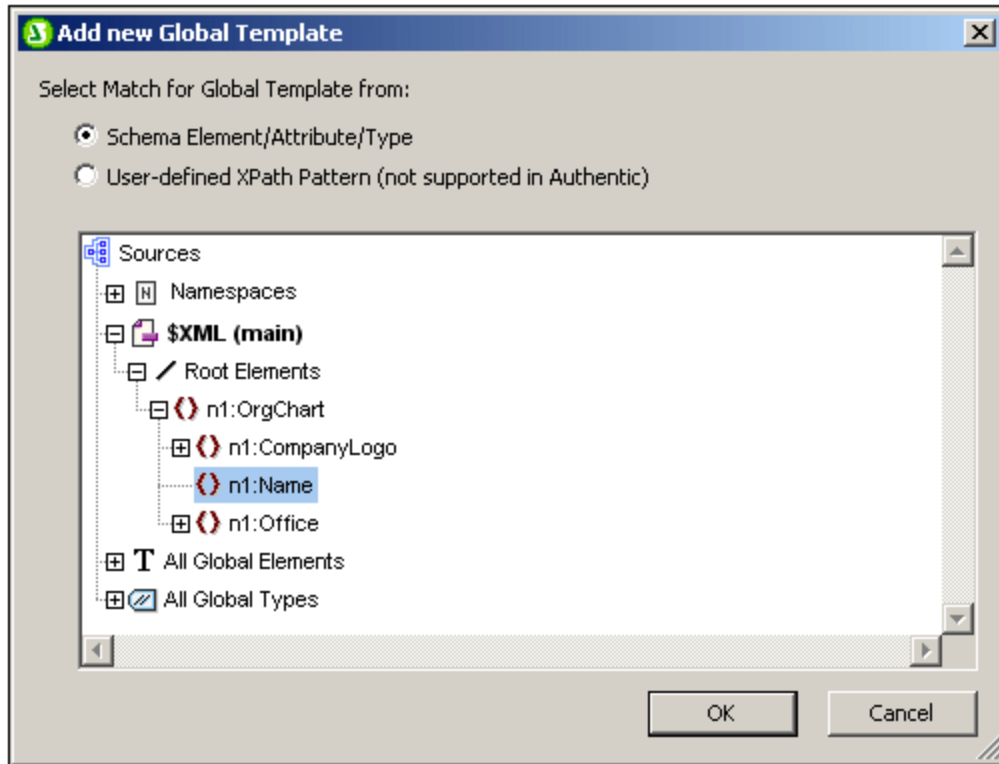
Global templates can be created for any node or type in the schema, or for a node specified in an XPath pattern., and are created from the Schema Tree sidebar (*screenshot below*).



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](#)²⁵³ in the context menu that appears. This pops up the [Edit XPath Expression dialog](#)⁶⁸⁵, 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](#)⁷³⁸). 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.

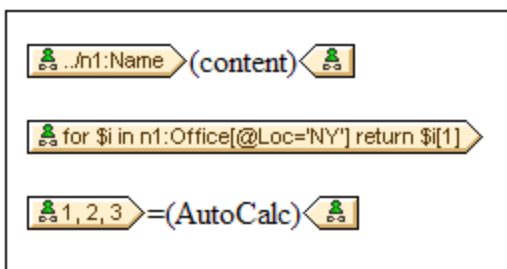
It is important to note that a simple global template will be automatically converted to a complex global template if a [predefined format](#)³⁵⁰ or newline is created **around** the element node for which the global template was created. This will result in the processing behaviour for complex global templates (described in the previous list item). To revert to the simple global template, the [predefined format](#)³⁵⁰ 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](#)⁴⁸), as the case may be. To avoid the automatic conversion from simple global template to complex global template, make sure that the [predefined format](#)³⁵⁰ 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](#)²³⁰, 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](#)²³⁰.

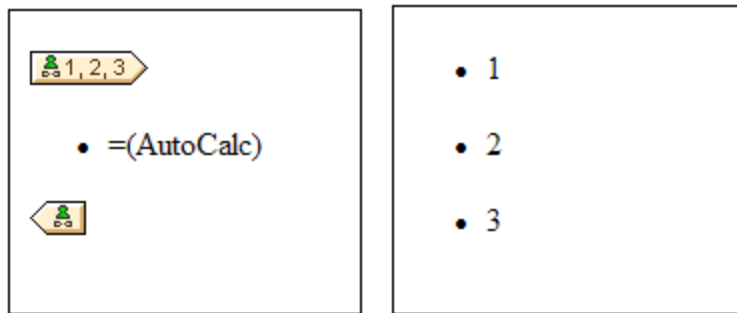
6.4.3 User-Defined Templates

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.

Brackets	Underlying XSLT Mechanism	Effect
----------	---------------------------	--------

No	<pre><xsl:for-each select="Org"> <xsl:for-each select="Office"> <xsl:for-each select="Dept"> ... </xsl:for-each> </xsl:for-each> </xsl:for-each></pre>	Each <code>Office</code> element has its own <code>Dept</code> population. So grouping and sorting can be done within each <code>Office</code> .
Yes	<pre><xsl:for-each select="/Org/Office/Dept"> ... </xsl:for-each></pre>	The <code>Dept</code> population extends over all <code>Office</code> elements and across <code>Org</code> .

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:

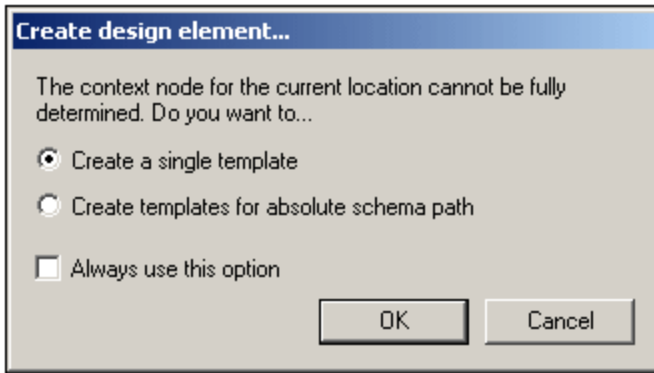
1. Click the **Insert User-Defined Template** icon in the Insert Design Elements toolbar and then click the design location where you wish to insert the template. Alternatively, right-click the design location where you wish to insert the template and, from the context menu that appears, select the **Insert User-Defined Template** command.
2. In the [Edit XPath Expression](#)⁶⁸⁵ 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](#)²⁵².

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

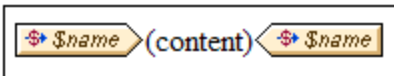
If a node from the schema tree is added to a User-Defined Template, the context for the new node will not be known if the User-Defined Template has been created for a node or sequence that cannot be placed in the context of the schema source of the SPS. You will therefore be prompted (*screenshot below*) 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](#)⁸³⁹.

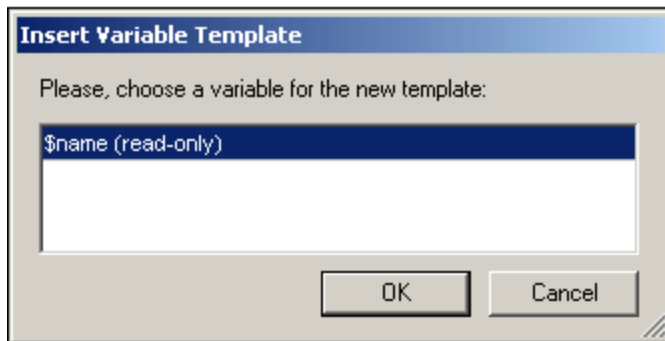
6.4.4 Variable Templates

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:

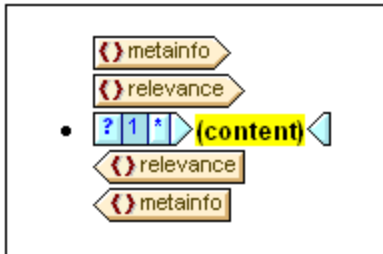
1. Place the cursor in the design at the location where the template is to be inserted.
2. Click the **Insert | Variable Template** command. This pops up the Insert Variable Template dialog (*screenshot below*).



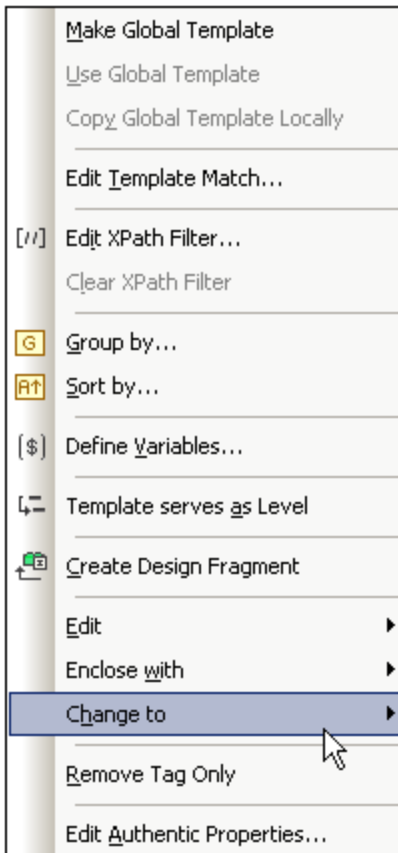
3. The dialog contains a list of all the [user-declared parameters and variables](#)³⁰² defined in the SPS. Select the variable for which you wish to add a variable template.
4. Click **OK** to finish.

6.4.5 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](#)⁶⁸¹.



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](#) ²⁵³
- [Template match](#) ²⁵³
- [XPath filters](#) ²⁵³
- [Group by, Sort by, Define variables, Template serves as level](#) ²⁵⁴
- [Create Design Fragment](#) ²⁵⁴
- [Remove Tag Only](#) ²⁵⁵
- [Edit, Enclose with, Change to](#) ²⁵⁵
- [Authentic properties](#) ²⁵⁵

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](#) ²⁴⁴.

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](#) ⁶⁸⁵, 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](#) ¹³⁷.

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](#) ²⁴⁴.

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.



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](#)²⁸⁸.
- 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](#)²⁹⁶.
- 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](#)³⁰⁷.
- 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](#)³¹².

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](#)¹²⁵, [special paragraph](#)¹²⁷, [Bullets and Numbering](#)¹⁶³, [Hyperlink](#)³³⁹, [Condition](#)²⁸⁰, [TOC Bookmark and Level](#)³¹².
- *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](#)¹⁹⁷.

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](#)⁴³³.


6.4.6 Design Fragments

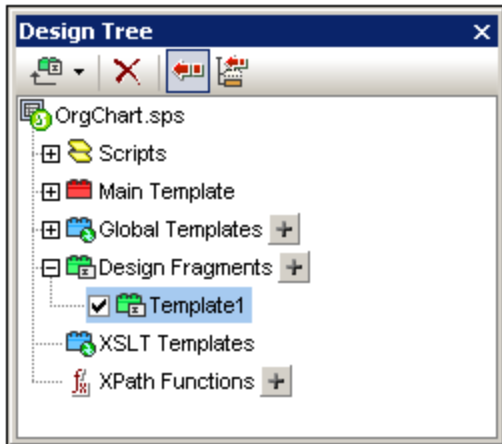
Design Fragments are useful for creating parts that can be re-used at different locations in the document, similar to the way functions are re-used. The usage mechanism is as follows:

1. [Create the Design Fragment in the design](#)²⁵⁵
2. [Fill out the contents of the Design Fragment](#)²⁵⁶
3. [Insert the Design Fragment at a location in a template](#)²⁵⁷.

Creating a Design Fragment

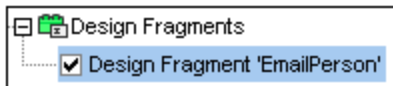
To create a Design Fragment do the following:

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.*)



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](#)⁷¹¹ icons in StyleVision's [Design Filter](#)⁷¹¹ 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.



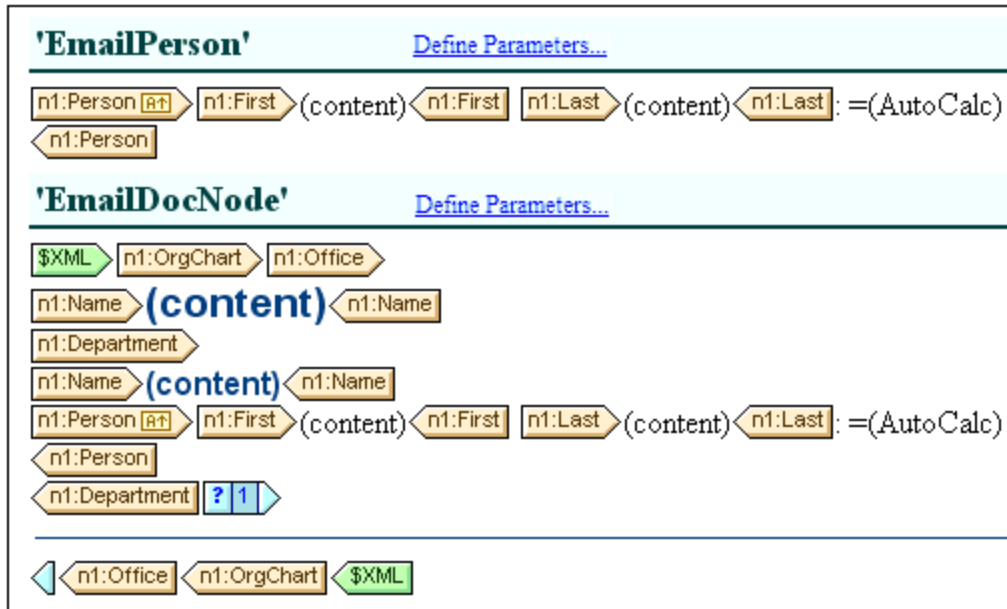
3. In the design, create the contents of the Design Fragment template. How to do this is described in the next section.

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 point in the design. The Design Fragment template is also appended to the existing Design Fragment templates at the bottom of the design and added to the Design Tree and Schema Tree. Creating a Design Fragment in this way also applies it directly at the point where it was created, there is no need to [insert it from the Design Tree or Schema Tree](#)²⁵⁷.

Creating the contents of a Design Fragment

The contents of the Design Fragment template are created [as for any other template](#)¹²⁴. 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](#)²⁹, `C:\Documents and Settings\\My Documents\Altova\StyleVision2024\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](#)³⁰⁴ 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](#)²⁵⁶ 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](#)⁷³⁸). 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](#)²³⁰, 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](#)²³⁰.

Example file

For an example SPS, go to the [\(My\) Documents folder](#)²⁹, C:\Documents and Settings\\My Documents\Altova\StyleVision2024\StyleVisionExamples\Tutorial\DesignFragments.

6.5 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.

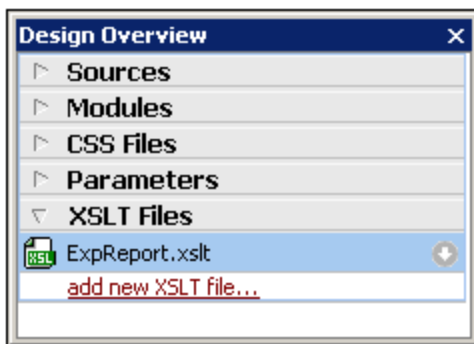
Note the following points:

- Imported XSLT templates cannot be modified in StyleVision.
- XSLT templates are not supported in Authentic View.

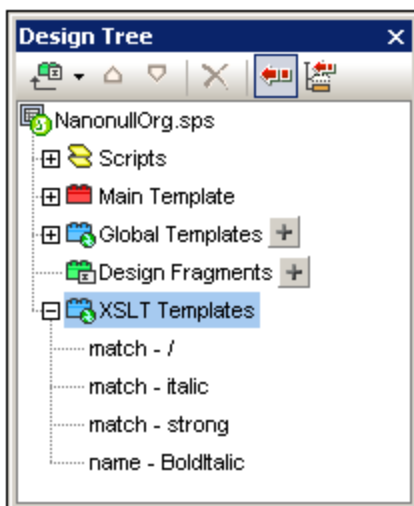
Importing the XSLT file

To import an XSLT File, do the following:

1. In the Design Overview sidebar (*screenshot below*), click the **Add New XSLT File** link.



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.



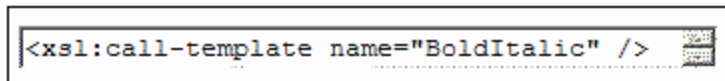
There are two types of imported XSLT templates: (i) match templates (indicated by *Match*), and (ii) named templates (indicated by *Name*). In the Design Tree, these two types are listed with (i) the value of the `select` attribute of match templates, and (ii) by the value of the `name` attribute of 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

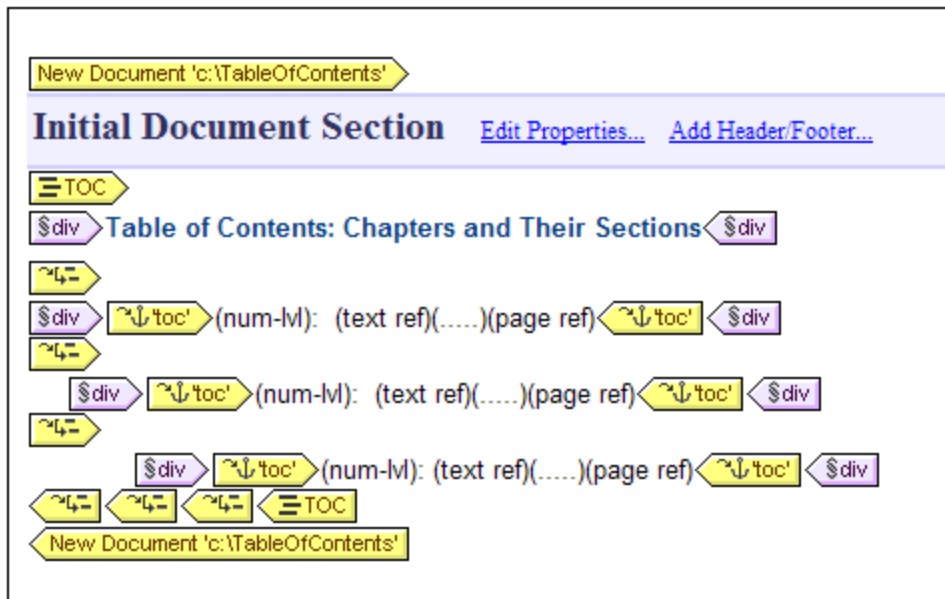
A named template can be dragged from the Design Tree to any location in the design. At this location, it will be created as an `xsl:call-template` element (*screenshot below*) that calls the named template.

A screenshot of a code editor showing an XSLT call-template element. The code is: `<xsl:call-template name="BoldItalic" />`. The code is highlighted in a light blue color. To the right of the code, there is a small icon of a document with a magnifying glass, indicating a search or help function.

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.

6.6 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](#)²⁶³.

The multiple output-document feature is available in HTML, RTF, and Text formats. 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:

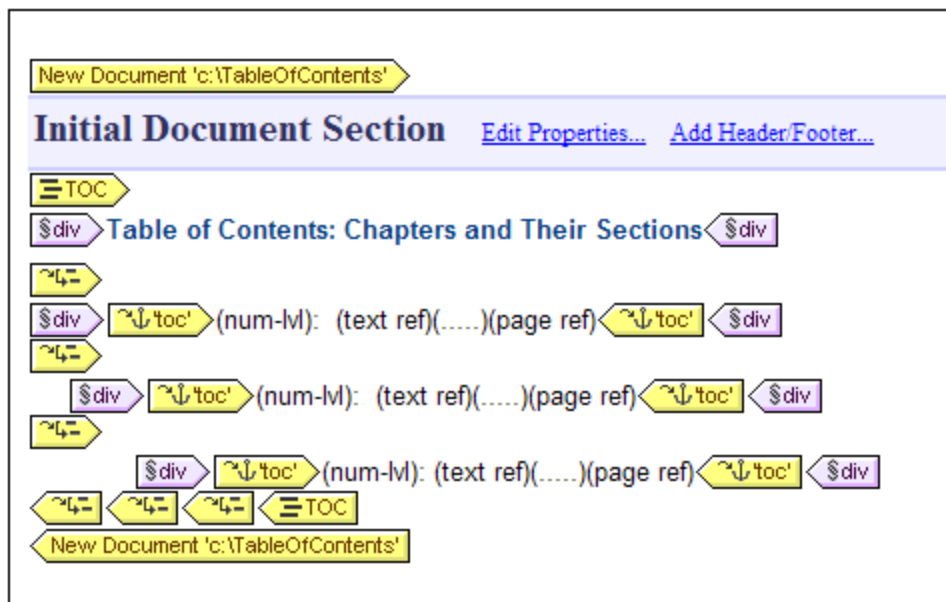
- [Inserting a New Document Template](#)²⁶²
- [New Document Templates and Design Structure](#)²⁶³
- [URLs of New Document Templates](#)²⁶³
- [Preview and Output Document Files](#)²⁶⁵
- [Document Properties and Styles](#)²⁶⁸

6.6.1 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](#)²⁶³.
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](#)³⁹³.

6.6.2 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](#)²⁶³ as well as the [input \(source XML document\) structure](#)²⁶³ 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.

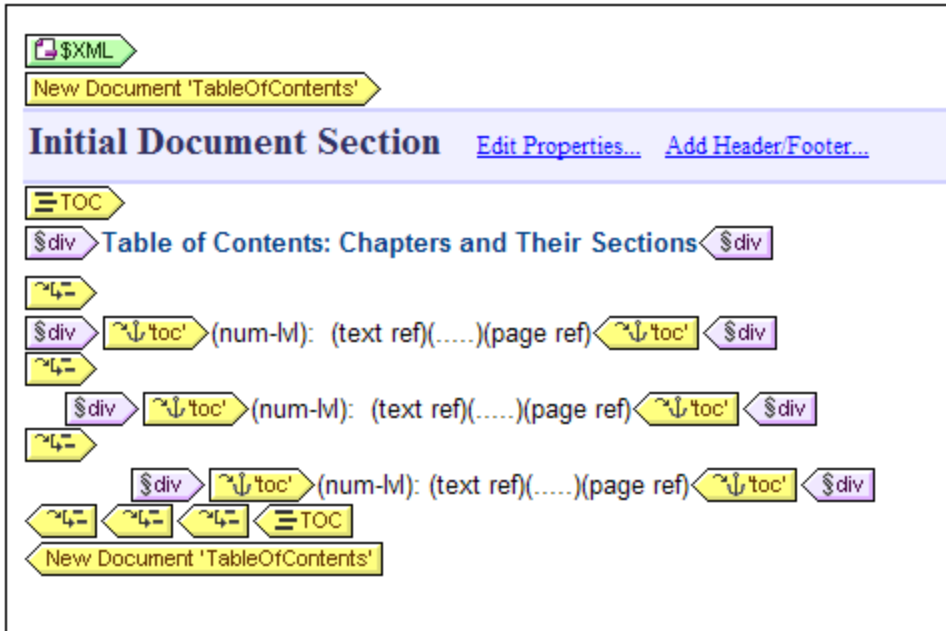
6.6.3 URLs of New Document Templates

In this section we describe how the [URLs of New Document templates relate to design structure](#)²⁶³, how [URLs are edited](#)²⁶⁵, and [how multiple output-documents can be linked](#)²⁶⁵ among each other.

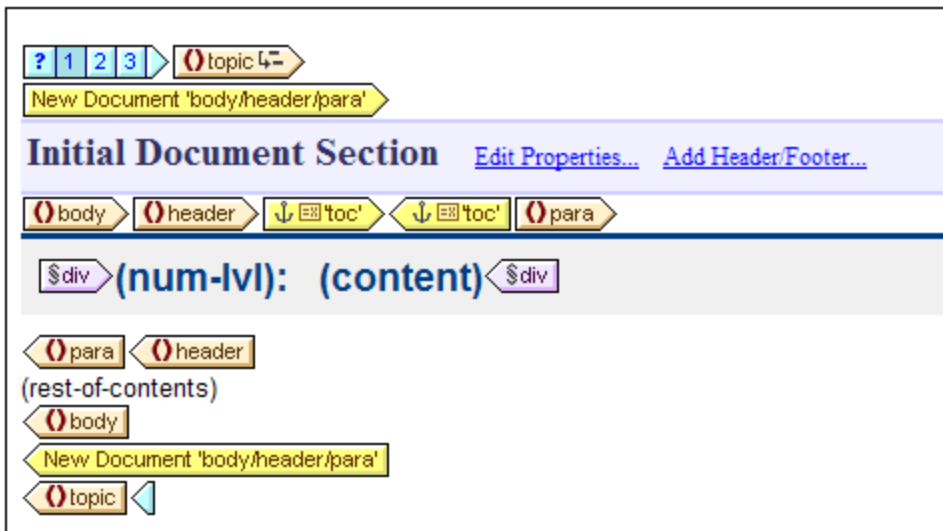
URLs of New Document templates

If the New Document template is processed only once (see [preceding section](#)²⁶³), 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](#) ²⁶⁵ for details). Alternatively, if the URL contained a path, the output-document will be saved to the location specified in the path.

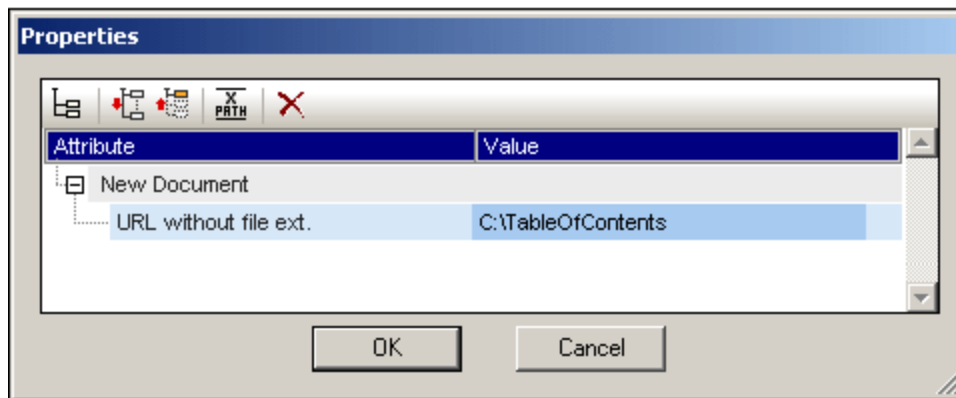


If, on the other hand, a New Document template will be processed multiple times to generate multiple output-documents (see [preceding section](#) ²⁶³), then the template's *URL* property must be a dynamic URL that is selected with an XPath expression. In the screenshot below, the URL of the New Document 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: `DocumentX` (where `X` is an integer). If you wish to edit the URL, right-click the New Document template and select the command **Edit URL**. This pops up the Properties dialog (*screenshot below*), in which you can edit the Value field of the *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 `concat()` function of XPath. For example: `concat('C:\MyOutput\', body\header\para)`. This example expression will generate the URL string: `C:\MyOutput\filename`. The appropriate file extension will be generated automatically according to the output format.

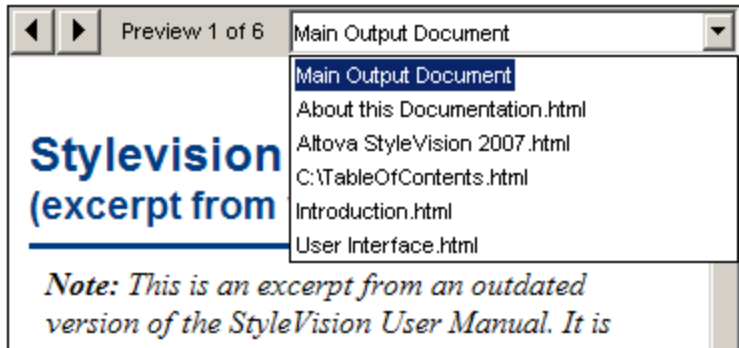
Linking the documents

Multiple output-documents can be linked to one another using [bookmarks and hyperlinks](#)³³⁹. A [bookmark](#)³³⁹ can be placed at the head of a New Document Template or anywhere within the New Document Template. [Hyperlinks](#)³⁴¹ 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 \(TOC\)](#)³¹² 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.

6.6.4 Preview Files and Output Document Files

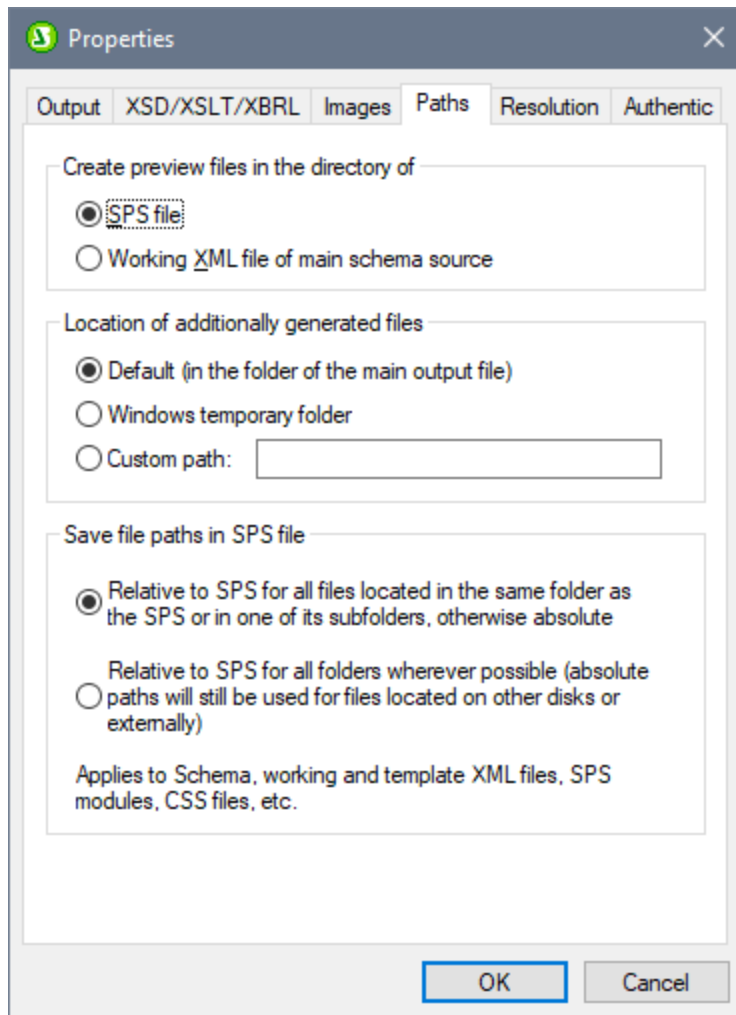
The output previews of a design document show each of the multiple output-documents that have been specified in the design as separate documents (*see screenshot below*).



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**.

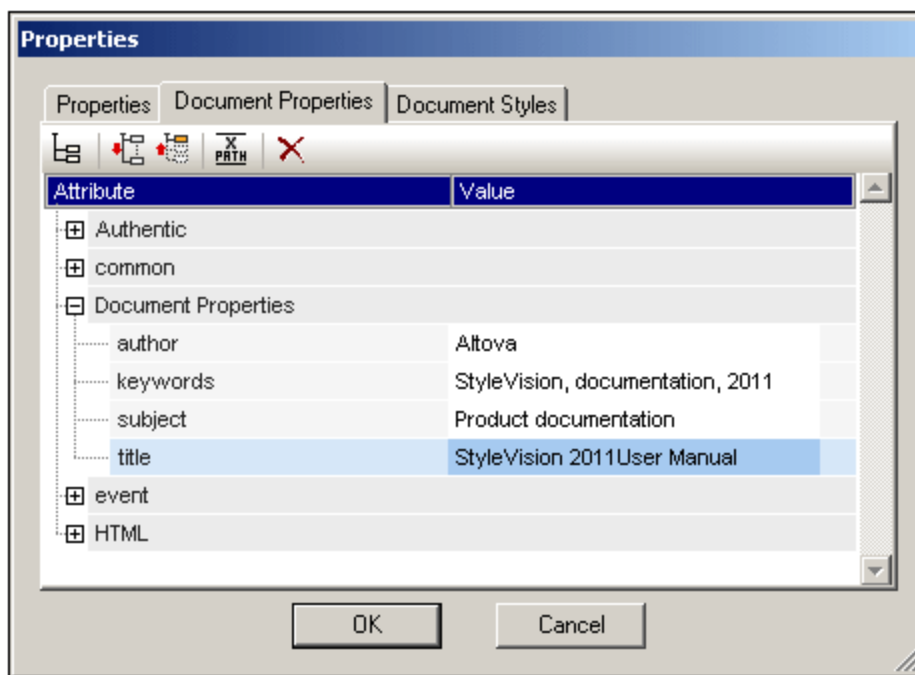
The location of all output-document files as well as other additionally generated files, such as image files and chart-image files, will be displayed in a pop-up window for your information.

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.

6.6.5 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](#) ³⁷³.

7 Advanced Features

How to create the basic content and structure of the SPS design is described in the sections, [SPS File Content](#)¹²⁴ and [SPS File Structure](#)²⁰⁰. 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](#)²⁷⁰. 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](#)²⁸⁰. 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](#)²⁸⁸. Processing can be defined for a group of elements that are selected with an XPath expression.
- [Sorting](#)²⁹⁶. A set of XML elements can be sorted on multiple sort-keys.
- [Parameters and Variables](#)³⁰². 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](#)³¹². 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](#)³³⁴ (repeating nodes in the document can be numbered automatically and the numbers formatted); (ii) [Text References](#)³³⁸ (text in the document can be marked for referencing and then referenced from elsewhere in the document); and (iii) [Bookmarks and Hyperlinks](#)³³⁹ (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.

7.1 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:

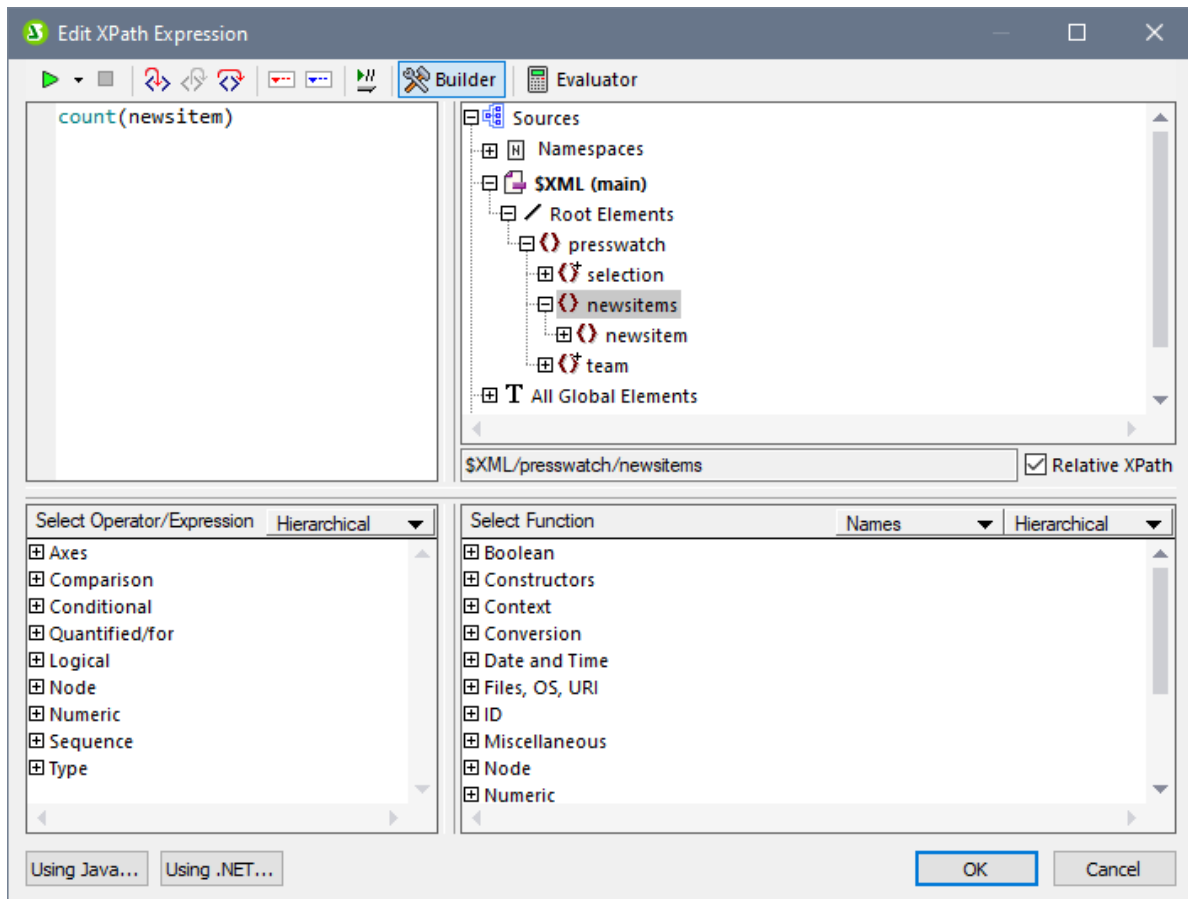
- Inserting calculations involving operations on dynamic data values. For example, you can count the number of `Employee` elements in an `Office` element (with `count(Employee)`), or sum the values of all `Price` elements in each `Invoice` element (with `sum(Price)`), or join the `FirstName` and `LastName` 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.

7.1.1 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*).

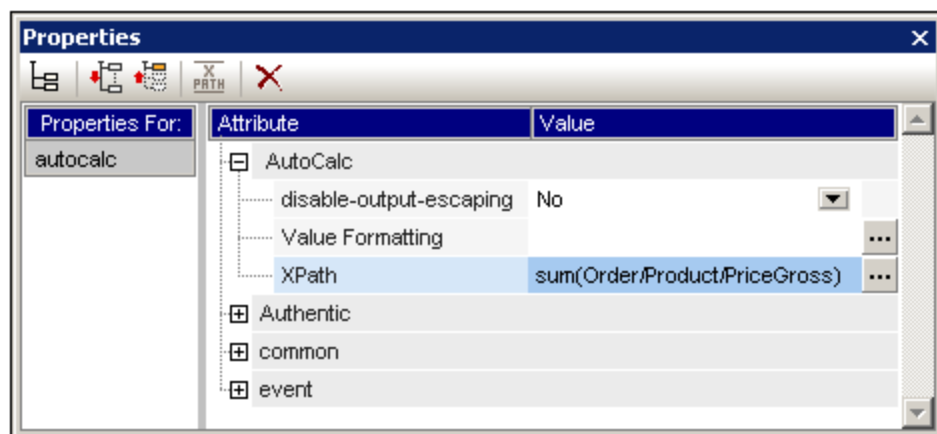


2. In the *Expression* pane, enter the XPath expression for the Auto-Calculation via the keyboard. Alternatively, enter the expression by double-clicking nodes, operators, and/or functions in the respective panes of the dialog. It is important to be aware of the context node at the insertion point; the context node is highlighted in the schema source tree (in the screenshot above the context node, *newstitems*, 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](#)⁶⁸⁵.
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](#)²⁷².

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.

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](#)⁶⁸⁵ (*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](#)³⁵⁴ 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

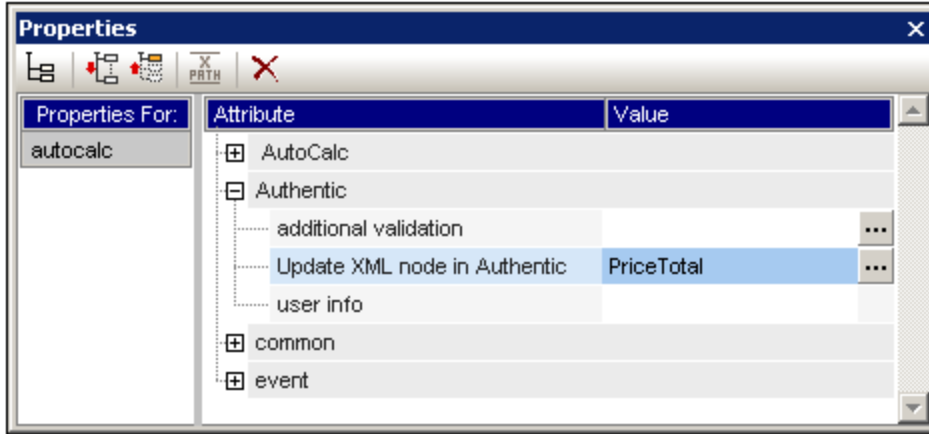
Note the following points:

- 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.

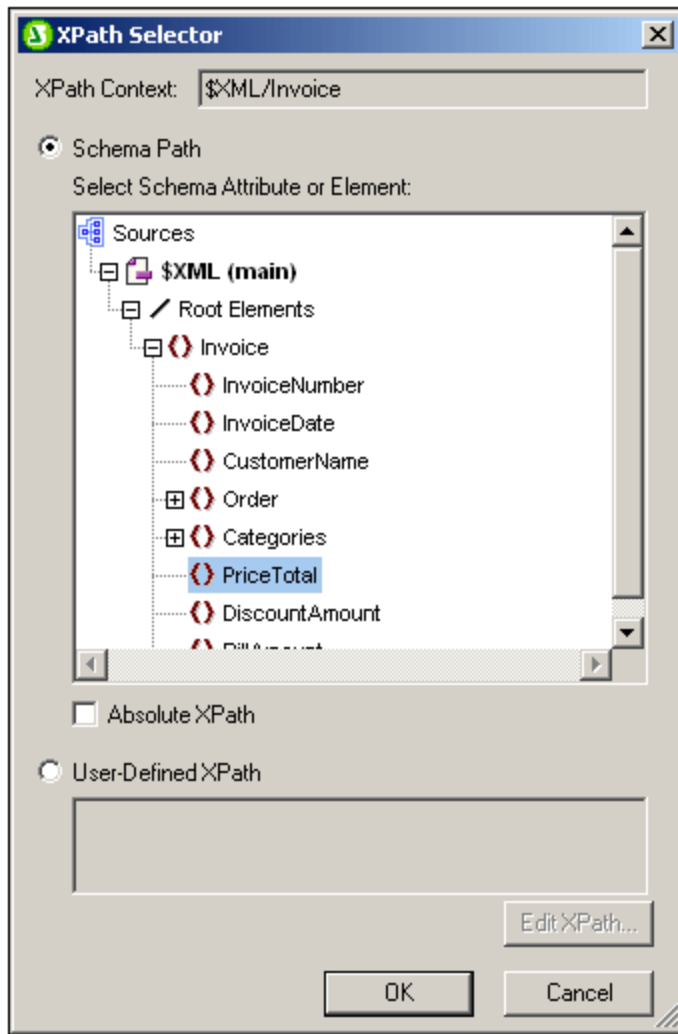
7.1.2 Updating Nodes with Auto-Calculations

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](#) ²⁷⁰.
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!

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 `Price*1.2`, and the Auto-Calculation is set to update the `VATPrice` node, then the `Price` element must be modified in Authentic View in order for the `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).

7.1.3 Auto-Calculations 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](#)²⁷⁶.

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](#)²⁷⁶.

7.1.4 Example: An Invoice

The `Invoice.sps` example in the [\(My\) Documents folder](#)²⁹, `C:\Documents and Settings\\My Documents\Altova\StyleVision2024\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*).

Product 1:	Learning XMLSpy
Net price:	€ 35.00
Category:	Book
VAT:	10%
Price including VAT:	€ 38.5
<hr/>	
Product 2:	Scooby Doo's Greatest Hits

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

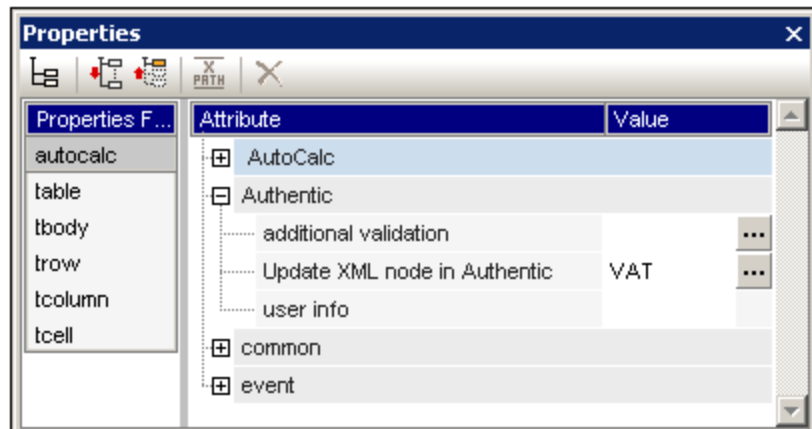
In the Invoice example, the user selects the category of product (`Book`, `CD`, `DVD`, or `Electronics`) via a combo box. This selection is entered in the `//Product/Category` node in the XML document. An Auto-Calculation then uses this value to reference a "lookup table" in the XML document and identify 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:

$$\text{Gross Price} = \text{Net Price} + \text{VAT-amount}$$

$$\text{Since } \text{VAT-amount} = \text{Net Price} * \text{VAT-percentage} \text{ div } 100$$

$$\text{Gross Price} = \text{Net Price} + (\text{Net Price} * \text{VAT-percentage} \text{ div } 100)$$

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:

$$\text{PriceNet} + (\text{PriceNet} * (\text{VAT} \text{ div } 100))$$

The XPath expression can be [viewed and edited in the Properties window](#)²⁷⁰. 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.

Product 6:	A Short History of the American Century
Net price:	€ 20.00
Category:	DVD
VAT:	15%
Price including VAT:	€ 23
<hr/>	
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	<u>Volume Discounts</u>
Less Volume Discount:	€ 35.89	Over € 100 = 05%
Bill Amount:	€ 323.01	Over € 300 = 10%
		Over € 600 = 12%

You can see these two additional Auto-Calculations in the file `InvoiceWithDiscounts.sps`, which is in the [\(My\) Documents folder](#)²⁹, `C:\Documents and Settings\\My Documents\Altova\StyleVision2024\StyleVisionExamples\Tutorial\Auto-Calculations` folder.

7.2 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](#)²⁸⁰, which describes how to create a condition and its branches.
- [Editing Conditions](#)²⁸³, about how to edit the XPath expressions of condition branches after they have been created.
- [Conditions for Specific Outputs](#)²⁸⁴, which shows how conditions are used to produce different output for different output formats.
- [Conditions and Auto-Calculations](#)²⁸⁵, explains usage issues when conditions and Auto-Calculations are used in combination.

7.2.1 Setting Up the Conditions

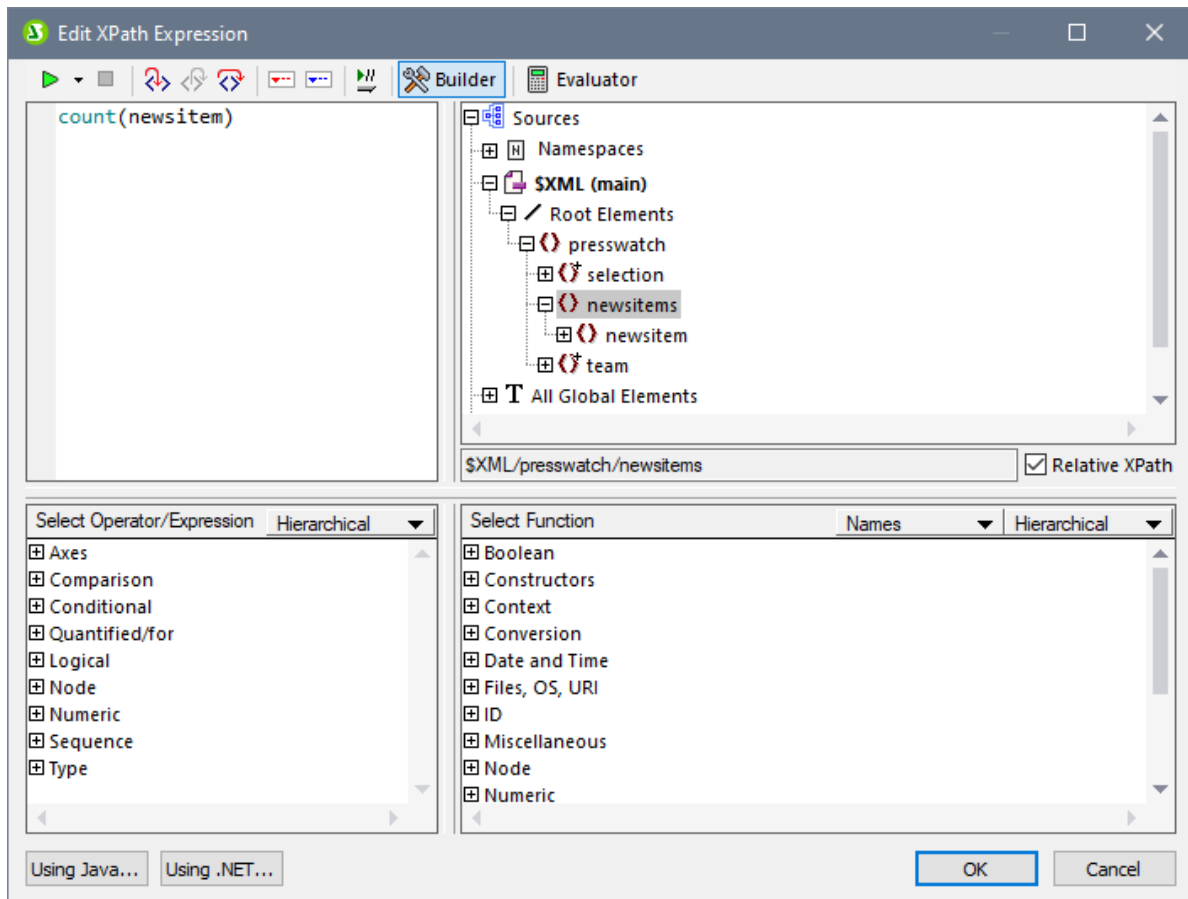
Setting up the condition consists of the following steps:

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.

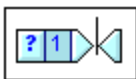
Creating the condition with its first branch

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](#)⁶⁸⁵ pops up (*screenshot below*).



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.

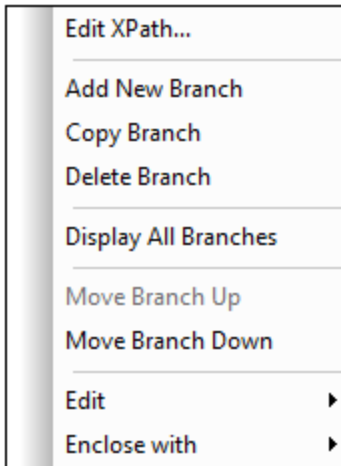


To select the entire condition, click the cell with the question mark. To select the first branch, click the cell with the number one.

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**).



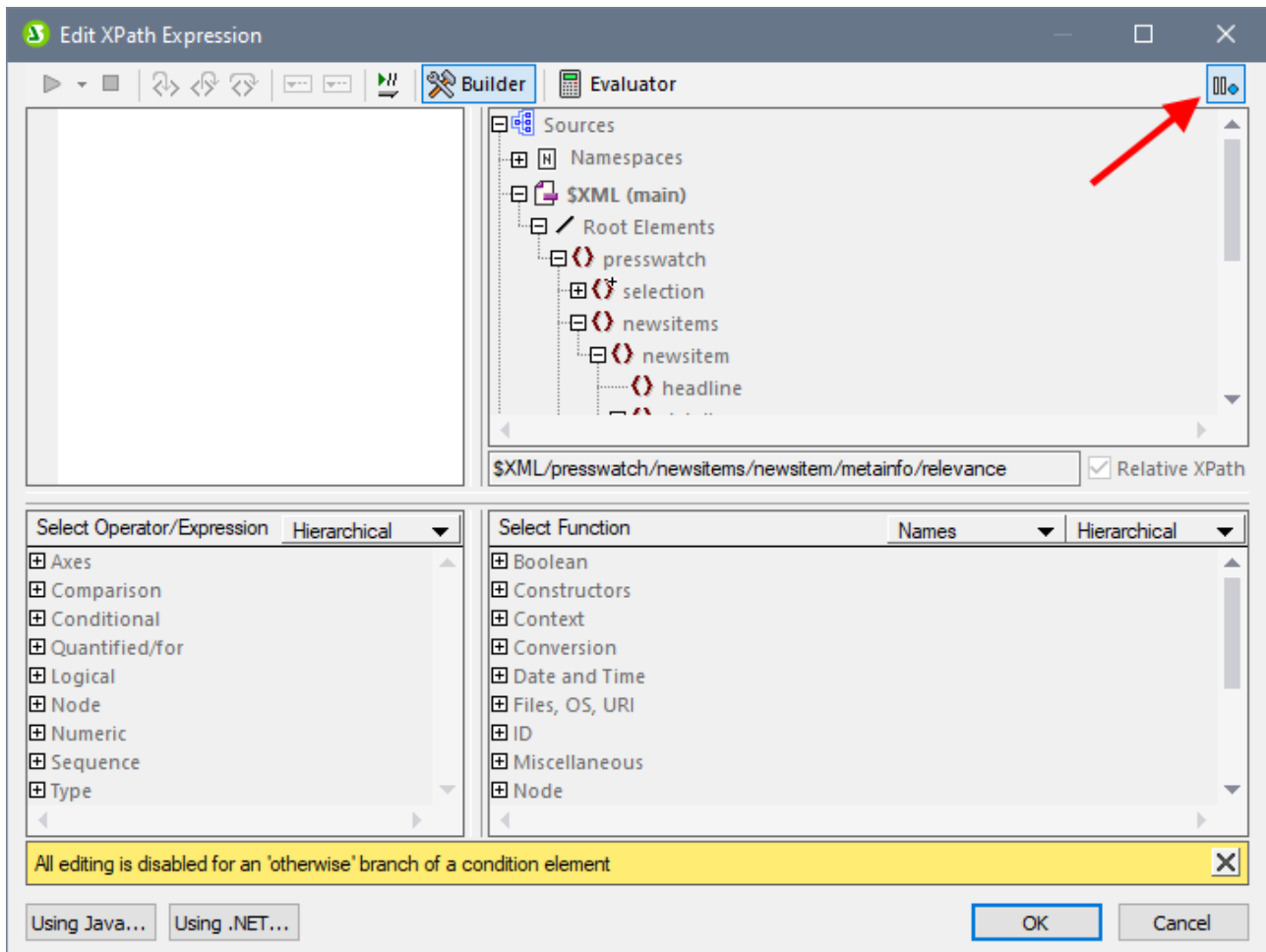
To create a new branch, right-click any branch of the condition and select **Add New Branch** from the context menu. The [Edit XPath Expression dialog](#)⁶⁸⁵ will pop up. After entering an XPath expression and clicking **OK**, a new empty branch is added to the condition. This is indicated in the design 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](#)⁶⁸⁵ 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](#)⁶⁸⁵ 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**.

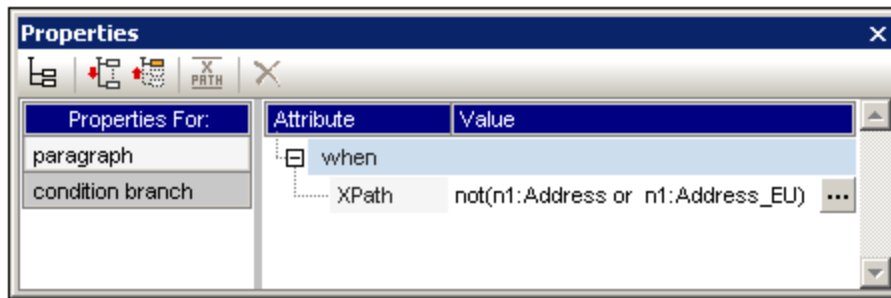
Deleting a branch



To delete a branch, select the branch to be deleted, then right-click and select **Delete Branch**.

7.2.2 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.


7.2.3 Output-Based Conditions

Individual components in the document design can be processed differently for StyleVision's different output formats (Authentic View, RTF, Text 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 four branches, each having the same content (the selected component). Each branch represents a single output (Authentic View, RTF, Text, 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 `$SV_OutputFormat = 'format'`, where `format` is one of the values: `Authentic`, `RTF`, `Text`, or `HTML`. You can edit the XPath expression of a condition branch (in the Properties sidebar, in the Condition Branch entry, [click the Edit button](#)²⁸³). For example, you could combine the Authentic View and HTML output formats in one condition branch (using the XPath expression: `$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](#)²⁸⁰ and [Editing Conditions](#)²⁸³.

Using the `$SV_OutputFormat` parameter

In the XSLT file generated for each output, `$SV_OutputFormat` is created as a global parameter and assigned the value appropriate to that output format (that is, `Authentic`, `RTF`, `Text`, or `HTML`). This parameter can be overridden by passing another value for it to the processor at runtime. This could be useful, if, for example, you wish to create two alternative HTML output options, one of which will be selected at runtime. You could then create condition branches `$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.

7.2.4 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.


7.3 Conditional Presence

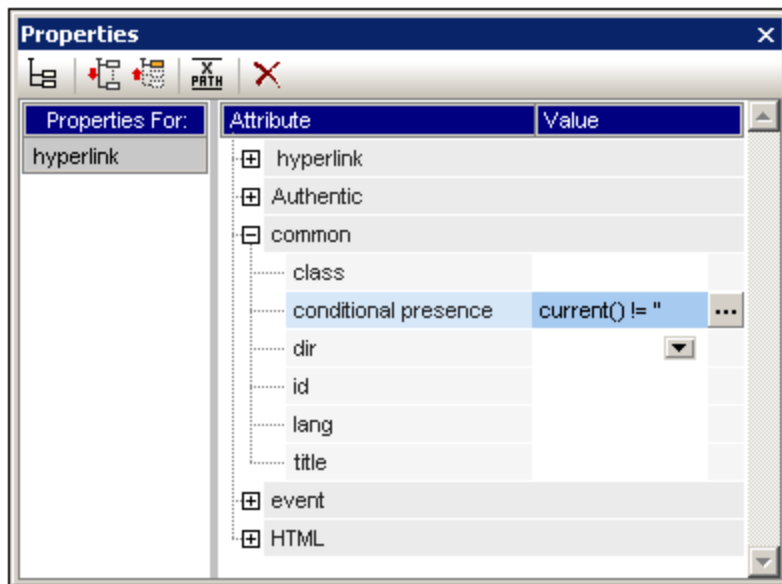
The Conditional Presence feature enables you to create certain design components if a specified condition is fulfilled. The design components that can be created conditionally are:

- [Hyperlinks](#) ³⁴¹
- [Bookmarks](#) ³³⁹
- [New Documents](#) ²⁶²
- [User-Defined Elements](#) ¹⁴⁰
- [TOC Levels](#) ³¹⁵

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](#) ⁶⁸⁵ 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.) 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).

7.4 Grouping

The grouping functionality is available in **XSLT 2.0 and 3.0** SPSs and for HTML, Text, and RTF 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.

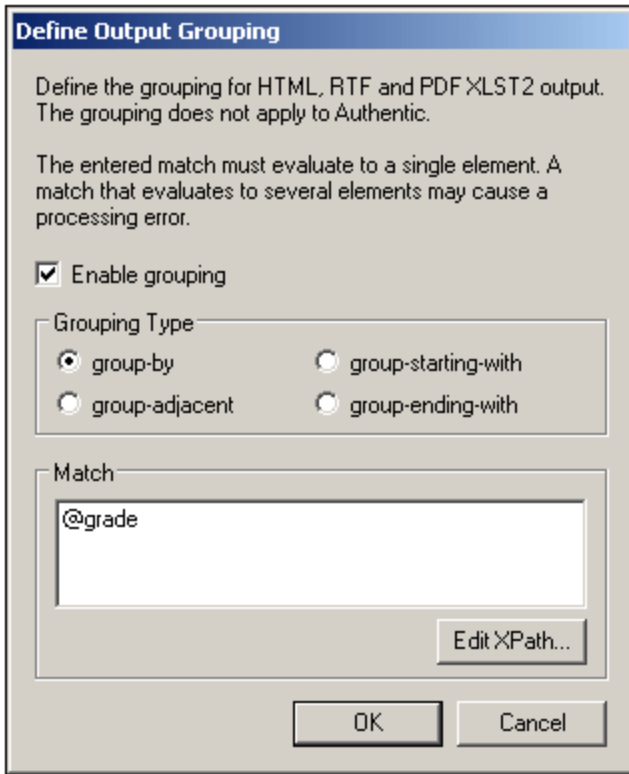
Grouping criteria

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:

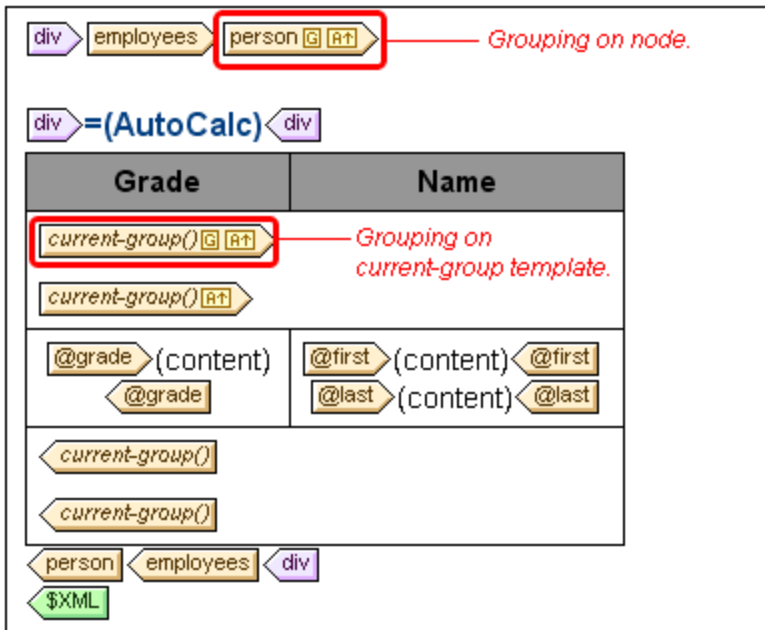
- **group-by**, which groups items on the basis of an XPath-defined key. For example, `car` elements can be grouped on the basis of their `brand` attributes. The grouping is set on the `car` element, and an XPath expression selects the `brand` attribute.
- **group-adjacent** uses a combination of grouping-key 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](#)²⁹⁶. How to use this dialog is described in the section [Sorting](#)²⁹⁶. 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](#)²⁴⁸ 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.

7.4.1 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](#)²⁹, `C:\Documents and Settings\\My Documents\Altova\StyleVision2024\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:

```
<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

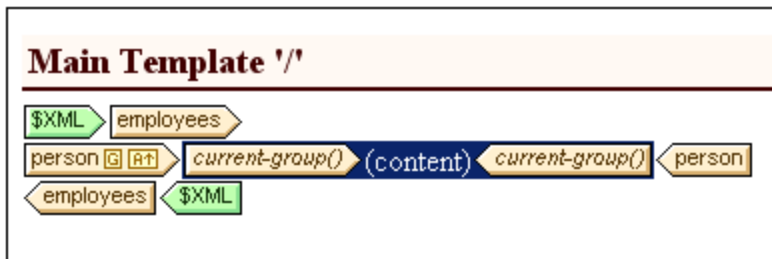
The strategy for creating the groups is as follows. The grouping is created on the `person` element with the `department` attribute being the grouping-key. This causes the `person` elements to be ordered in groups based on the value of the `department` attribute. (If sorting is specified, then the department 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.

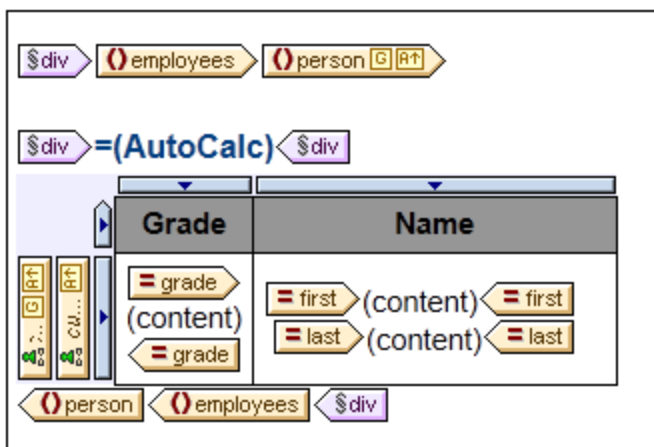
Creating the SPS

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 `@grade` and the second column contains the `@first` and `@last` nodes (*see screenshot below*).
7. Within the current group, which is grouped by department, we wish to group by grade. So on the `current-group()` template, create a grouping for the `grade` attribute. Confirm the default sorting. A new `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 `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 `current-grouping-key()` function of XPath 2.0/3.0.
11. Repeat the entire process, to create similar output, but 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 **Group by** or **Sort by** from the context menu. This pops up the respective dialog, in which the settings can be viewed or modified.

7.4.2 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](#)²⁹, `C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2024\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:

```
<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**Group A**

Team	P	W	D	L	F	A	Pts
Brazil	3	2	0	1	4	3	6
Italy	3	1	2	0	5	4	5
Germany	3	0	2	1	4	5	2
Holland	3	0	2	1	4	5	2

Strategy

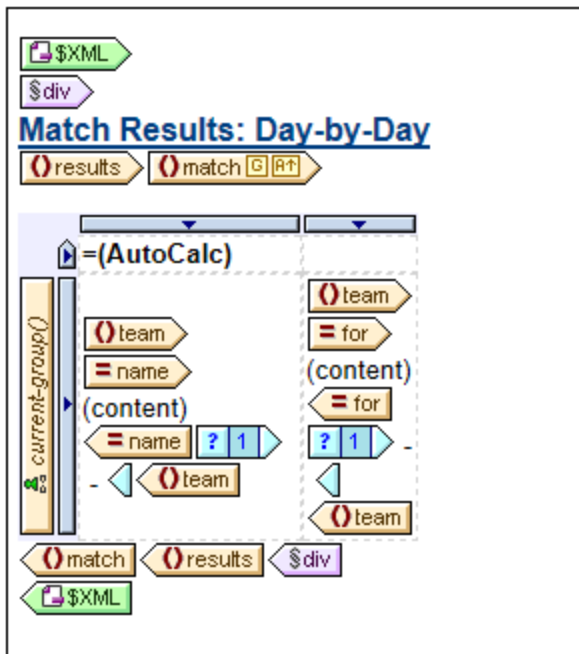
For the two sections containing the match results, we group matches by date and tournament-group. For members of each group (date and tournament group), we create borderless tables (for alignment purposes). So matches played on a single date will be in a separate table, and all the match results of a 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.

7.5 Sorting

The sorting functionality is available for HTML, Text, and RTF 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, Text, and RTF output.

User-defined templates

[User-defined templates](#)²⁴⁸ 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.

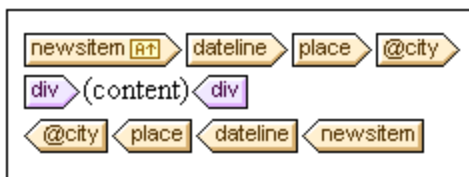
In this section

- The [sorting mechanism](#)²⁹⁶ is described.
- An [example](#)²⁹⁶ demonstrates how sorting is used.

7.5.1 The Sorting Mechanism

Setting up a schema element node for sorting consists of two steps:

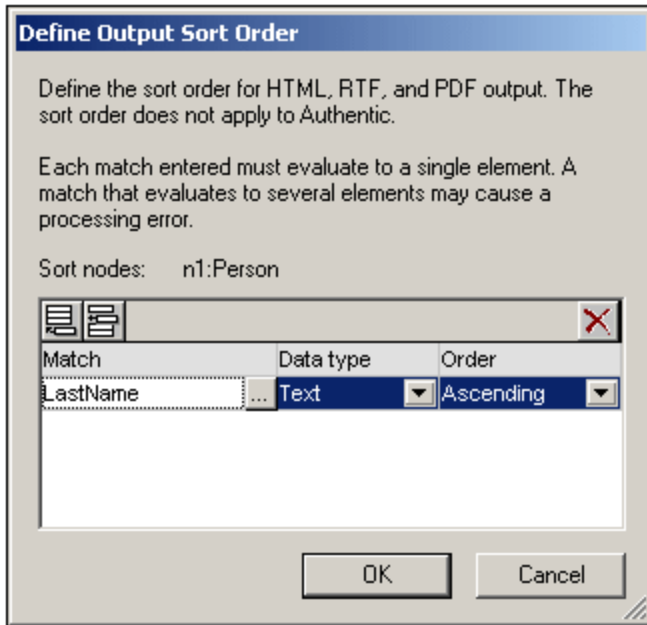
1. In Design View, select the schema element node that is to be sorted. Note that it is the instances of **this** element in the XML document that will be sorted. Often it might not immediately be apparent which element is to be sorted. For example, consider the structure shown in the screenshot below.



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`.

- 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.



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) and 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](#) ²⁹⁸.

User-defined templates

[User-defined templates](#) ²⁴⁸ 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.

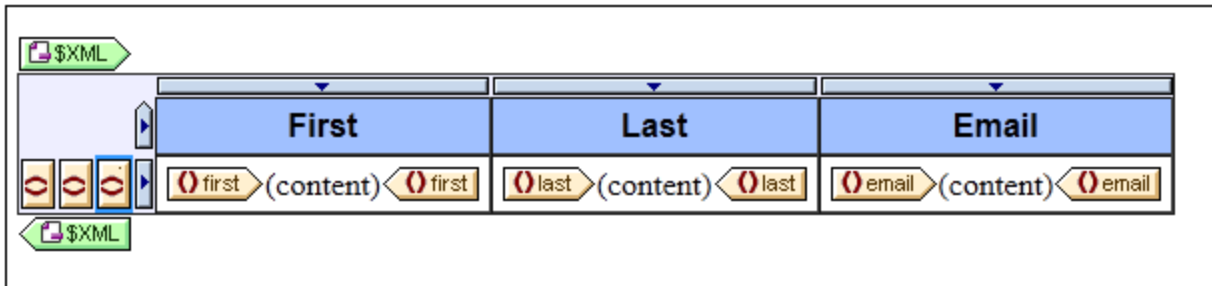
7.5.2 Example: Sorting on Multiple Sort-Keys


In the simple example below (available in the [\(My\) Documents folder](#)²⁹, `C:\Documents and Settings\\My Documents\Altova\StyleVision2024\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.

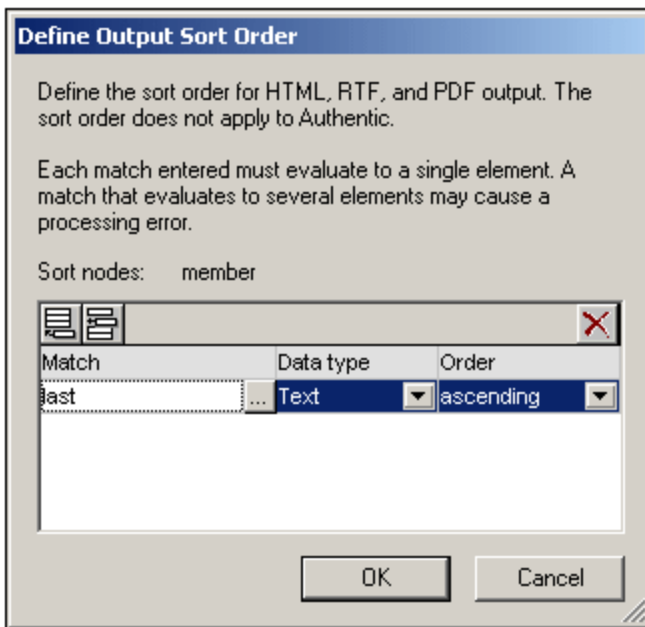
When the list is unsorted, the output order is the order in which the `member` elements are listed in the XML document (*screenshot below, which is the HTML output*).


First	Last	Email
Andrew	Bentinck	a.bentinck@nanonull.com
Nadia	Edwards	n.edwards@nanonull.com
John	Edwards	j.edwards@nanonull.com
Janet	Ashe	j.ashe@nanonull.com

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.



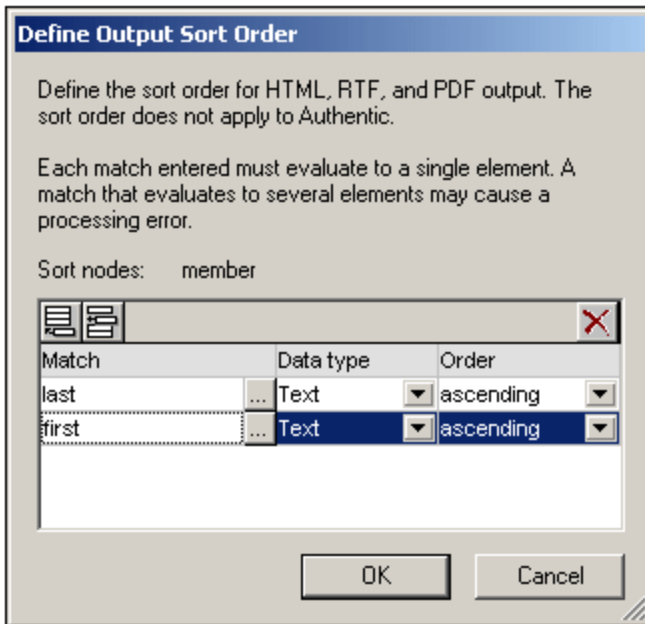
This pops up the Define Output Sort Order dialog (*screenshot below*). Notice that the element selected for sorting, `member`, is named at the Sort Nodes entry. This node is also the context node for XPath expressions to select the sort-key. Click the Add Row button (at left of pane toolbar) to add the 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.

First	Last	Email
Janet	Ashe	j.ashe@nanonull.com
Andrew	Bentinck	a.bentinck@nanonull.com
Nadia	Edwards	n.edwards@nanonull.com
John	Edwards	j.edwards@nanonull.com

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.

First	Last	Email
Janet	Ashe	j.ashe@nanonull.com
Andrew	Bentinck	a.bentinck@nanonull.com
John	Edwards	j.edwards@nanonull.com
Nadia	Edwards	n.edwards@nanonull.com

7.6 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](#)³⁰² explains how user-defined parameters can be used in an SPS.
- [Parameters for Design Fragments](#)³⁰⁴ describes how parameters can be used with design fragments.
- [SPS Parameters for Sources](#)³⁰⁶ 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](#)³⁰⁷ 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.

7.6.1 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](#).

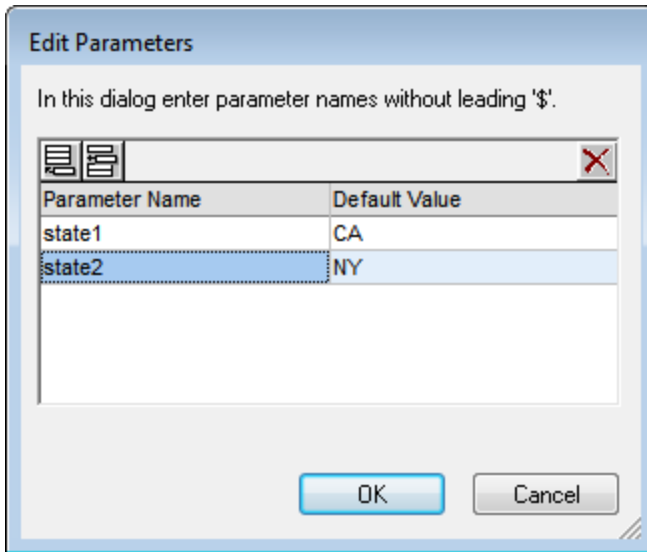
Usage mechanism

Working with user-declared parameters in the SPS consists of two steps:

1. [Declaring the required parameters](#)³⁰².
2. [Referencing the declared parameters](#)³⁰³.

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](#)⁷⁵⁰ command and the **Parameters** button in the Edit Database Filters dialog ([Edit | Edit DB Filter](#)⁸¹⁰).



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](#)⁴².

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](#)³⁰² in the [Edit Parameters dialog](#)⁷⁵⁰.

Note: While it is an error to reference an undeclared parameter, it is not an error to declare a parameter and not reference it.

7.6.2 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:

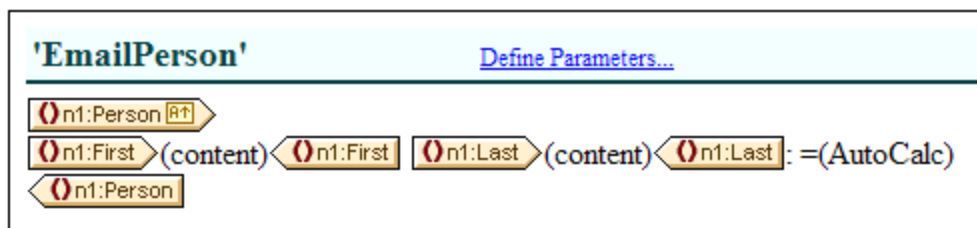
1. [Defining the parameter](#)³⁰⁴ with a default value on the design fragment where it is created.
2. [Editing the parameter value](#)³⁰⁵ where the design fragment is used.

These parts are explained in detail below.

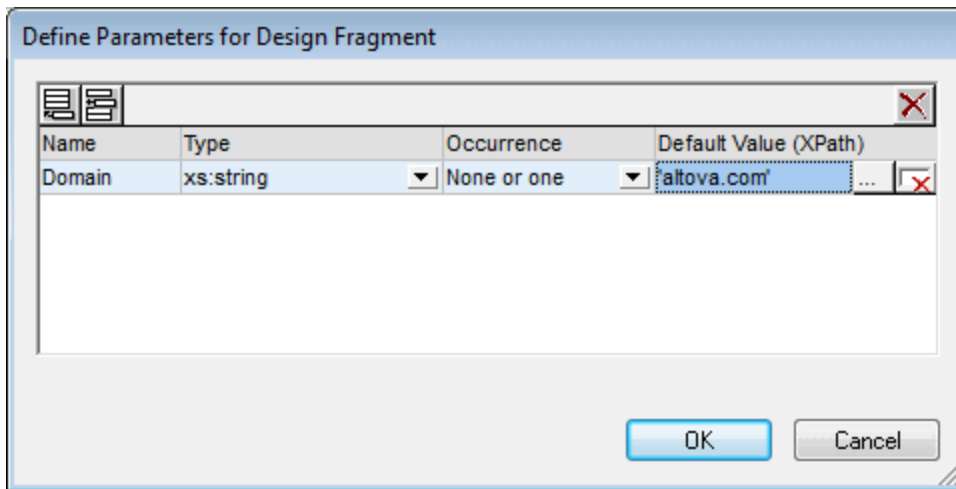
Note: Parameters for Design Fragments are supported in Authentic View only in the Enterprise Editions of Altova products.

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*).



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.

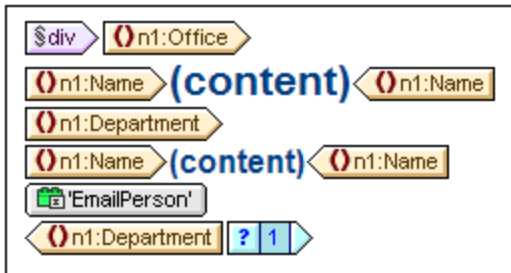


There are two types of **Delete** icon. The Delete icon to the right of each parameter entry deletes the default value of that parameter. The **Delete** icon at the top right of the pane deletes the currently highlighted parameter.

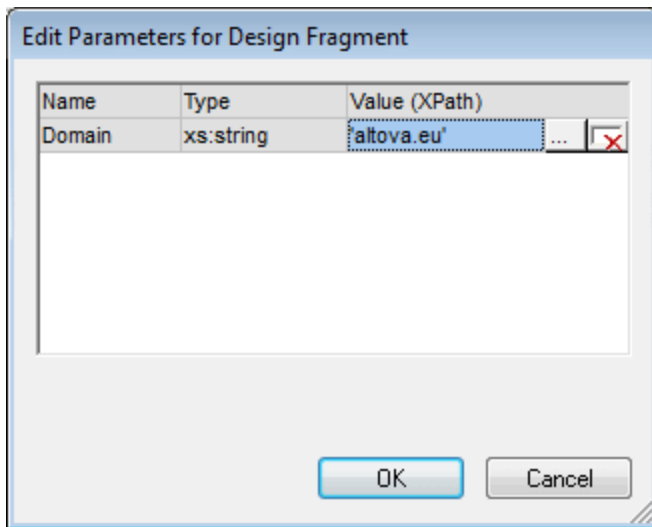
Note: If the SPS uses XSLT 1.0, then the XPath expression you enter must return a node-set. Otherwise an error is reported.

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.

7.6.3 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: `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.

7.6.4 Variables

Using variables consists of two parts: (i) [declaring the variable](#)³⁰⁷, and (ii) [using the variable](#)³⁰⁹.

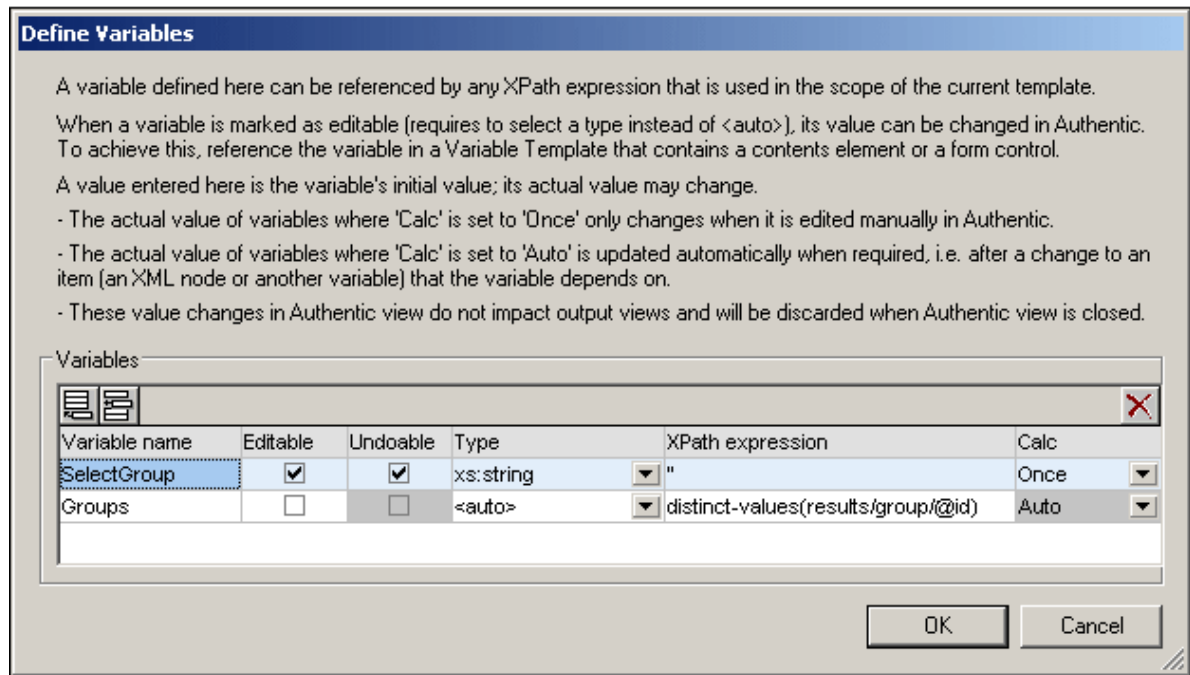
Note: Variables are supported in Authentic View only in the Enterprise Editions of Altova products.

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.



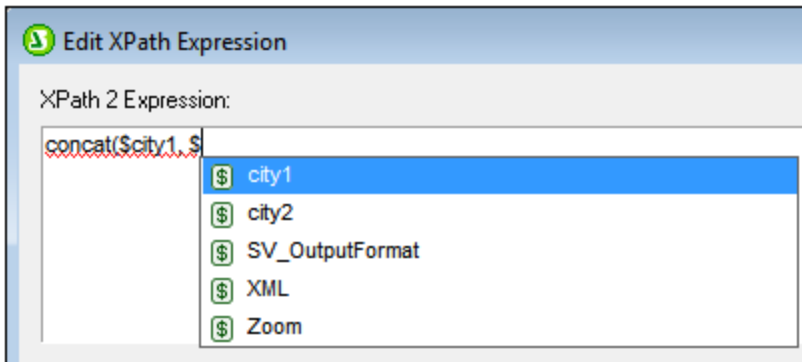
- Setting a variable to Editable (by checking the *Editable* check box) enables the [variable to be edited in Authentic View](#)³⁰⁹. In this case, you must also set the datatype value to the correct type, such as `xs:string`. When a variable is editable, the original value set by the SPS designer can 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.
- 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.
- 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.
- 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](#) ⁶⁸⁵, 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.

7.6.5 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](#) ²⁸⁴ feature.

Example file

The file `AuthenticVariables.sps` in the [\(My\) Documents folder](#) ²⁹, `C:\Documents and Settings\\My Documents\Altova\StyleVision2024\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.

Initial Document Section [Edit Properties...](#) [Add Header/Footer...](#)

Variable defined on template (1)

Combo-box for Authentic View user to select value of variable (2)

User-Defined Template given the variable name (2)

Output-based condition for Authentic View output (2)

Template filter uses the variable to select XML content (3)

Team	P	W	D	L	F	A	Pts
= name (content)	=	=	=	=	=	=	=
= name	(AutoCalc)	(AutoCalc)	(AutoCalc)	(AutoCalc)	(AutoCalc)	(AutoCalc)	(AutoCalc)

Output-based condition to specify different content for each output (3)

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 `$XML` template, a [non-editable variable](#)³⁰⁷ called `Groups` is defined with a type of `<auto>`. Its purpose is to dynamically collect the distinct values of all the `results/group/@id` attributes. These distinct values are planned to be displayed in the dropdown list of the [combo box](#)¹⁷⁷ from which the Authentic View user will select the group he or she wishes to have displayed.
- A [User-Defined Template](#)²⁴⁸ is created and given the name `$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](#)²⁴⁸, 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](#)²⁸⁴ 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](#)²⁵² on the `results/group` template. The XPath expression of this filter is:

```
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](#)²⁸⁴, 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:

The screenshot shows a web interface with a dropdown menu labeled "Select group:" containing options "A", "B", and "All". Below the dropdown, the text "Group B" is displayed. Underneath, a table of football statistics is shown, filtered to only include Group B teams: Argentina, France, England, and Spain.

Team	P	W	D	L	F	A	Pts
Argentina	3	3	0	0	8	1	9
France	3	1	1	1	3	3	4
England	3	0	2	1	2	4	2
Spain	3	0	1	2	0	5	1

Notice the entries in the combo box, the combo box selection of Group B, and the display limited to Group B.

7.7 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](#)³³⁹ 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](#)³¹². 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](#)¹⁰⁵ (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](#)³¹⁵ 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](#)³²² (*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.

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, Text, and RTF 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](#)¹⁰⁵ is the selected XSLT version.

2. [Structure the document in TOC levels](#)³¹⁶. 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](#)³¹⁹ 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 \(levelrefs\)](#)³²². 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](#)³²⁵. 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](#)³²⁵. 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](#)³³⁴, for details.)

Updating TOC page numbers in DOCX and RTF documents

When a user edits a DOCX or RTF output document in MS Word in such a way that the page count changes, it may happen that the TOC is not updated with the new page references. This is an MS Word issue. To update the page references in the TOC, press **Ctrl+A** to select everything, and then press **F9**. For more information, see [here](#).

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).

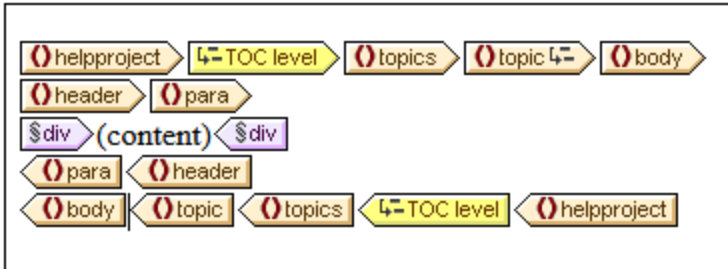
- The **TOC components in the document body** mark out items that will be used in the TOC template.
- The **TOC components in the TOC template** reference the marked items in the document body. Components in the TOC template have the word *'reference'* in their names.


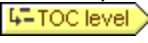
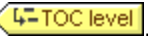
Document body	TOC template
TOC level: The TOC levels structure the document in a nested hierarchy.	Level references (levelrefs): Correspond to the TOC-level structure defined in the document body. Enables TOCrefs in a given level to target TOC bookmarks at the corresponding level.
TOC bookmark: Has a name, with which it identifies a node in the document as a TOC item.	TOC references (TOCrefs): References a TOC bookmark by its name.

7.7.1 Bookmarking Items for TOC Inclusion

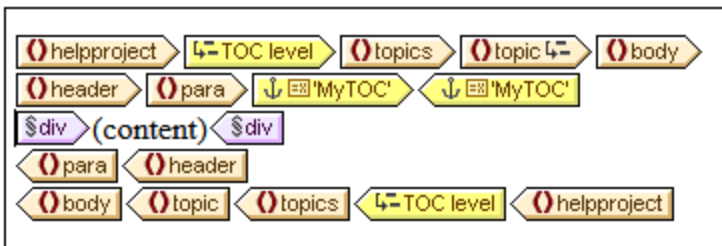
Bookmarking an item in the design for inclusion in a TOC consists of two steps, which can be done in any order:

1. [Structuring the design document in a hierarchy of nested TOC levels](#)³¹⁶. 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 .




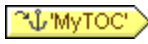
When a level is created on a template, this is indicated by the level icon inside the start tag of the template, for example, . When a level is created around a component it is indicated by TOC level tags  . In the screenshot above, the `topics` template component is enclosed by a level. The difference between the two ways of marking levels is explained in the section [Structuring the Design in Levels](#)³¹⁶. When the [TOC template is created](#)³²², 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](#)³¹⁹ 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.

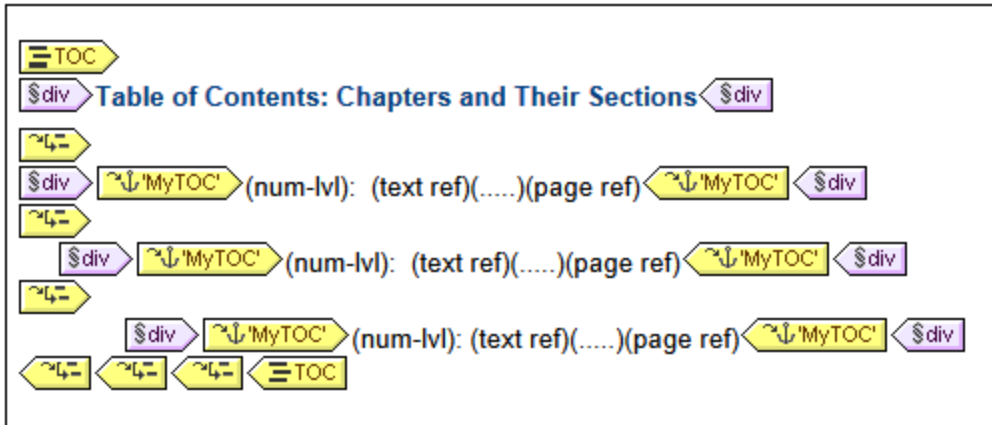



The TOC bookmark serves as an anchor in the document. In the screenshot above, the TOC bookmark (and anchor) is located at the start of `para` element instances. The TOC bookmark has two attributes: (i) a name that will be used to reference the TOC bookmark when creating the TOC item in the TOC template, 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](#)³¹⁹.

How bookmarked items are referenced in the TOC template

The [TOC template](#)³²² 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](#) ³¹⁶, and (ii) how [bookmarks are created](#) ³¹⁹. How the [TOC template is created](#) ³²² is described in the section, [Creating the TOC Template](#) ³²².

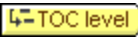

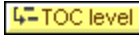

7.7.1.1 Structuring the Design in TOC Levels

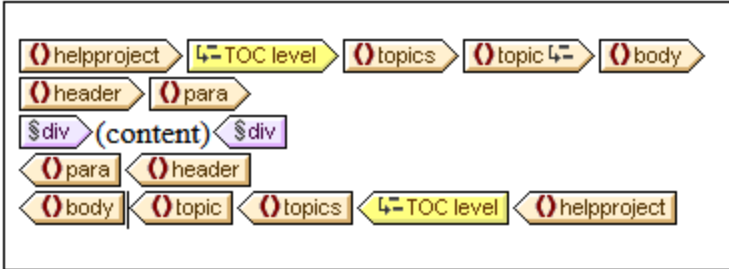
The hierarchical structure you wish to design for the TOC is specified as a set of **nested levels**. As such it is a hierarchical structure which, although related to the XML document structure, is separate from it. This structure is specified in the SPS document design. The TOC template that you construct 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 ) and

another around the `topics` element (indicated by  ). The essential difference between these two ways of creating levels is that the `enclose-within-a-level` option   enables levels to be created around components other than templates.

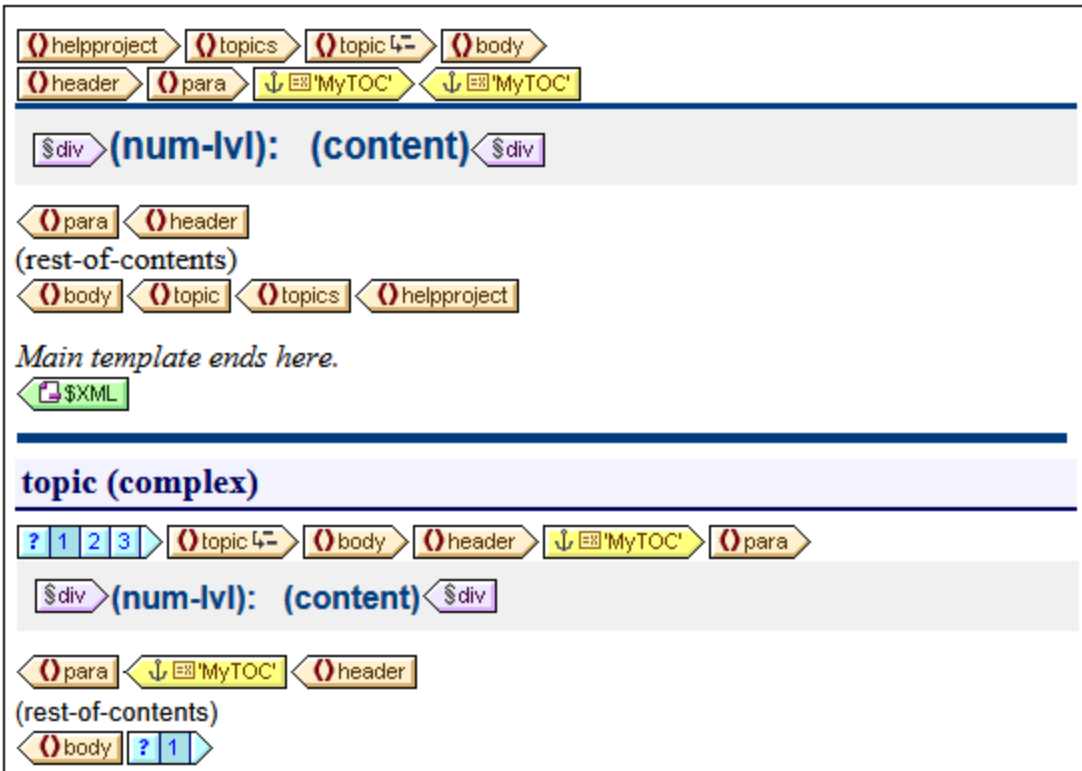


To create a level, do the following:

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 `<XML` tag), a level has been set on the first level of `topic` `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 `topic`). 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

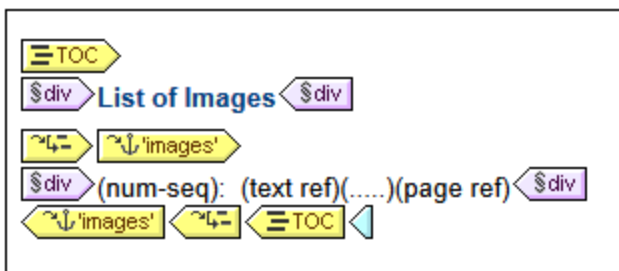
In a flat TOC hierarchy, TOC items will be output at a single level: for example, a simple list of the images in a document.

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.



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.

7.7.1.2 Creating TOC Bookmarks

TOC bookmarks are created within a [TOC level](#)³¹⁶ 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.

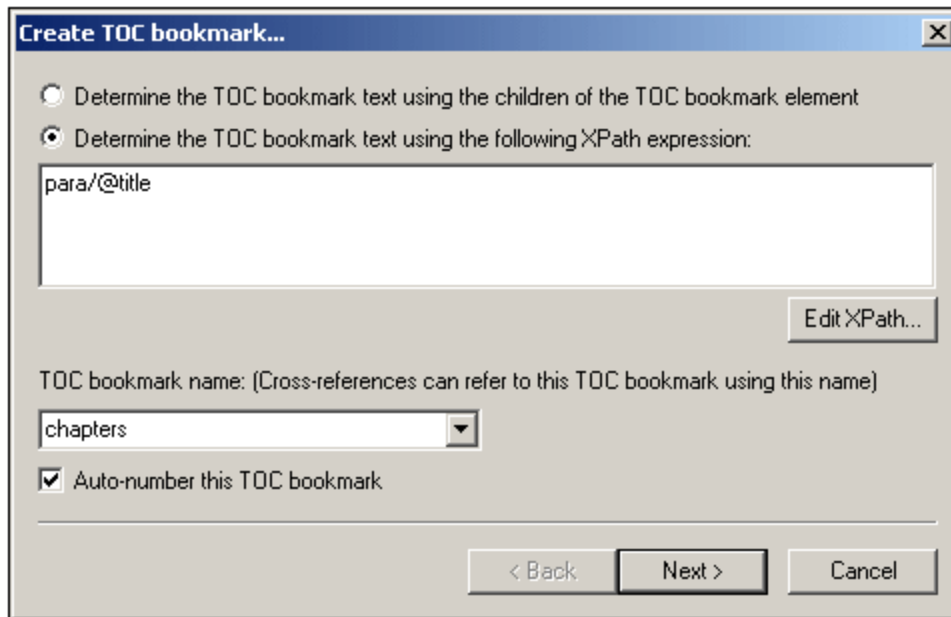
You can create a TOC bookmark in two ways:

- By using the [Create TOC Bookmark Wizard](#)³¹⁹, 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](#)³²¹, the properties of which will be defined subsequently.

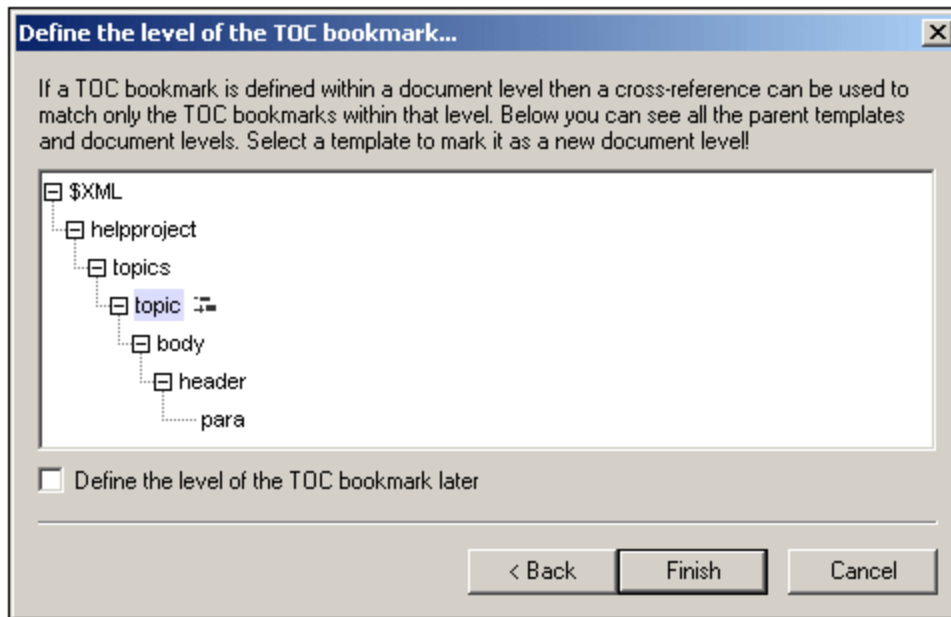
Creating the TOC bookmark with the Create TOC Bookmark Wizard

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 created 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](#)³²² (see *below*).

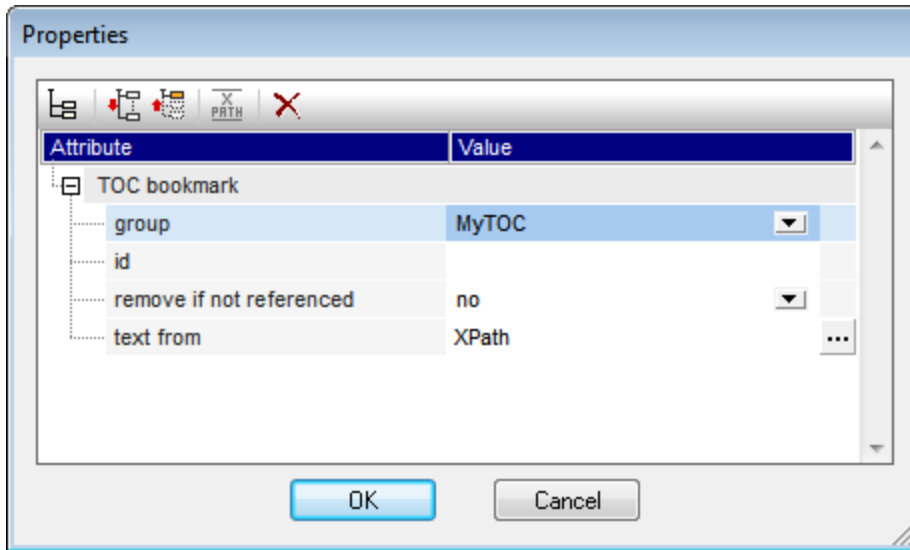
Inserting hierarchical or sequential numbering for a component

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](#)⁵⁵ or right-click the TOC bookmark and select the property you wish to edit.



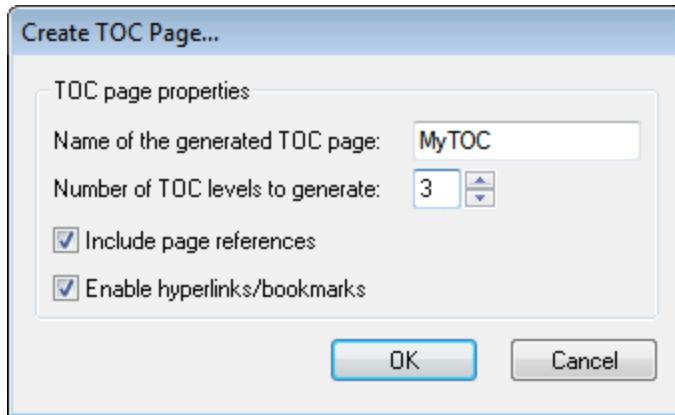
The TOC bookmark has the following properties: (i) the name of the TOC bookmark group (*Group*); (ii) a unique ID; (iii) an option to remove the bookmark if it is not referenced; and (iv) an option (*Text From*) to specify the text entry, which could come from the bookmark's content or from an XPath expression.

7.7.2 Creating the TOC Template

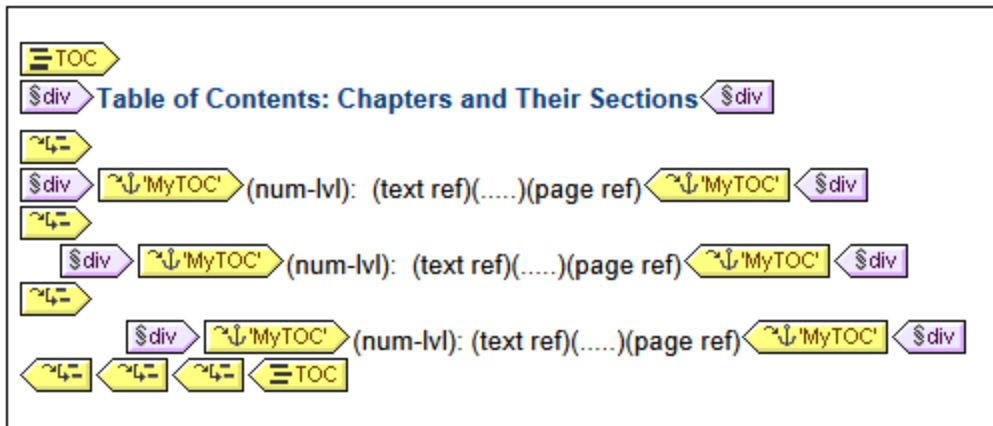
The TOC template is the template that produces the table of contents in the output. It can be created anywhere within the SPS design, and multiple TOC templates can be created in a single SPS design.

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.)



- 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](#)³¹⁹ 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](#)³²⁵). (ii) The number of [TOC level references \(levelrefs\)](#)³¹⁶ 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.
- 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*).



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](#)³²⁵.

Editing the TOC template

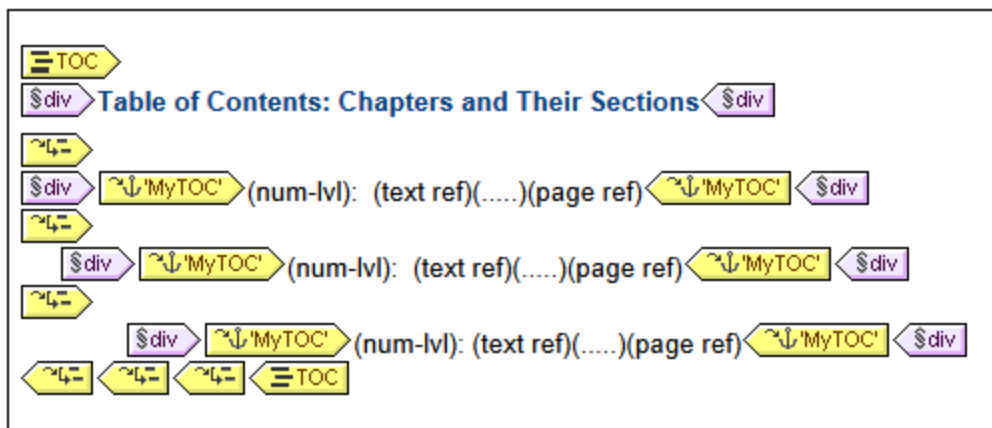
The following editing options are available:

- 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](#)³²⁴ can be added to or deleted from the structure of the TOC template.
- The [properties of individual TOC references](#)³²⁵ (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](#)³²⁵ can be added to or deleted from any levelref in the TOC template.
- The [TOC item](#)³²⁵ within a TOCref can be formatted with CSS properties using the standard [StyleVision mechanisms](#)³⁶⁴.
- Standard SPS features (such as images, Auto-Calculations, and block-formatting components) can be inserted anywhere in the TOC template.

7.7.2.1 Levelrefs in the TOC Template

The [TOC template](#)³²² 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](#)³²².



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](#)³¹⁹ within a [specified scope](#)³²⁵ 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.

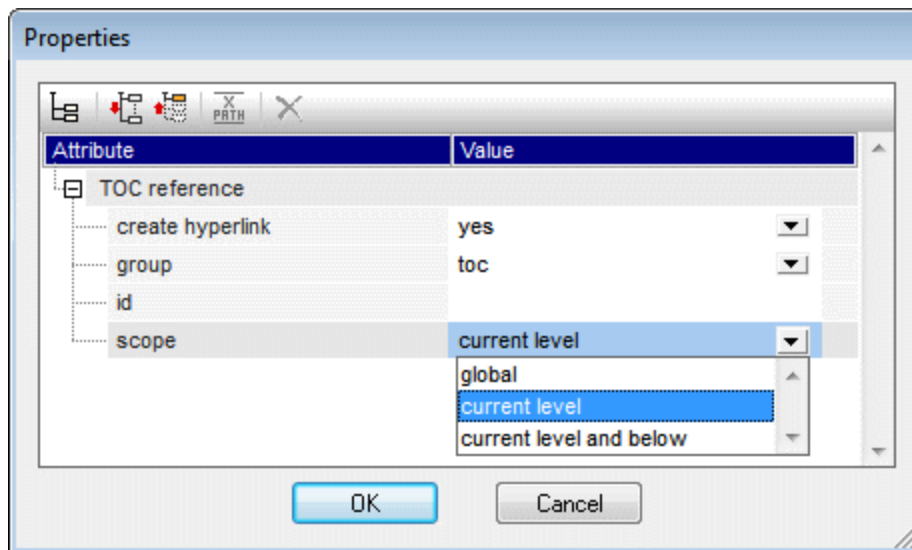
7.7.2.2 TOC References: Name, Scope, Hyperlink

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*).



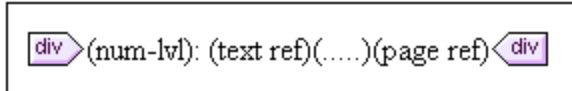
Alternatively, with the TOCref tag selected, go directly to the required property in the Properties window (*TOC reference* group of properties).

7.7.2.3 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

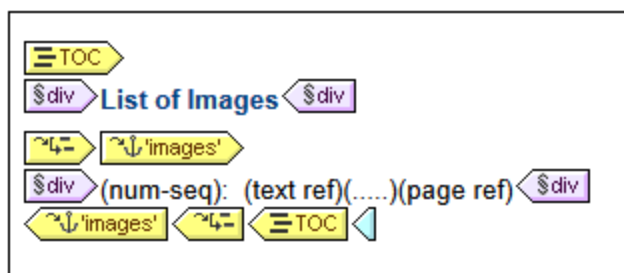
In a flat TOC hierarchy, TOC items will be output at a single level: for example, a simple list of the images in a document.

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.



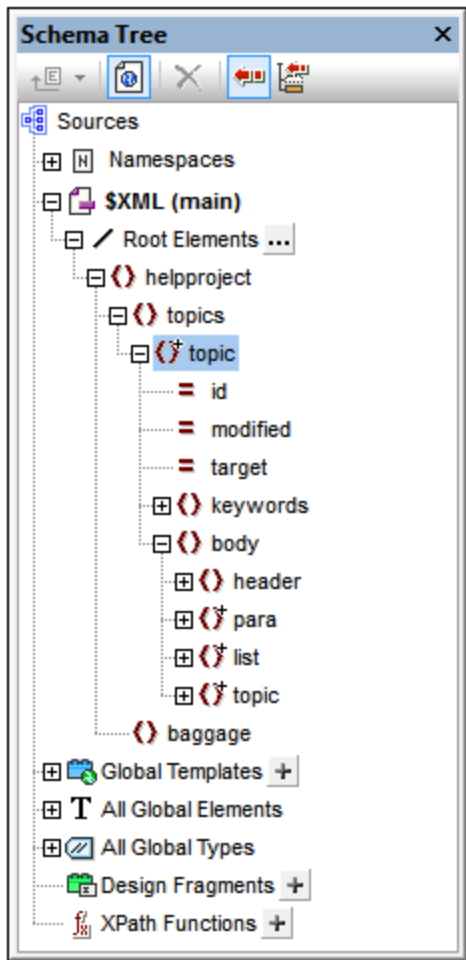
If the design contains more than one level, and a flat TOC is required, say, for items in the second level, then the TOC template could have two levelrefs with a TOCref only within the second level (no TOCref within the first level). Alternatively, the scope property of TOCrefs can be used to specify 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](#)³⁶⁴ via the [Styles sidebar](#)³⁷¹. Individual TOC item components can be separately formatted by selecting the component and assigning it [style properties](#)³⁷¹ in the Styles sidebar.

7.7.3 Example: Simple TOC

An example SPS file to demonstrate the basic use of TOCs, called `ChaptersSimple.sps`, is in the [\(My\) Documents folder](#)²⁹, `C:\Documents and Settings\\My Documents\Altova\StyleVision2024\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](#)³³¹.)



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](#)³¹⁶: One or more levels are inserted in the design document to structure the (output) document hierarchically. This hierarchic 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](#)³¹⁹: 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](#)³²²: 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.



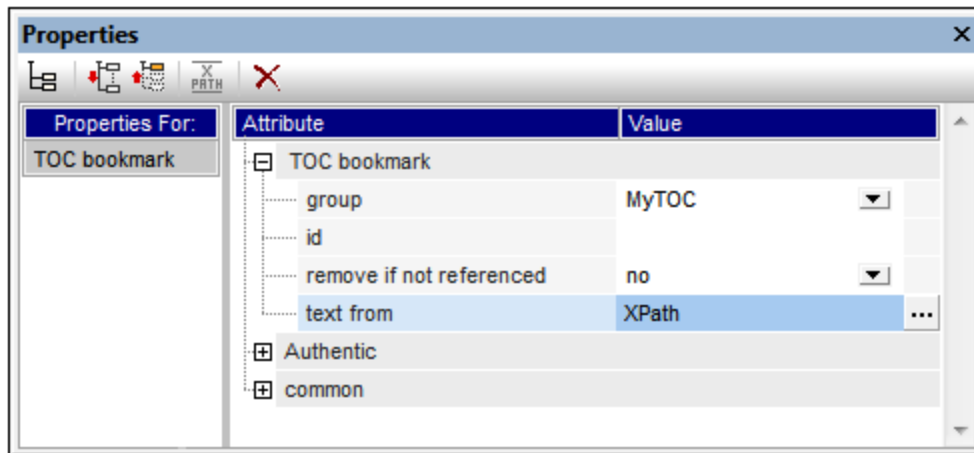
The screenshot above shows the contents of the first conditional branch, for first-level, chapter-type `topic` elements. Note that a TOC level has been created on the template start-tag of this `topic` element. In the other two conditional branches no TOC level has been created on the topic template. 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.




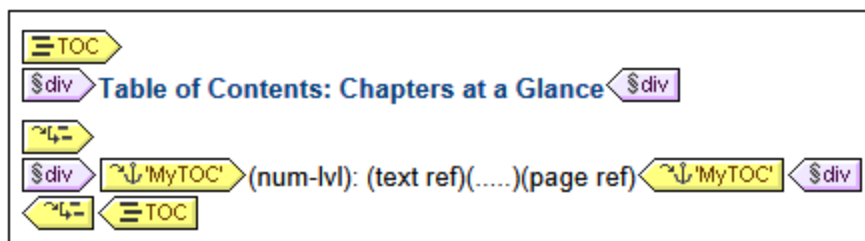
The properties of the TOC bookmark can be edited in the Properties sidebar (*screenshot below*).

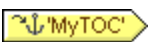


The *Group* property sets the TOC bookmark group (and is the name of the TOC bookmark). In our example, we have specified the value `MyTOC` for this property. The bookmark group will be referenced in the TOC when it is created, and it enables different TOC groups to be specified within the same level. 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).

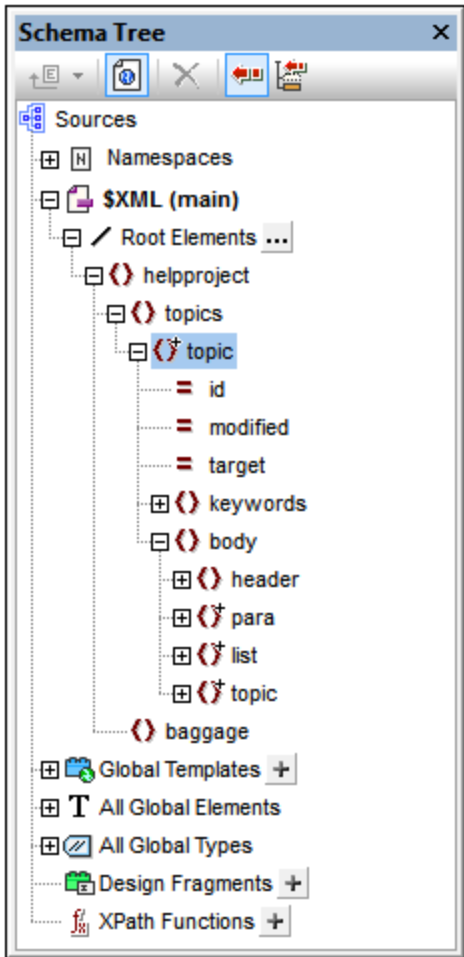


Within this levelref, a TOC reference (TOCref)  has been inserted. This TOCref has been set to select bookmarks (i) that are in the bookmark group named `MyTOC` (see 'Creating the TOC bookmark' above), and (ii) that are within the scope of the current level only. These settings can be made in the Properties 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.

7.7.4 Example: Hierarchical and Sequential TOCs

An example SPS file to demonstrate the use of TOCs, called `Chapters.sps`, is in the [\(My\) Documents folder](#)²⁹, `C:\Documents and Settings\\My Documents\Altova\StyleVision2024\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](#)³³², which lists the names of each chapter (the first-level topics).
2. [Chapters and their sections](#)³³³, 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](#)³³³, 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).



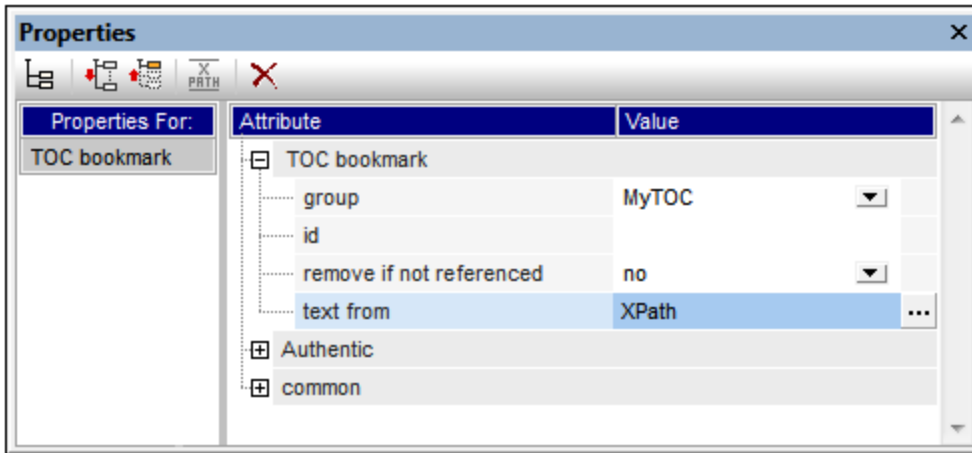
Since the `topic` element is recursive, the TOC level and the TOC bookmark will also recurse. This means that, at the first recursion, a new hierarchically subordinate TOC level and a new TOC bookmark is created. This process continues for each descendant topic, thus creating a hierarchy of descendant 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.

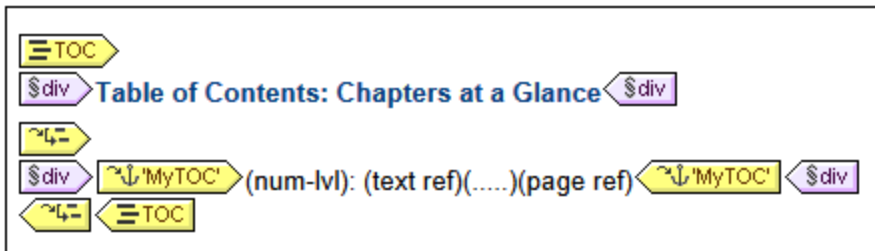
TOC descriptions

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.

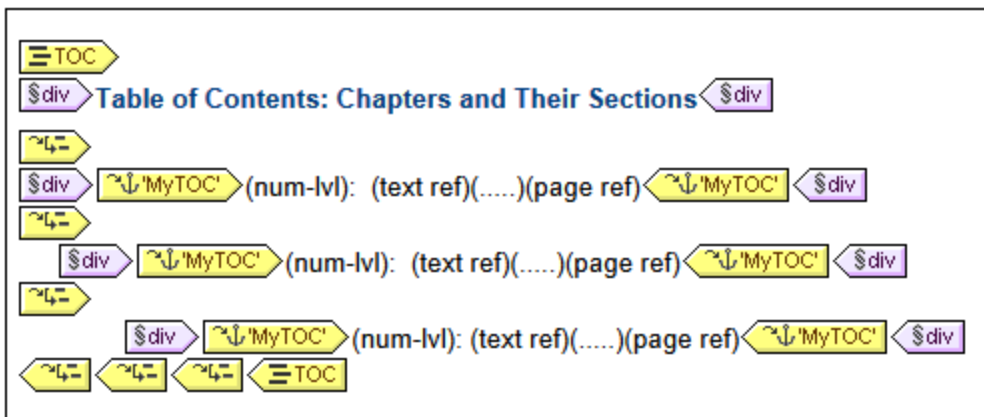


The TOC template itself (screenshot below) contains one level reference (levelref) , and the TOCref within that levelref  has been set to select TOC bookmarks named `MyTOC` within the scope of the current level only—which is the first level. As a result, TOC items will be created only for first-level 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

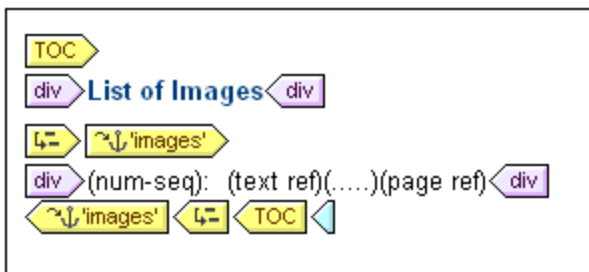
topic elements, `image` elements will occur within the first, second, and/or third levels of the document. There is therefore no need to create any level for the `image` element.

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).



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.



Since the selected numbering is sequential, the images are numbered sequentially in a flat list.

7.7.5 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.

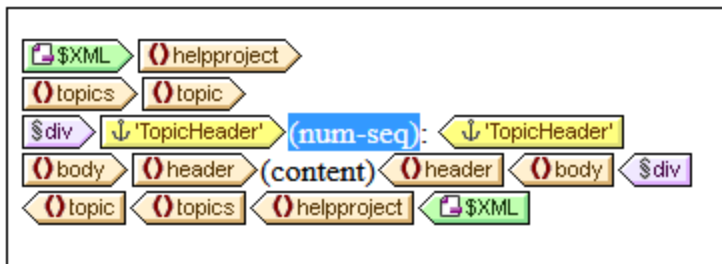
Auto-Numbering can be either sequential (flat) or hierarchical. Sequential numbering provides ordinary numbering on a single level. Hierarchical numbering is based on the TOC-level hierarchy created in the document and creates numbering according to the element's position in the TOC-level hierarchy.

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](#)³¹⁹ 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*).



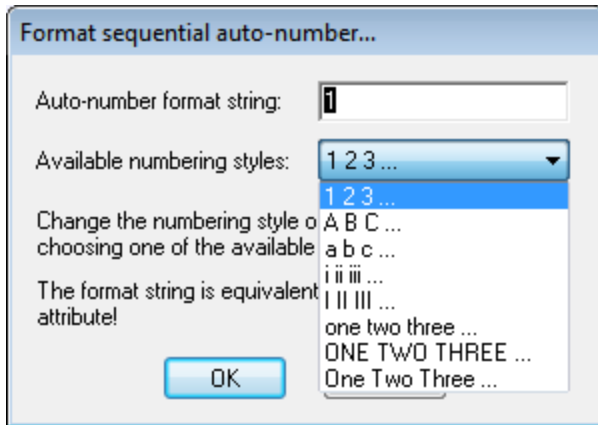
3. If the TOC bookmark is going to be referenced from within a TOC template, then you can enter TOC bookmark properties as required. However, if the TOC bookmark is going to be used only for sequential numbering, there is no need to name it. If you wish to name it, right-click it and select the **Edit 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.

```
1: Altova StyleVision 2007
2: About this Documentation
3: Introduction
4: User Interface
```

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*).

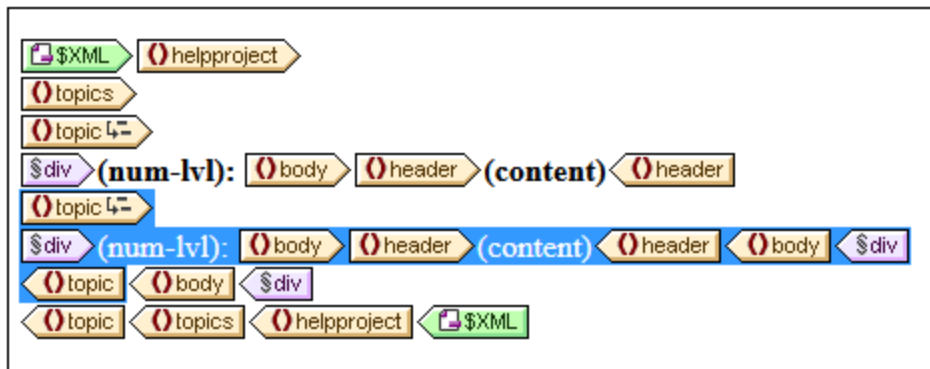


Select the format you want from the dropdown box of the *Available numbering styles* combo box (see *screenshot above*) and click **OK** to apply the selected format.

Hierarchical numbering (num-lvl)

Hierarchical numbering can be inserted within a [TOC level in the design](#)³¹⁶. 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](#)³¹⁶. 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*).



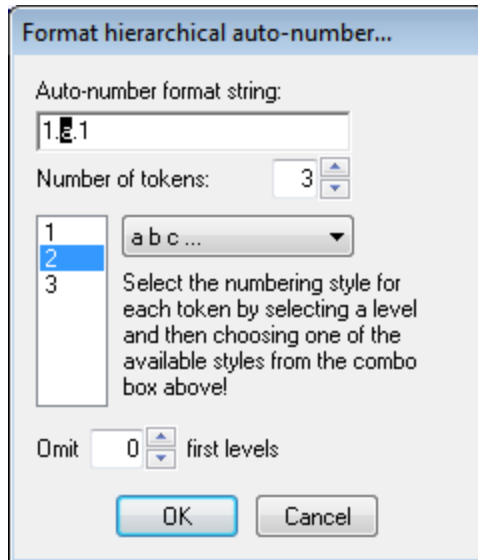
In the screenshot above, there are two levels. The `topic` element is recursive, and a level has been created on two `topic` elements (by right-clicking the node tag and selecting **Template Serves as Level**). One `topic` element (*highlighted in the screenshot above*) is nested within the other. As 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-1v1` 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-1v1` at each level must be separately formatted.

Click **OK** when done.

7.7.6 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:

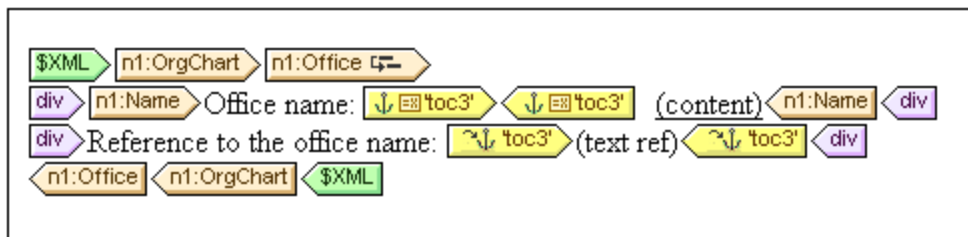
Step 1: Levels

The document is structured into TOC levels as described in the section [Structuring the Design in Levels](#)³¹⁶. 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.

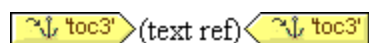
In the screenshot below, the TOC bookmark within the `n1:Name` element  has a name of `toc3` and an XPath expression that locates the current node. This means that the output text will be the contents of the `n1:Name` node.



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.

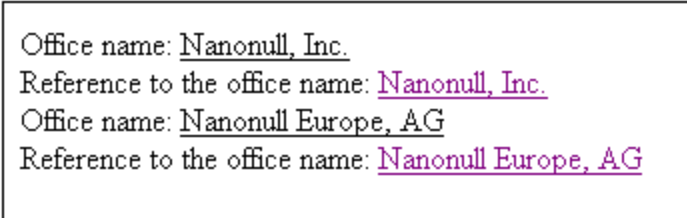
Step 3: Creating TOC references

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:



```
Office name: Nanonull, Inc.  
Reference to the office name: Nanonull, Inc.  
Office name: Nanonull Europe, AG  
Reference to the office name: Nanonull Europe, AG
```

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.

7.7.7 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](#)³³⁹ can be inserted in the SPS.
- How [hyperlinks](#)³⁴¹ 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.

7.7.7.1 Inserting Bookmarks

A bookmark (or anchor) can be inserted anywhere in the SPS, at a cursor insertion point or around an SPS component.

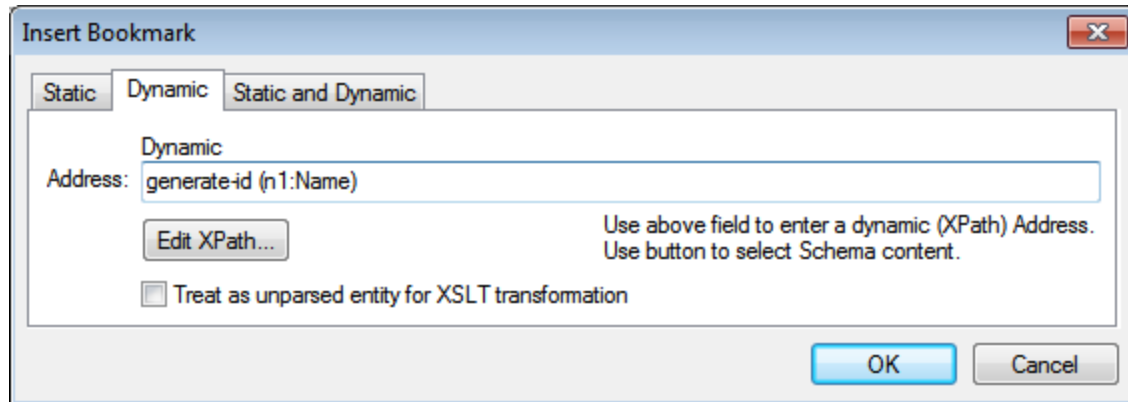
Bookmarks are created in the SPS via the Insert Bookmark dialog (*screenshot below*). In this dialog you define the name of the bookmark. The name can be a static name, or it can be a dynamic name that is (i) derived from XML document content, or (ii) generated arbitrarily with an XPath expression.

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](#)⁷⁷⁵, 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.

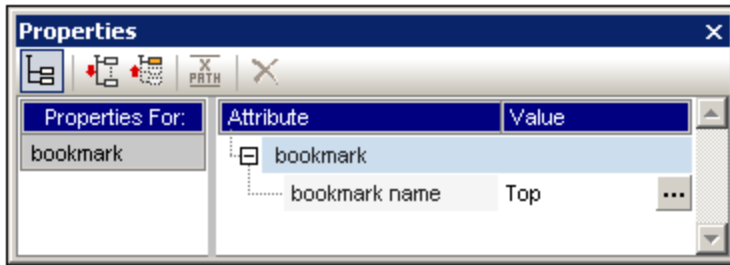
After a bookmark has been created, it can be [linked to by a hyperlink](#)³⁴¹.

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](#)³⁴⁴. 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.

7.7.7.2 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.

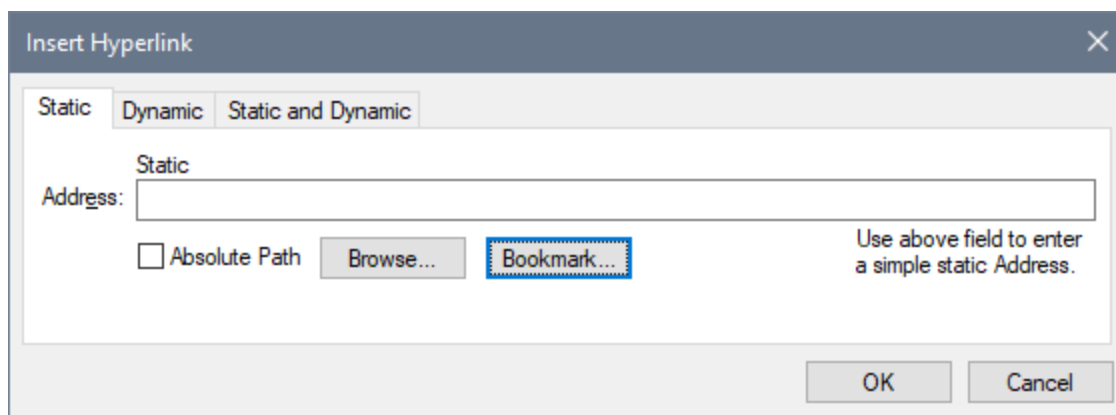
Creating hyperlinks

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 the hyperlink is created in the Insert Hyperlink dialog (*screenshot below*), which is accessed via the [Enclose With | Hyperlink](#)⁽⁷⁹⁰⁾ or [Insert | Hyperlink](#)⁽⁷⁷⁶⁾.



The target of a link can be either:

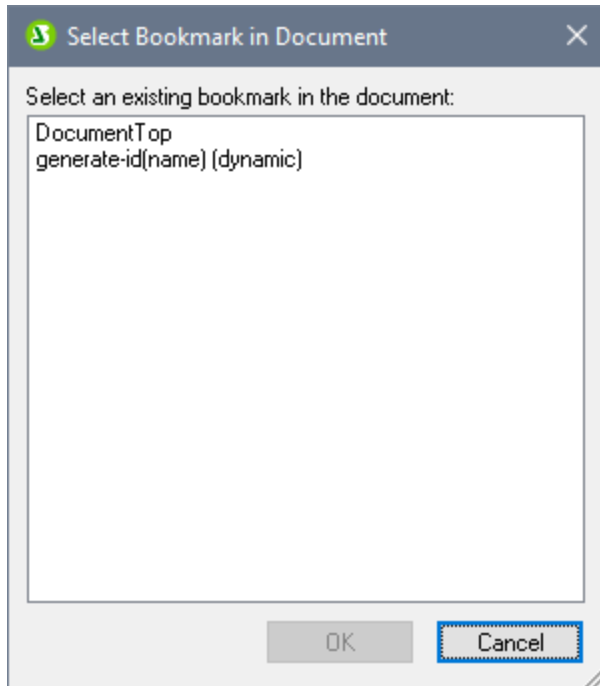
- A [bookmark](#)³⁴² in the same SPS design (in which case the target URI must be a fragment identifier),
- [Dynamically generated](#)³⁴⁴ to match bookmark anchors (these URIs are also fragment identifiers),
- An [external resource](#)³⁴⁴; the URI can be static (directly entered), dynamic (taken from 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.

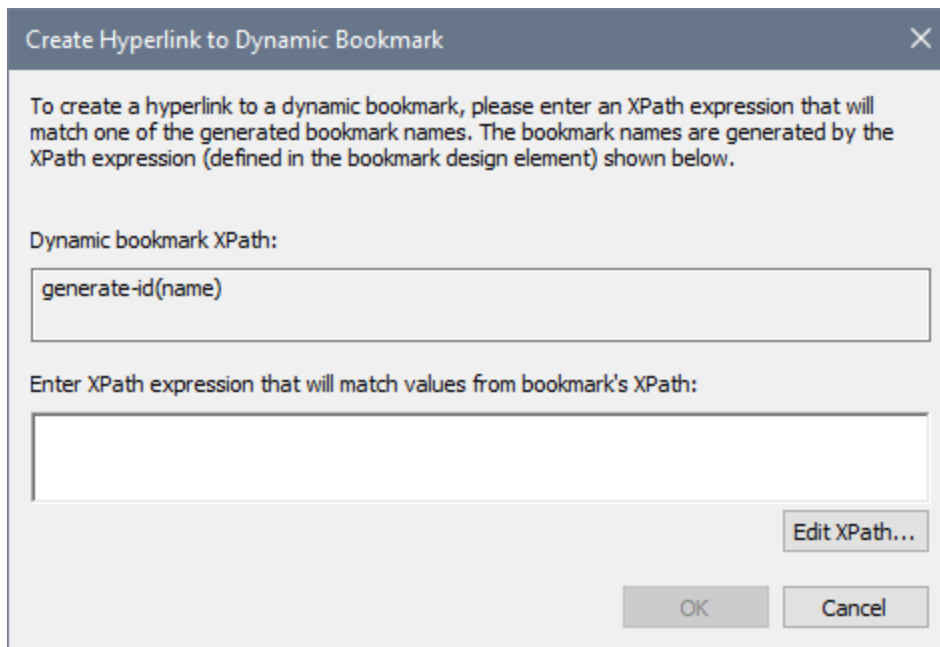
Linking to bookmarks

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](#)³³⁹ 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](#)³³⁹. 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](#)⁷⁷⁶ 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](#)⁷⁷⁶. 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.

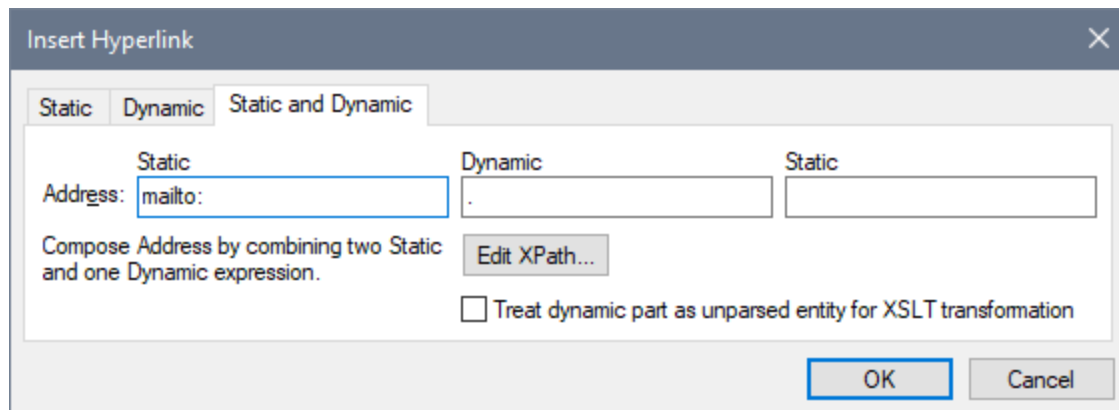
Two points should be borne in mind:

- 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](#)⁴⁴⁰ 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 could be created by 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*). The **Edit XPath** button opens the [Edit XPath Expression dialog](#)⁶⁹⁷ to help you build the dynamic part of the hyperlink.



How to use unparsed entities is described in the section [Unparsed Entity URIs](#)⁴⁴⁰.

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](#)⁴³⁵. 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.

7.8 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\\My Documents\Altova\StyleVision2024\StyleVisionExamples.`)

Using variables and conditions

The user's preferred language is entered in an [editable variable](#)³⁰⁹. A [condition](#)²⁸⁰ with multiple branches maps each language to the correct language content. The user's language choice is used to select the correct conditional branch.

The screenshot displays a form with a language selection section at the top. Below it is a table with columns for field names and their corresponding content. The table is designed to handle two languages: English and German. Each cell in the table contains a condition that checks the value of the language variable and then displays the appropriate content.

Field Name	Content
Vorname	First (content) First
Last	Last (content) Last
Middle Initial	Middle (content) Middle
Maiden Name	Maiden (content) Maiden
Date of Birth (month/day/year)	DateOfBirth ? 1 2 (content) DateOfBirth
Social Security #	SocialSecurity (content) SocialSecurity

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.

English German

Vorname	Niki
Nachname	Devgood
2. Vorname	
Mädchenname	
Geburtstag (Tag.Monat.Jahr)	16.02.1980 <input type="text" value="1"/>
Sozialversicherungsnummer	555-55-555

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](#)³⁰², the value of which triggers the required language output. The appropriate language output can be determined, for example, by means of an [Auto-Calculation](#)²⁷⁰. 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\\My Documents\Altova\StyleVision2024\StyleVisionExamples`.

8 Presentation and Output

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, Word 2007+ and Text in the Enterprise Edition; Authentic View, HTML, RTF, and Text 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). 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](#)³⁹². These [paged media options](#)³⁹², such as page size, page layout, and headers and footers, are defined additionally to the styling of components, and will be used for RTF output alone.

Note: Since plain text cannot be formatted, text styling properties and page layouts in the design will be ignored when generating Text output.

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](#)³⁶⁴ 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](#)³⁷⁵. The benefits of using XPath expressions are: (i) that the property value can be 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](#)³⁵⁰ 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](#)³⁵⁰ sub-section of this section. Note that the inherent styles of predefined formats are converted to equivalent markup for RTF 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). Also, when HTML selectors are used (in external stylesheets and global style rules), these will not be applicable to paged media output (RTF). When such selectors are used, a comment is displayed next to the selector to the effect that the style will not be applied to Text, RTF output.

Designing for paged media output

For StyleVision's paged media support (RTF outputs and XSLT stylesheets for RTF), [page definition and layout options](#)³⁹² are available. These options are used additionally to the component styling mechanism, and are described in the [Designing Print Output](#)³⁹² sub-section of this section.

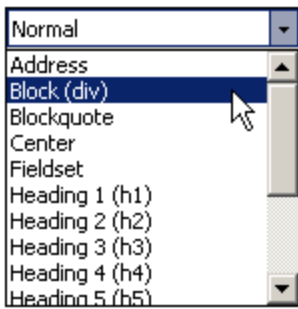
8.1 Predefined Formats

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](#)³⁷¹ 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](#)³⁷¹.

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.

8.2 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 `A`
- the decimal character reference of the character `A` is `A`
- the HTML (and XML) entity reference of the character `&` is `&`
- the hexadecimal (or Unicode) character reference of the character `&` is `&`
- the decimal character reference of the character `&` is `&`
- the HTML (and XML) entity reference of the character `<` is `<`

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 `all` (to set `disable-output-escaping` to `yes` for all outputs except Text).

Note: Output escaping does not apply to Text output. (It is neither enabled or disabled.)

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.

Static text	<code>disable-output-escaping</code>	Output text
-------------	--------------------------------------	-------------

&	no	&
&	yes	&
&	no	&
&	yes	&
<	no	<
<	yes	<
A	no	A
A	yes	A
&lt;	no	&lt;
&lt;	yes	<
&amp;lt;	yes	<
&<	yes	&<

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:

```
<b>This text is bold.</b>
```

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.

8.3 Value Formatting (Formatting Numeric Datatypes)

Value Formatting enables the contents of numeric XML Schema datatype nodes (see [list below](#)³⁵⁴) 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, Text, and RTF 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](#)³⁵⁴) for which Value Formatting is available.

In the sub-sections of this section, we describe:

- how the [Value Formatting mechanism works](#)³⁵⁴, and
- the [syntax](#)³⁵⁷ 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`


8.3.1 The Value Formatting Mechanism

Value Formatting can be applied to:

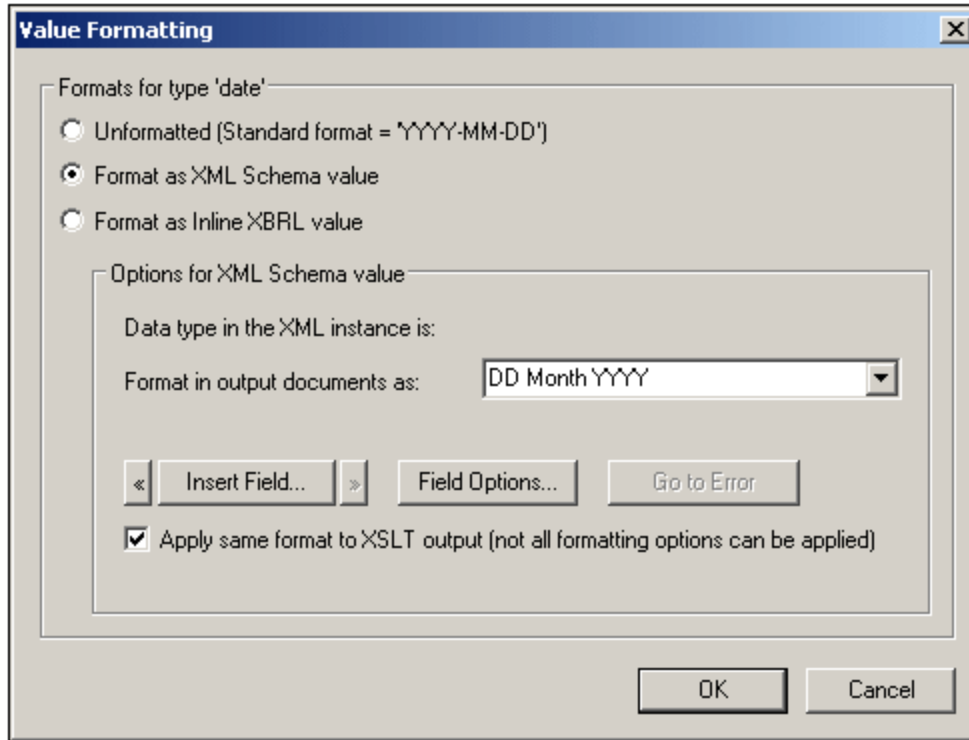
- A [numeric datatype node](#)³⁵⁴, such as `xs:decimal` or `xs:date` that is present in the SPS as **contents or an input field**.
- An Auto-Calculation that evaluates to a value which has the lexical format of a [numeric datatype](#)³⁵⁴.

Defining Value Formatting

To define Value Formatting for a node or Auto-Calculation in the SPS, do the following:

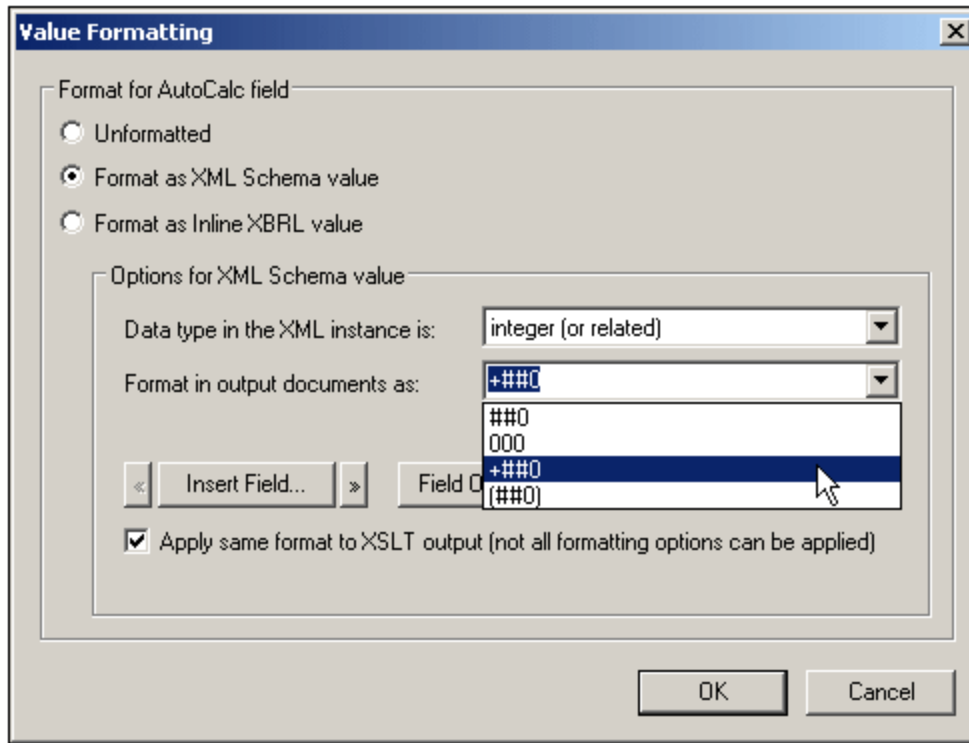
1. Select the `contents` placeholder or input field of the node, or the Auto-Calculation.
2. In the Properties sidebar, select the item, and then the *Content* group (or *AutoCalc* group) of properties. Now click the Edit button  of the `Value Formatting` property. Alternatively, right-click

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.



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](#)³⁵⁷ 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, Text, RTF 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).

8.3.2 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.

Field Type	Explanation
#	Space if no digit at position
0	Zero if no digit at position
.	Decimal mark
,	Digit group separator
Y	Year
y	year (base = 1930); see Note below
MM	Month, must have length of 2
DD	Day, must have length of 2
W	Week number
d	Weekday number (1 to 7)
i	Day in the year (1 to 366)
hh	Hour (0 to 23), must have length of 2
HH	Hour (1 to 12), must have length of 2
mm	Minute, must have length of 2
ss	Second, must have length of 2
AM	AM or PM
am	am or pm
AD	AD or BC

ad	ad or bc
CE	CE or BCE
ce	ce or bce

Field Type	Explanation
Weekday	Weekday (Sunday, Monday...)
WEEKDAY	Weekday (SUNDAY, MONDAY...)
weekday	Weekday (sunday, monday...)
Wkd	Weekday (Sun, Mon...)
WKD	Weekday (SUN, MON...)
wkd	Weekday (sun, mon...)
Month	Month (January, February...)
MONTH	Month (JANUARY, FEBRUARY...)
month	Month (january, february...)
Mon	Month (Jan, Feb...)
MON	Month (JAN, FEB...)
mon	Month (jan, feb...)

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 YYY, then the output for a value of 2006 would be 006; for a definition of YYYYYY, 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.

Prefix	Suffix	Explanation
<		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.
>		Right aligned; default for numbers.
?		Minus symbol adjacent to number if negative; nothing otherwise. This is the default for numbers.
<?		Minus symbol left-aligned if negative; nothing otherwise. Number left-aligned, follows minus sign.
<?>		Minus symbol left-aligned if negative; nothing otherwise. Number right-aligned.
-	-	Minus symbol adjacent to number if negative; space otherwise. Located before number (prefix), after number (suffix).
<-	>-	Minus symbol if negative; space otherwise. Number and sign adjacent. Left-aligned (prefix); right-aligned (suffix).
<->		Minus symbol left-aligned if negative; space otherwise. Number right-aligned.
+	+	Plus or minus sign always, located adjacent to number; before number (prefix), after number (suffix).
<+	>+	Plus or minus sign always, located adjacent to number; left-aligned (prefix), right-aligned (suffix).
<+>		Plus or minus sign always, left-aligned; number right-aligned.
()	Parentheses if negative; space otherwise. Adjacent to number.
<(Parentheses if negative; space otherwise. Adjacent to number. Left-aligned.
<(>		Parentheses if negative; space otherwise. Left parentheses left-aligned; number and right parentheses adjacent and right-aligned.
[]	Parentheses if negative; nothing otherwise. Adjacent to number.
*	*	Extendable number of digits to left (prefix) or to right (suffix)
_	_	Space
^	^	Fill character (defined in options)
	th	Ordinality of number in EN (st, nd, rd, or th)
	TH	Ordinality of number in EN (ST, ND, RD, or TH)

Field options

Field options enable advanced modifications to be made to fields. The following options are available:

Option	Explanation
uc	Make uppercase
lc	Make lowercase
left	Left aligned
right	Right aligned
ro	Read (XML) only; no editing allowed
edit	The field is editable (active by default)
dec=<char>	Specify a character for the decimal point (default is point)
sep=<char>	Specify a character for the digit separator (default is comma)
fill=<char>	Specify fill character
base=<year>	Base year for year fields (<i>see note below</i>)
pos	Show only positive numbers; input of negative numbers allowed

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 `dec` and `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 (**Control Panel | All Items | Region | Format**).

Note on Base Year

When using **short year formats** (such as `yy` and `YY`), the base year specifies a cut-off for a century. For example, the base year field option could be used in the definition `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.

8.4 Working with CSS Styles

The SPS design document is styled with CSS rules. Style rules can be specified:

- In [external CSS stylesheets](#)³⁶⁵. External CSS stylesheets can be added via the [Design Overview](#)⁴² sidebar and via the [Style Repository](#)⁵¹ sidebar.
- In [global styles](#)³⁶⁹ 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](#)⁵¹ sidebar.
- [Locally](#)³⁷¹, 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](#)⁵⁴ sidebar.

Each of the above methods for creating styles is described in detail in the sub-sections of this section ([links above](#)).

Note: Since CSS styles cannot be applied to plain text, they will not be applied to Text output.

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](#)⁵⁴ sidebar (local styles) and [Style Repository](#)⁵¹ 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](#)²³⁰.

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](#)⁷³⁸ whether the HTML and Authentic View output documents should be styled according to [Internet Explorer's older or newer interpretation](#)¹⁰⁶. You should then set CSS styles according to the selected interpretation. For more details, see [Properties: CSS Support](#)⁷⁴⁰.

Note: For more information about the CSS specification, go to <http://www.w3.org/TR/REC-CSS2/>.

8.4.1 External Stylesheets

This section describes how external CSS stylesheets can be managed from within the StyleVision GUI. It consists of the following parts:

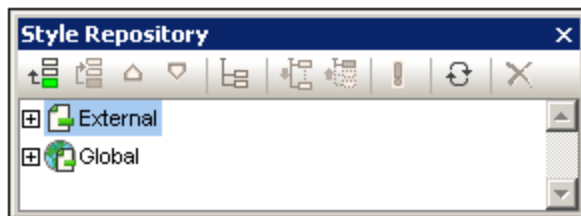
- [Adding an external CSS stylesheet to the SPS](#)³⁶⁵
- [Viewing the contents of an external CSS stylesheet and modifying the media applicability](#)³⁶⁶
- [Changing the precedence](#)³⁶⁷
- [Switching between the full CSS stylesheet set and a single CSS stylesheet](#)³⁶⁷

External CSS stylesheets can be managed from two sidebars: the [Style Repository sidebar](#)⁵¹ and the [Design Overview sidebar](#)⁴². 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*).



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](#)⁴² sidebar.

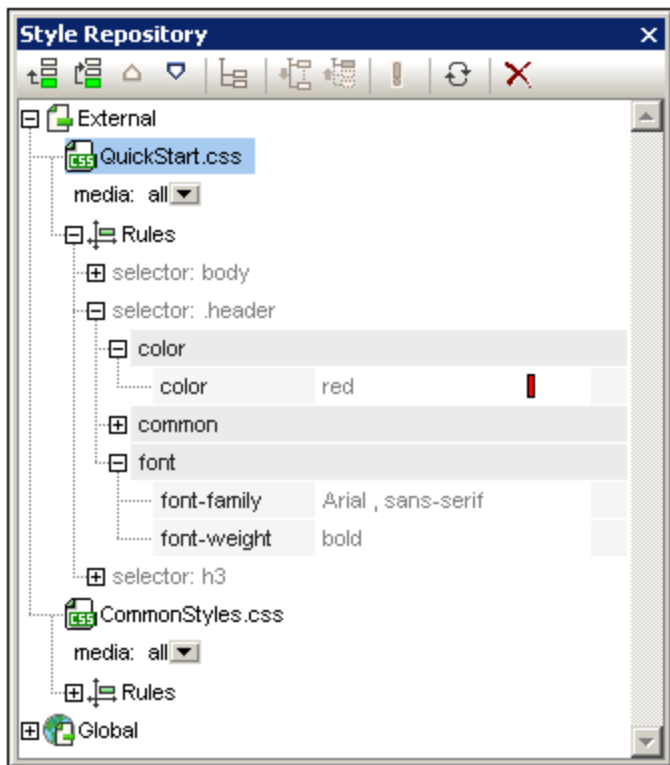
Viewing and modifying the tree of external CSS stylesheets

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
 - ...
 - Property-N
 - ...
 - Selector-N
 - + ...
- + CSS-N.css

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:

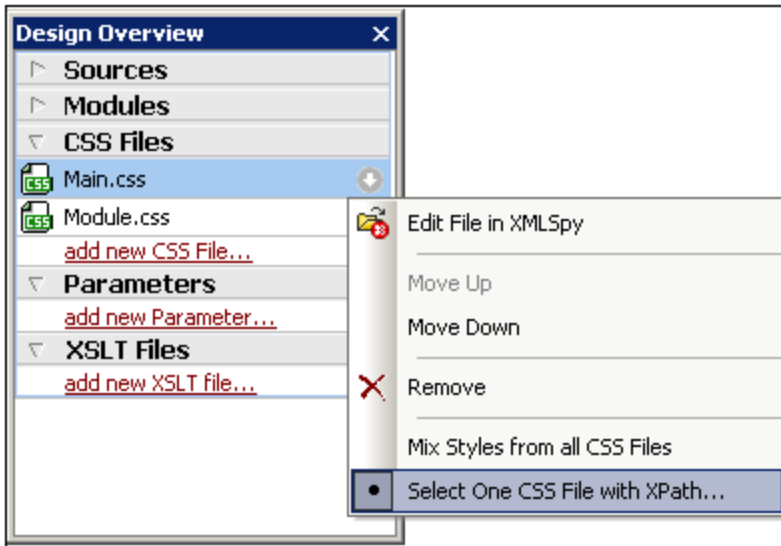
```
<html>
  <head>
    <style>
      <!--
        @import url("ExternalCSS-1.css");
        @import url("ExternalCSS-2.css")screen;
        @import url("ExternalCSS-3.css")print;
      -->
    </style>
  </head>
  <body/>
</html>
```

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.

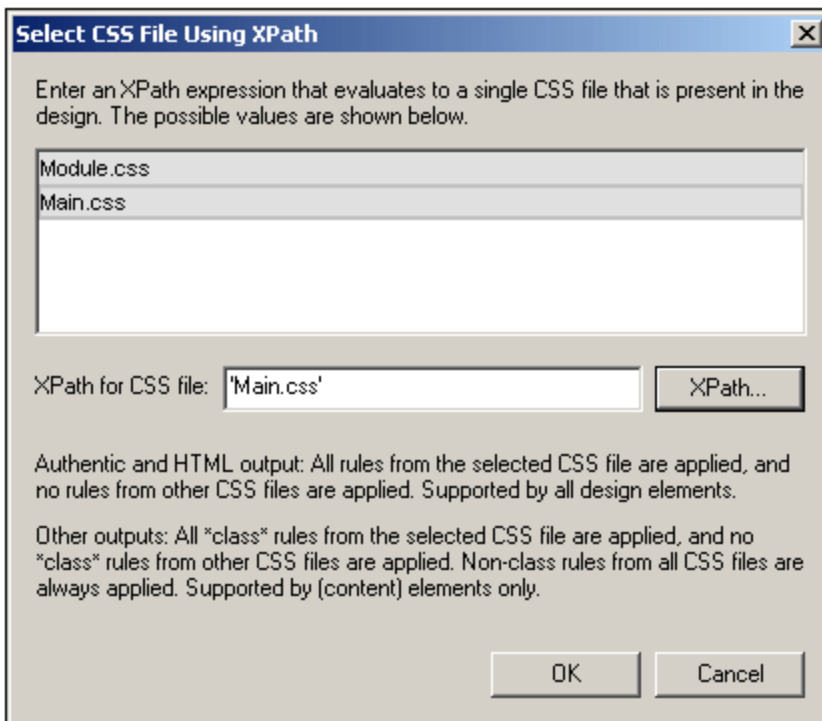
Important: Note that it is the lowermost stylesheet that has the **highest import precedence**, and that the import precedence decreases with each stylesheet higher in the listing order. The order of import precedence in the listing shown above is: (i) `ExternalCSS-3.css`; (ii) `ExternalCSS-2.css`; (iii) `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.



Note the following:

- *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 output, 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 output, these

rules can be applied to the following design components: [Auto-Calculations](#)²⁷⁰, [the \(contents\) placeholder](#)¹²⁵, [paragraph \(block\) components](#)¹²⁷, and [table cells](#)¹⁴³.

- *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 output, only non-class selector rules are applied, with conflicts being resolved on the basis of priority.

8.4.2 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

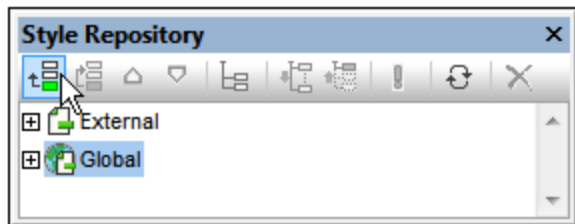
The following [selectors](#) are supported:

- *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`

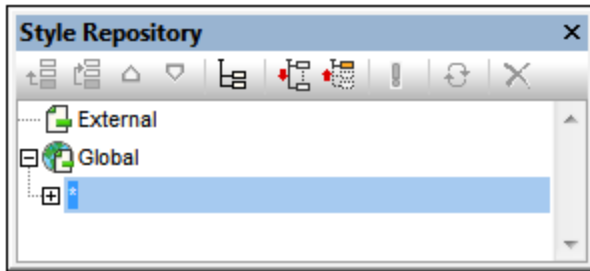
Adding a global style

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](#)³⁷³.
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](#)³⁷³. (The style properties can be displayed in three possible views. These views and how to switch between them are described in [Views of Definitions](#)⁵⁴.)

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>
  <head>
    <style>
      <!--
        h1 { color:blue;
            font-size:16pt;
          }
        h2 { color:blue;
            font-size:14pt;
          }
        .red { color:red;}
        .green { color:green;}
        .green { color:lime;}
      -->
    </style>
  </head>
</html>
```

```

    </style>
  </head>
  <body/>
</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.

8.4.3 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](#)⁷⁰⁴ or in the [Styles](#)⁵⁴ sidebar. (This is as opposed to global styles, which are defined in the [Style Repository](#)⁵¹ sidebar.)

Local styles via the Format toolbar

You can select content in the design and apply local styles via the Format toolbar (*screenshot below*).



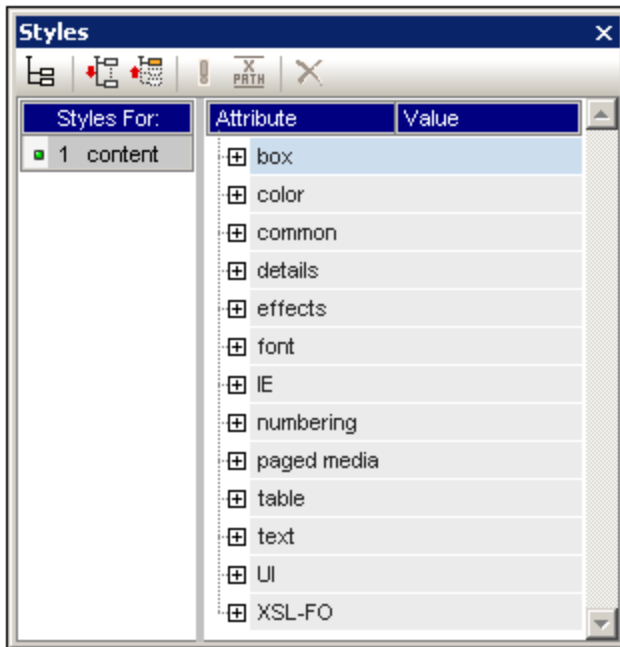
You can apply predefined HTML formatting (such as `div`, `h1`, `pre`, etc), text styling, background color, text alignment, lists, and hyperlinks. See the section, [Format toolbar](#)⁷⁰⁴, for details.

Local styles via the Styles sidebar

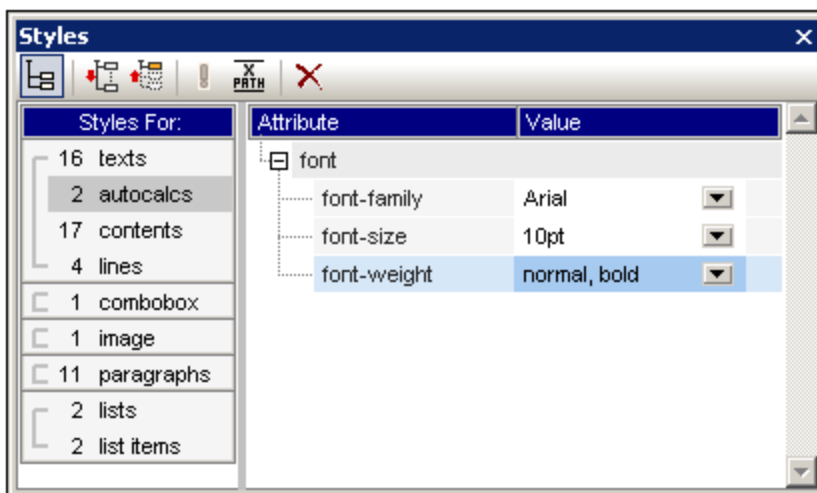
Defining a style locally via the Styles sidebar consists of three parts:

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.



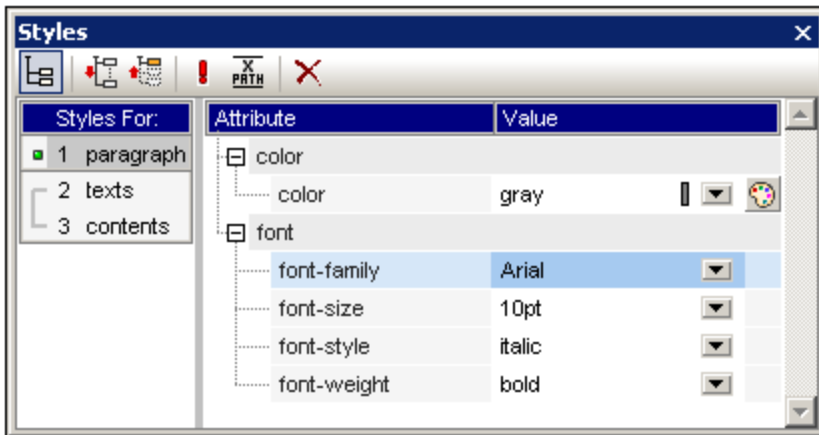
- After making the selection in Design View, you select, in the *Styles-For* column, the **component-type** you wish to style. If there is more than one instance of the component-type you select, then the styles you define will be applied to all these instances. So, for example, if you select the *16-texts* 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](#)⁵⁴. How to do this is described in the section [Setting Style Values](#)³⁷³.

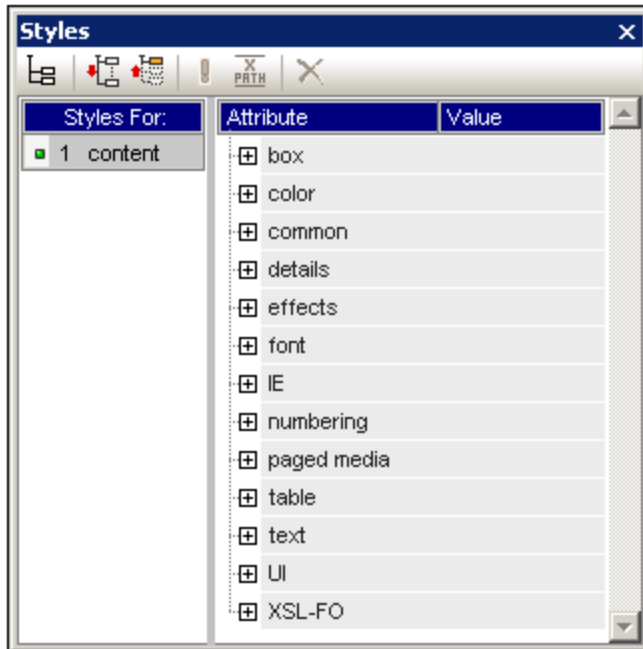
8.4.4 Setting Style Values

For the component-type selected in the *Styles-For* column, style properties are defined in the [Style Definitions pane of the Styles sidebar](#)⁵⁴ (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.

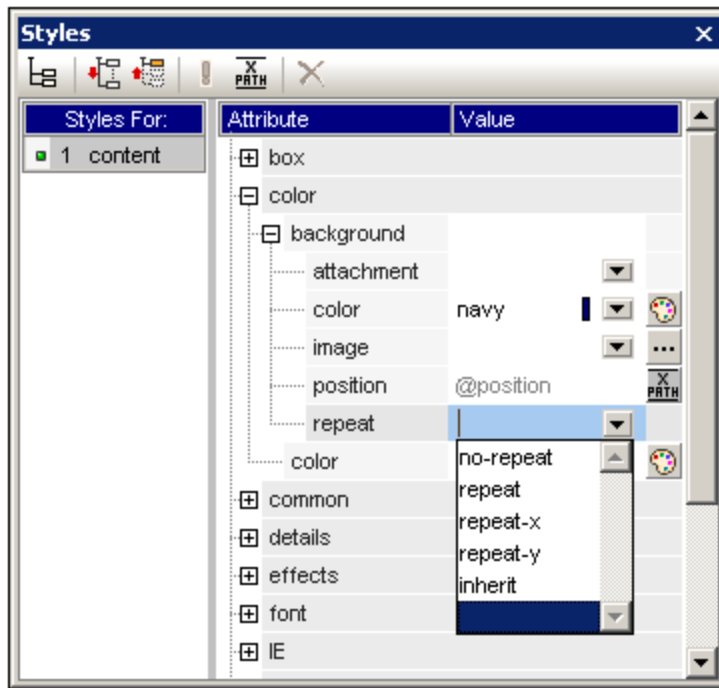


The display of properties can be modified using the [List Non-Empty](#) ⁵⁴, [Expand All](#) ⁵⁴, and [Collapse All](#) ⁵⁴ toolbar buttons ⁵⁴. Each group of style properties can be expanded to access style properties or sub-groups of style properties (see screenshot below).

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](#) ³⁷⁵.

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](#) ³⁷⁴.

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**.

8.4.5 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

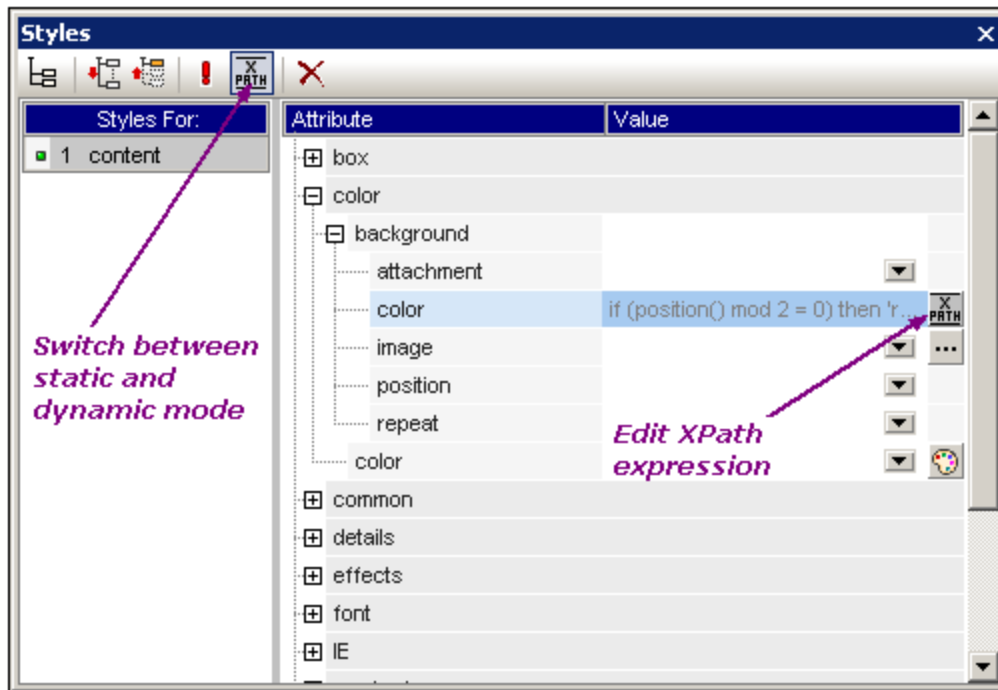
Static mode and dynamic (XPath) mode for property values

For those properties where [XPath expressions are enabled](#)³⁷⁵, 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](#)⁶⁸⁸ 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](#)⁷⁴⁵.) 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](#)⁶⁸⁵. 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](#)⁶⁸⁵ 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](#)⁶⁸⁵ 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.

8.4.6 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](#) ²⁷⁰
- [The \(contents\) placeholder](#) ¹²⁵
- [Paragraph \(block\) design elements](#) ¹²⁷
- [Table cells](#) ¹⁴³

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](#) ³⁸¹, 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:

```
<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:

```
<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

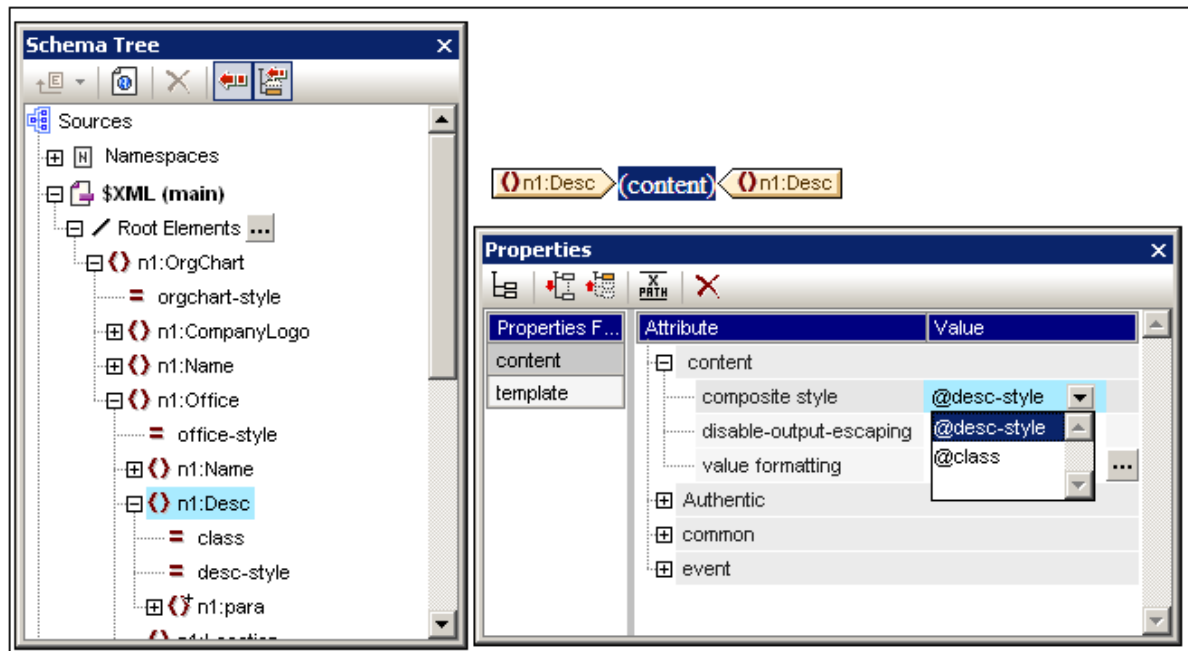
text-decoration

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.



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.

8.5 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](#)³⁸¹: 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](#)³⁸³ feature of Authentic View. See the section [Composite Styles](#)³⁸¹ for a description of how this mechanism works.
- [RichEdit](#)³⁸³: 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](#)³⁸³ for details.
- [Text State Icons](#)³⁸⁷: 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.

8.5.1 Composite Styles

The Composite Styles feature gives Authentic View users the ability to style the entire output of the following design components:

- [Auto-Calculations](#)²⁷⁰
- [The \(contents\) placeholder](#)¹²⁵
- [Paragraph \(block\) design elements](#)¹²⁷
- [Table cells](#)¹⁴³

The mechanism for doing this is explained below.

Note: For a further more general description of Composite Styles, see the section [Composite Styles](#)³⁷⁸. 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:

```
<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:

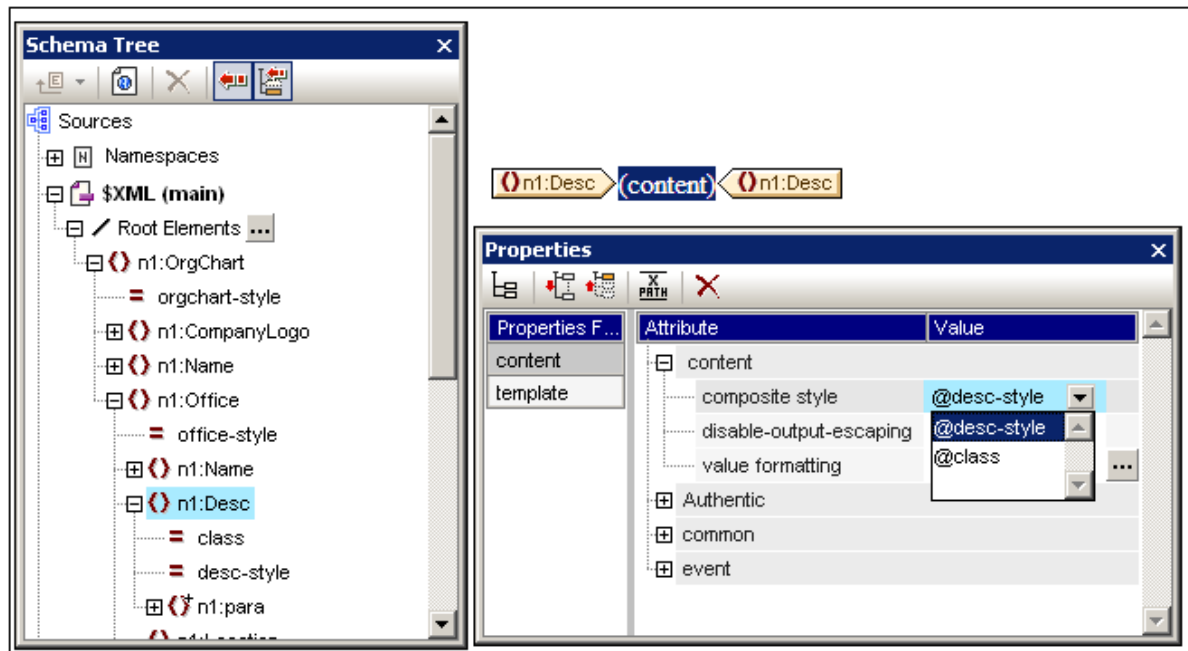
<code>font-family</code>	<code>font-size</code>	<code>font-weight</code>	<code>font-style</code>
<code>color</code>	<code>background-color</code>	<code>text-align</code>	<code>text-decoration</code>

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.

8.5.2 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](#) ³⁸⁴
- [Creating an element in the SPS design as a RichEdit component](#) ³⁸⁴
- [Using RichEdit in Authentic View](#) ³⁸⁶

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:

```
<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:

```
<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:

```
<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.

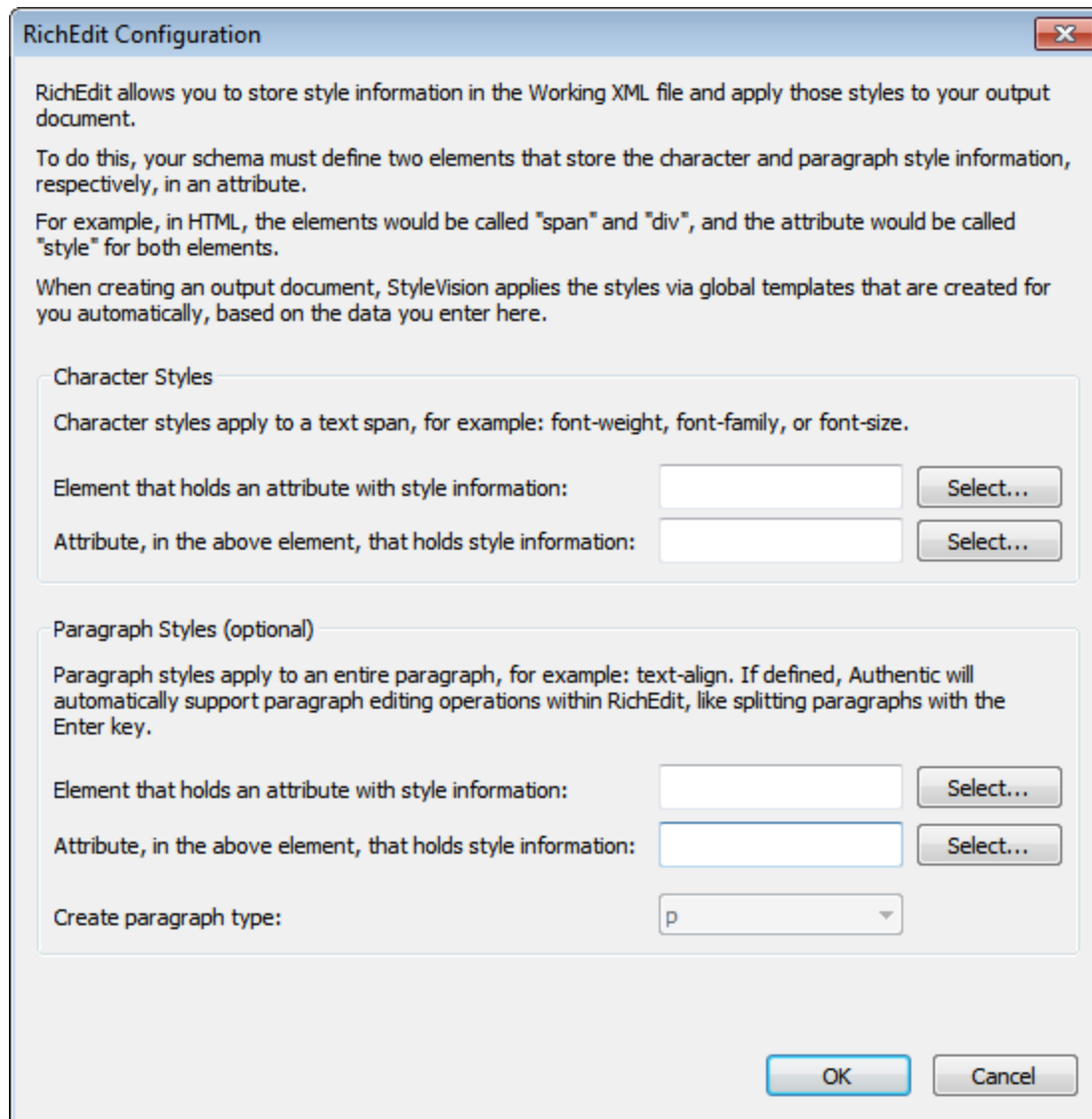
When the text is processed with an XSLT stylesheet, the style properties are passed to the output as markup appropriate to the output format.

RichEdit also enables Authentic View users to apply block-level formatting (such as text-alignment). 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.

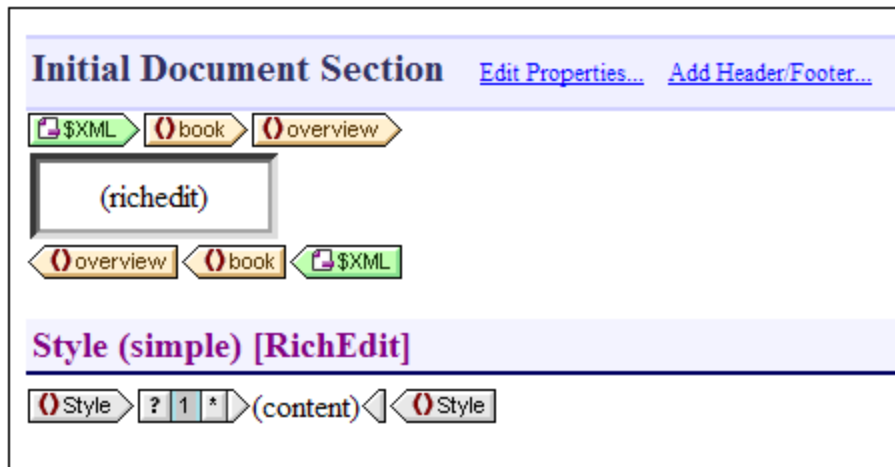
Creating an element as a RichEdit component

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.

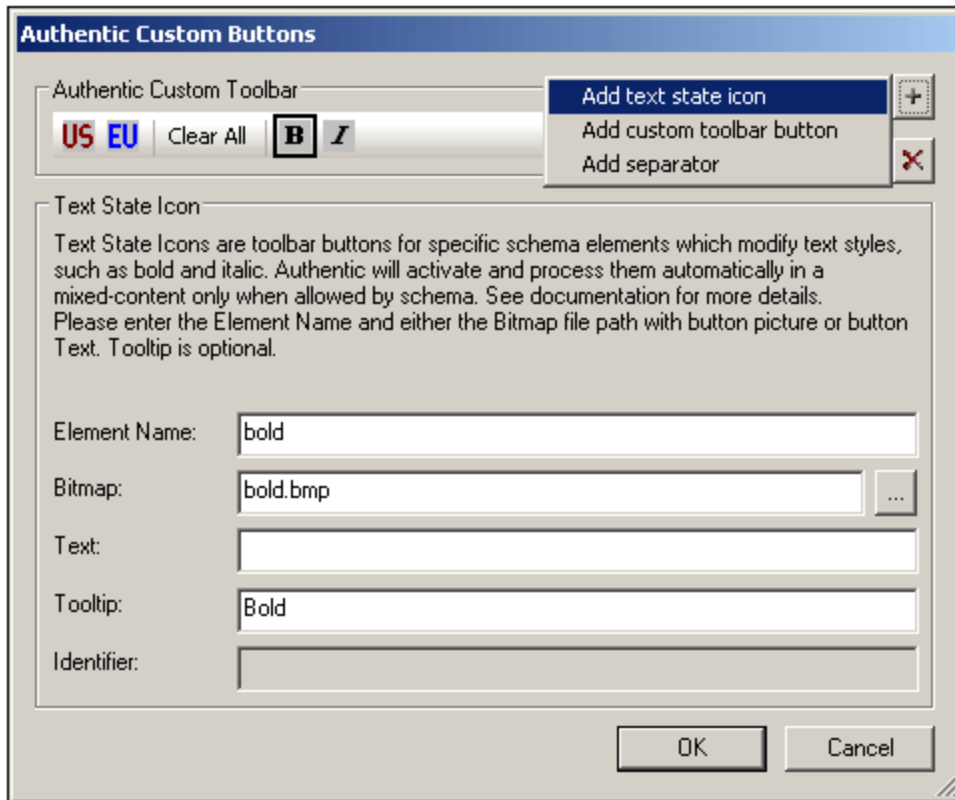
8.5.3 Text State Icons

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

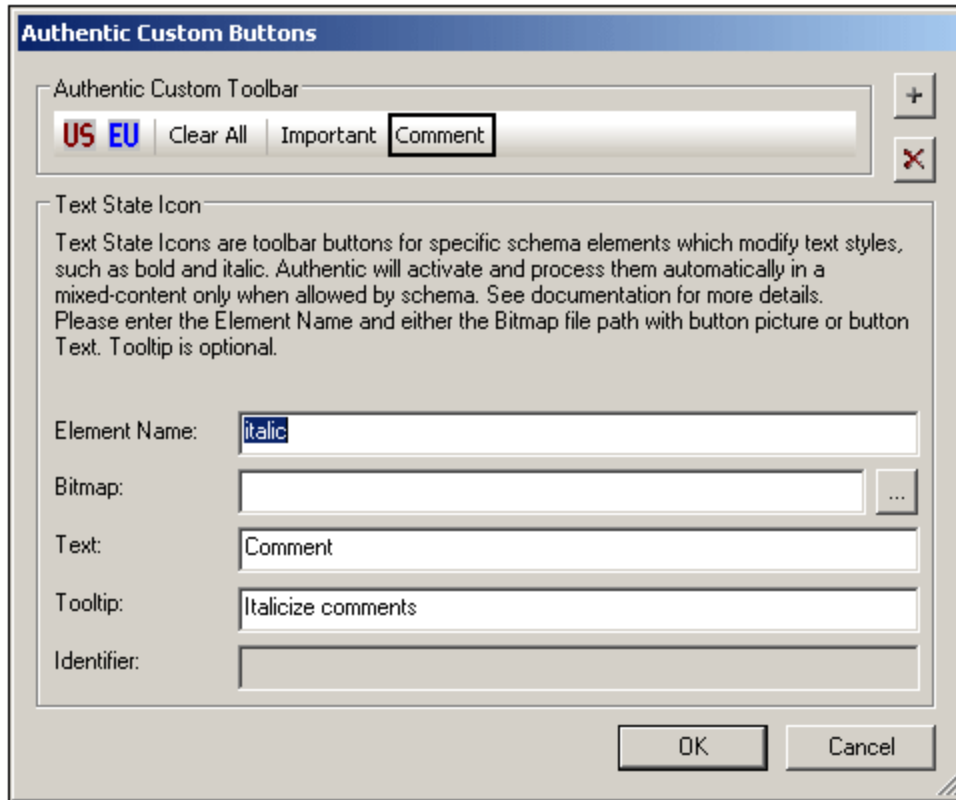
To create a Text State Icon for the Authentic toolbar, first select the command **Authentic | Custom Toolbar Buttons**, then click the **Add** button at the top right of the Authentic Custom Buttons dialog (see *screenshot below*) and select **Add Text State Icon**.



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.

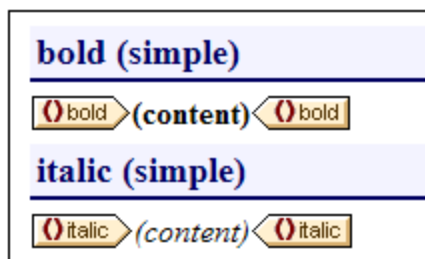
In the example above, we have used familiar bitmaps as the Text State Icons to mark up text with the `bold` and `italic` elements. If you wish, you could use any bitmap or button text that you think might be easier for the Authentic View user to relate to. For example, important text fragments could 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.

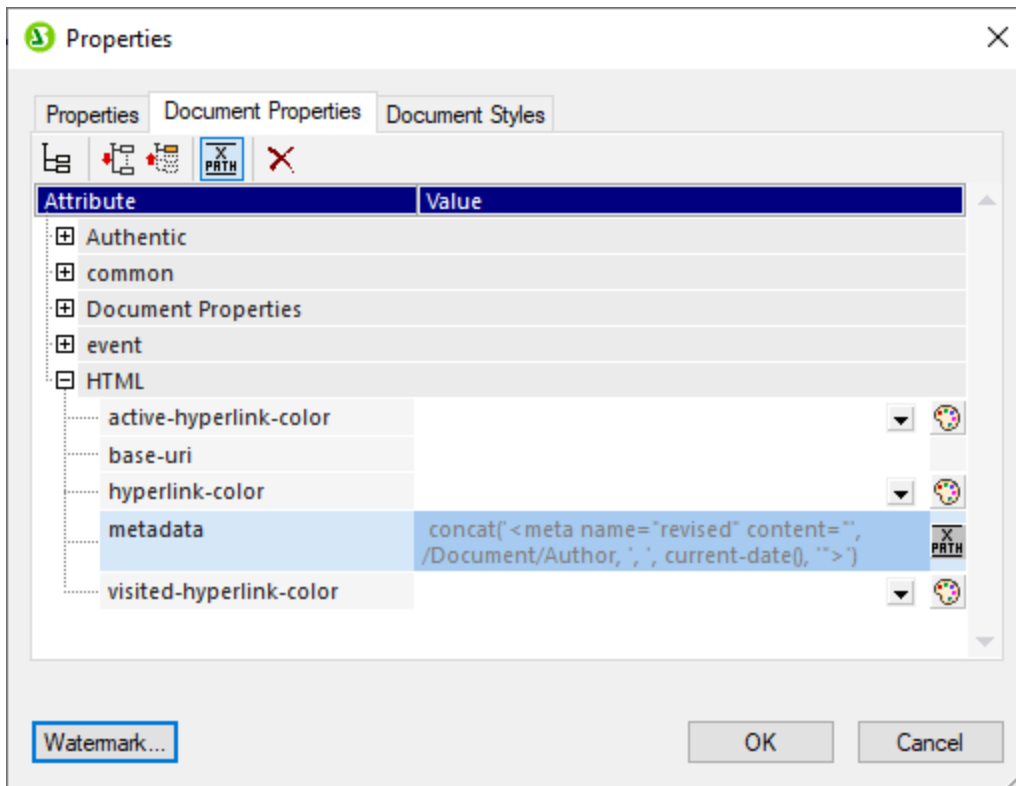
Consequently, whenever the Authentic View user creates these elements around text fragments, the respective global template (with its text styles) is applied to that text fragment.

8.6 HTML Document Properties

Properties of the output HTML document can be specified either in the *Document Properties* tab in the Properties dialog of the [Initial Document Section](#)³⁹⁵ or in Properties View when the [Main Template in the Design Tree window](#)⁴⁸ is selected.

Via properties of the Initial Document Section

Click the *Edit Properties* hyperlink in the [Initial Document Section](#)³⁹⁵ title bar and then select the *Document Properties* tab.

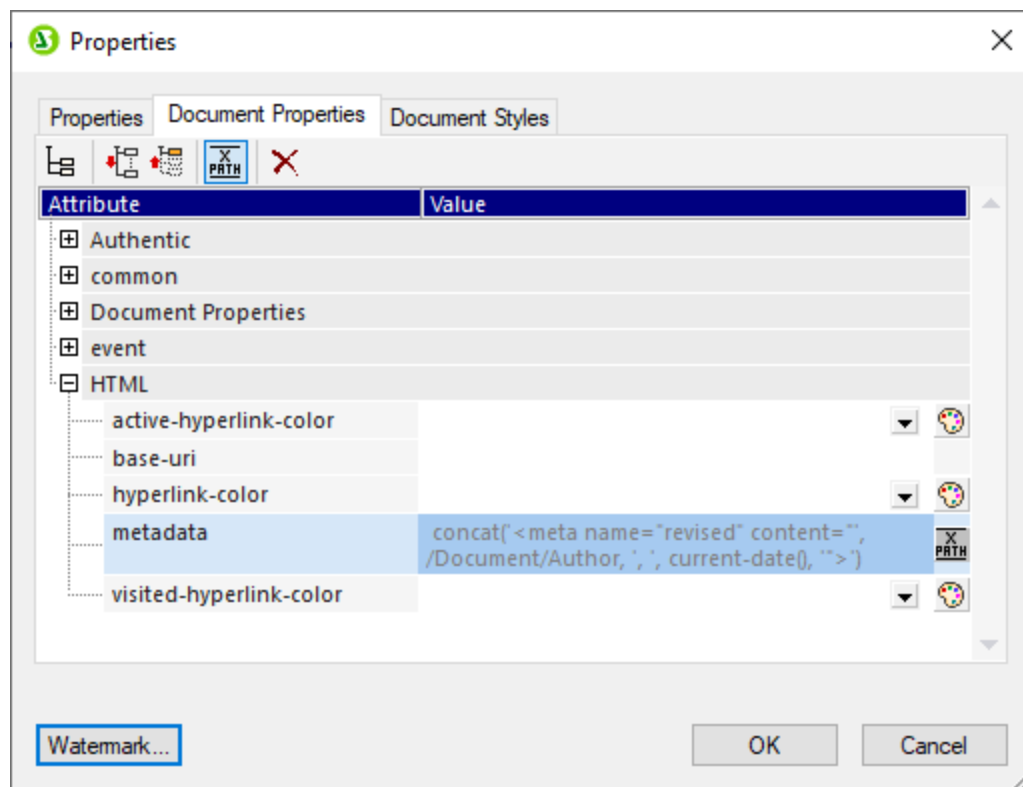


Here you can set various properties of the HTML output document that will be generated. Note the following points:

- These are HTML properties that will be applied at the document level, for example, the `class` and `id` properties of the *Common* properties section.
- The *Document Properties* and *HTML* sections contain generic properties that relate to the HTML document as a whole, such as link colors, and information that goes into the `meta` tags of the HTML document.
- The `metadata` property of the *HTML* section enables you to enter any text that you want to go into the `HEAD` element of the HTML document (see screenshots above and below). The text you enter could, for example, be a `script` or `meta` element, or several such elements. You can enter these HTML elements directly as text (without quotes) or, as in the screenshots, as an XPath expression. In the screenshot example, the XPath expression sets a meta tag for the revision date of a document.

Via the properties of the Main Template

In the [Design Tree](#) ⁴⁸ sidebar, select the Main Template. In the Properties sidebar (*screenshot below*), you can now set the properties of the output HTML document.



The properties in this dialog are exactly the same as those in the *Document Properties* tab described above.

8.7 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](#)⁵⁵. 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: Since page layouts cannot be applied to plain text output, the features for designing print output that are described in this section will not be applied to Text output.

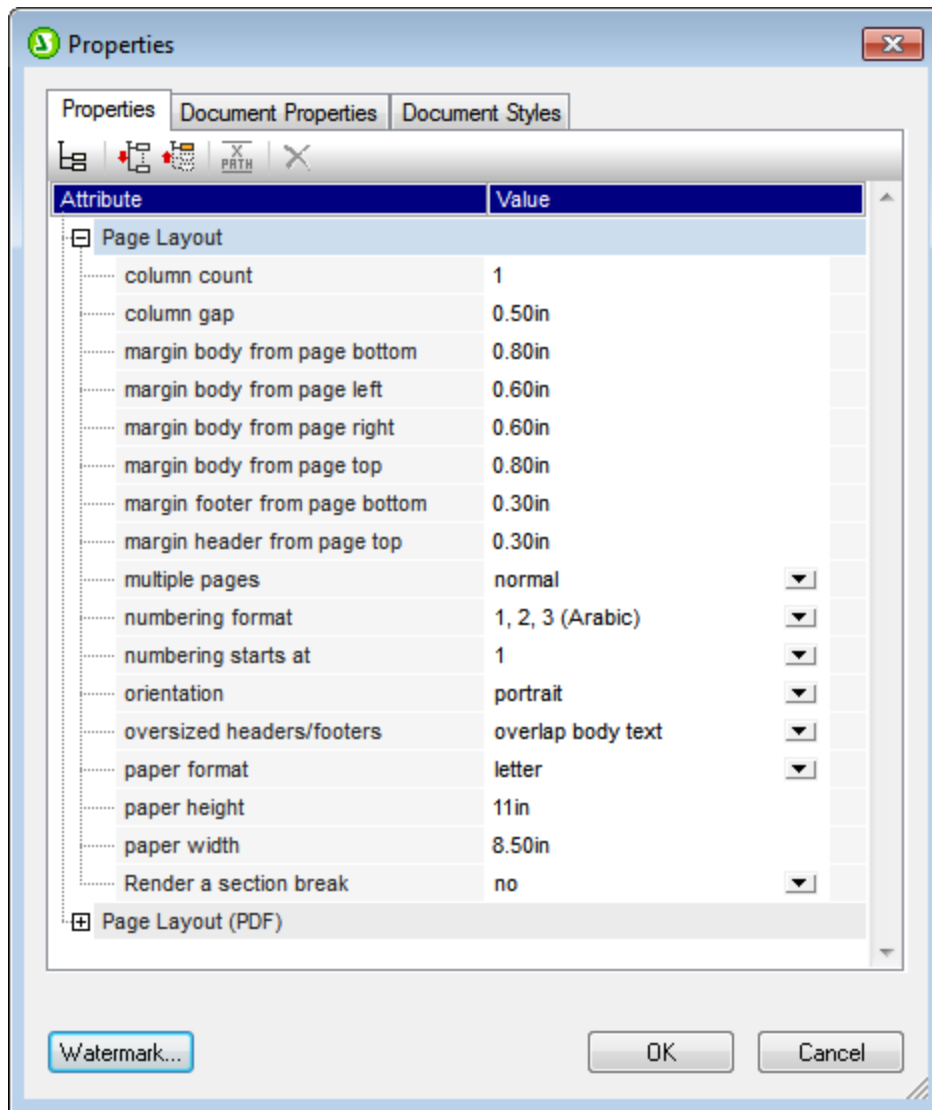
Properties sidebar

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*).

Document Section

[Edit Properties...](#) [Add Header/Footer...](#) [Hide Headers/Footers](#)

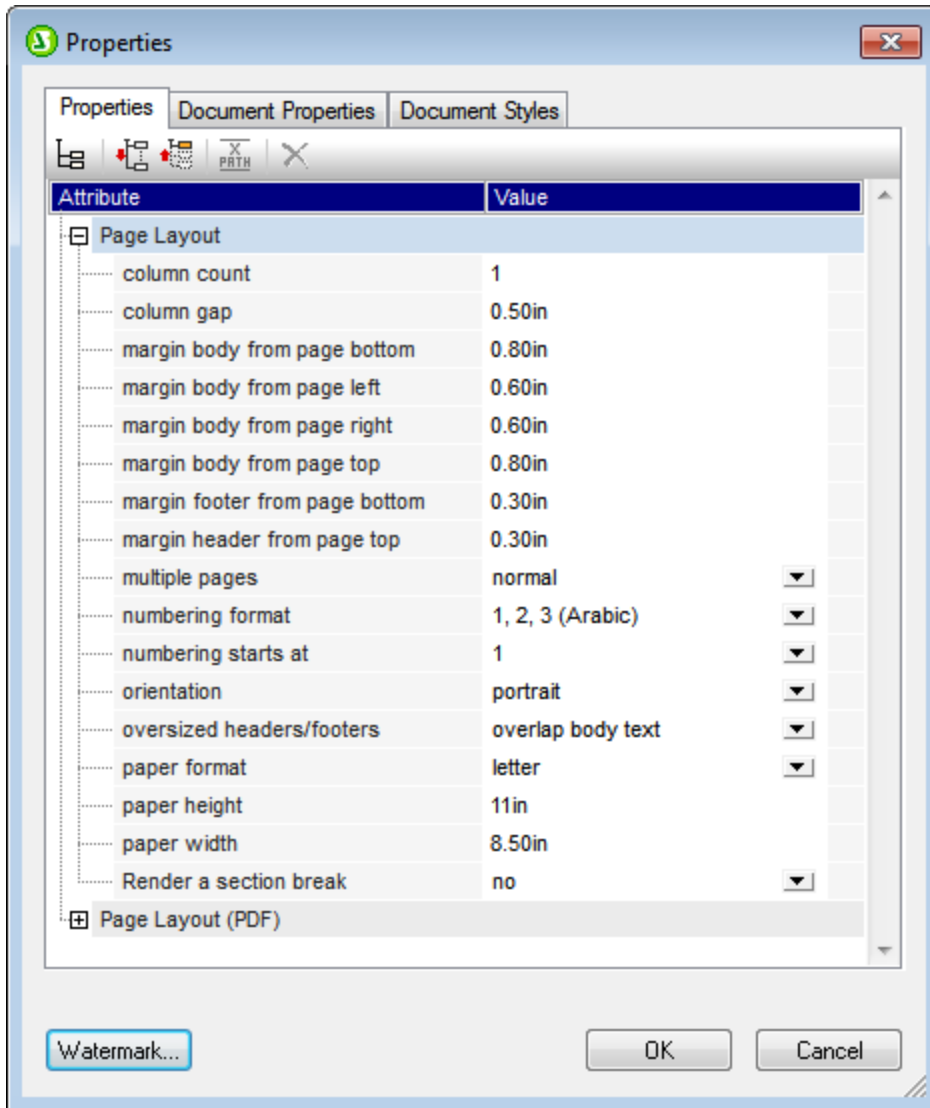
Clicking the Edit Properties link pops up the Properties window, with the Page Layout properties active within it (*screenshot below*).



How to set the values of these properties is discussed in the section, [Page Properties](#)³⁹⁸.

8.7.1 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](#)³⁹⁸ (see *screenshot below*) can be defined. Additionally, each document section can also have different [headers and footers](#)⁴⁰³.



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](#)³⁹⁵.

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.

Document Section

[Edit Properties...](#)
[Add Header/Footer...](#)
[Hide Headers/Footers](#)

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](#)³⁹⁸ for the document section can be edited via the *Edit Properties* hyperlink of the Document Section. If required, separate [headers and footers](#)⁴⁰³ can be added for the document section (via the *Add Header/Footer* hyperlink). How to define [page layout properties](#)³⁹⁸ and [headers and footers](#)⁴⁰³ 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 RTF output generated by XSLT 1.0 SPSs, only document sections that are immediate children of the Main Template are allowed. This restriction does not apply to 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

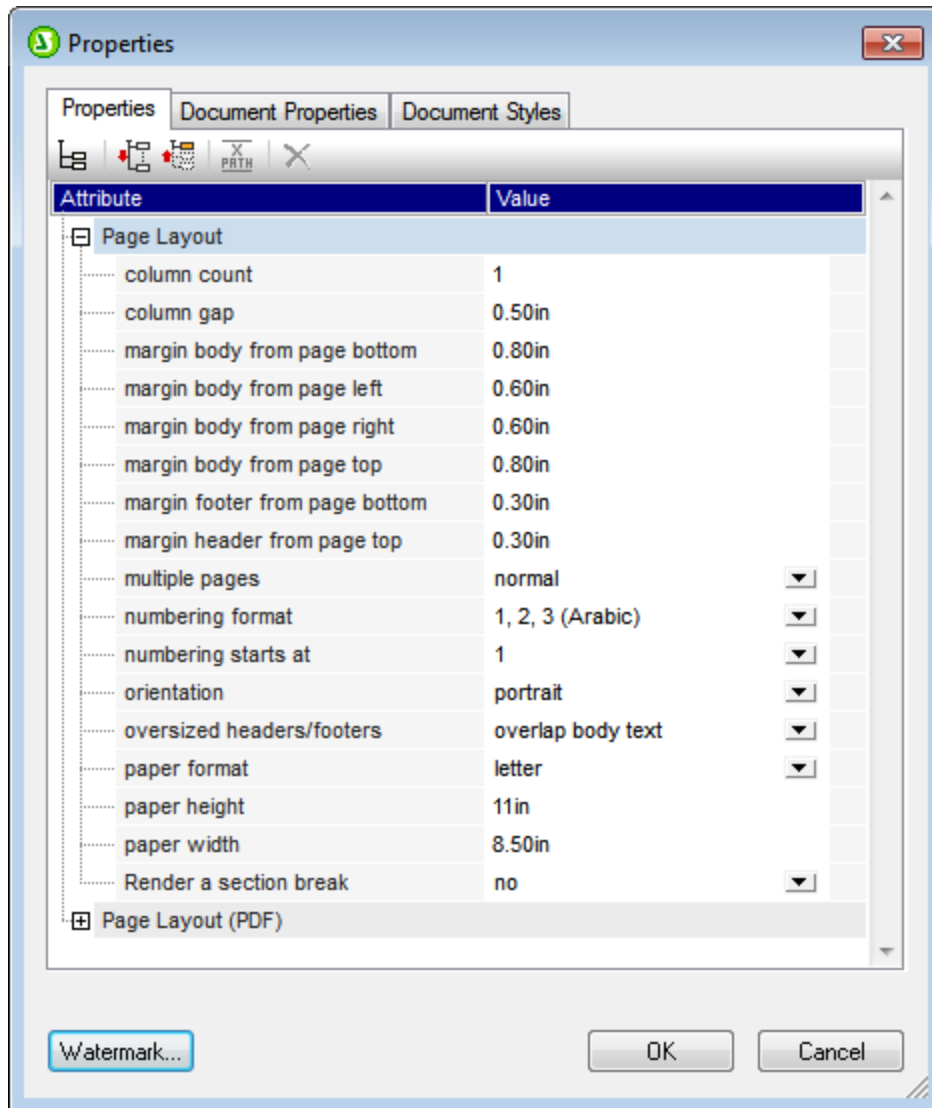
To delete a document section, in the title bar of the document section, right-click the words *Document Section*, and in the menu that pops up select the command **Edit | Delete**. The document section will be deleted, and this will be indicated by the deletion of the title bar. By deleting the document 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.

8.7.1.1 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](#)³⁹⁷ are also created in Initial Document Sections.

Initial Document Section [Edit Properties...](#) [Add Header/Footer...](#)

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](#)³⁹⁸. The **Watermarks** button brings up the Watermarks dialog in which you can [define a watermark](#)⁴¹¹ for the section's pages. A different watermark can be defined for each section.

Document properties

Properties of the output HTML document are specified in the [Document Properties tab](#)³⁹⁰.

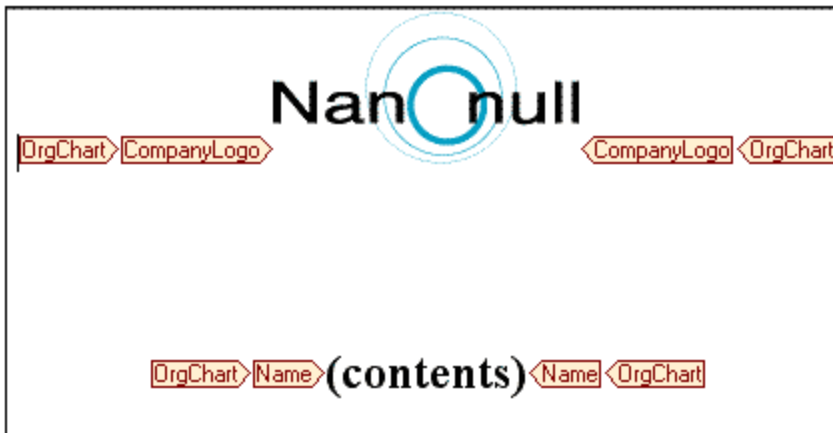
Document styles

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.

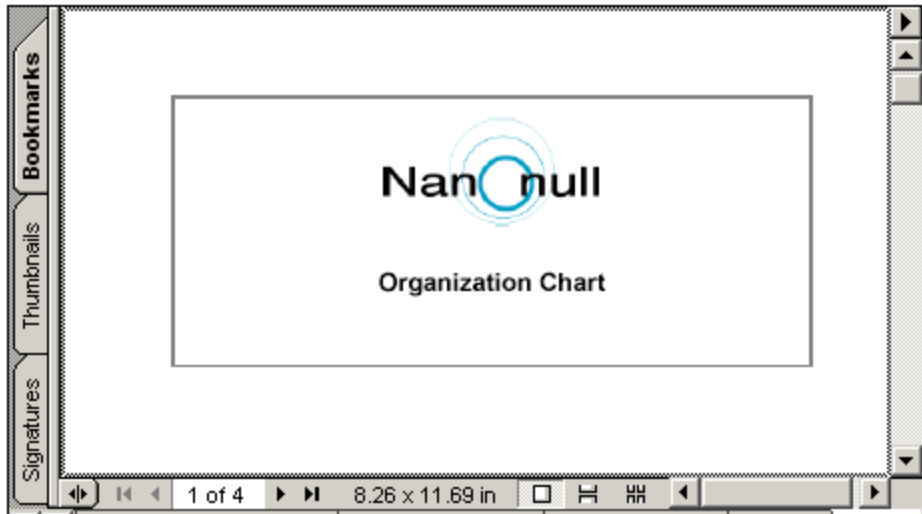
Cover pages

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 tab to see the result in the preview window.



8.7.1.2 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.

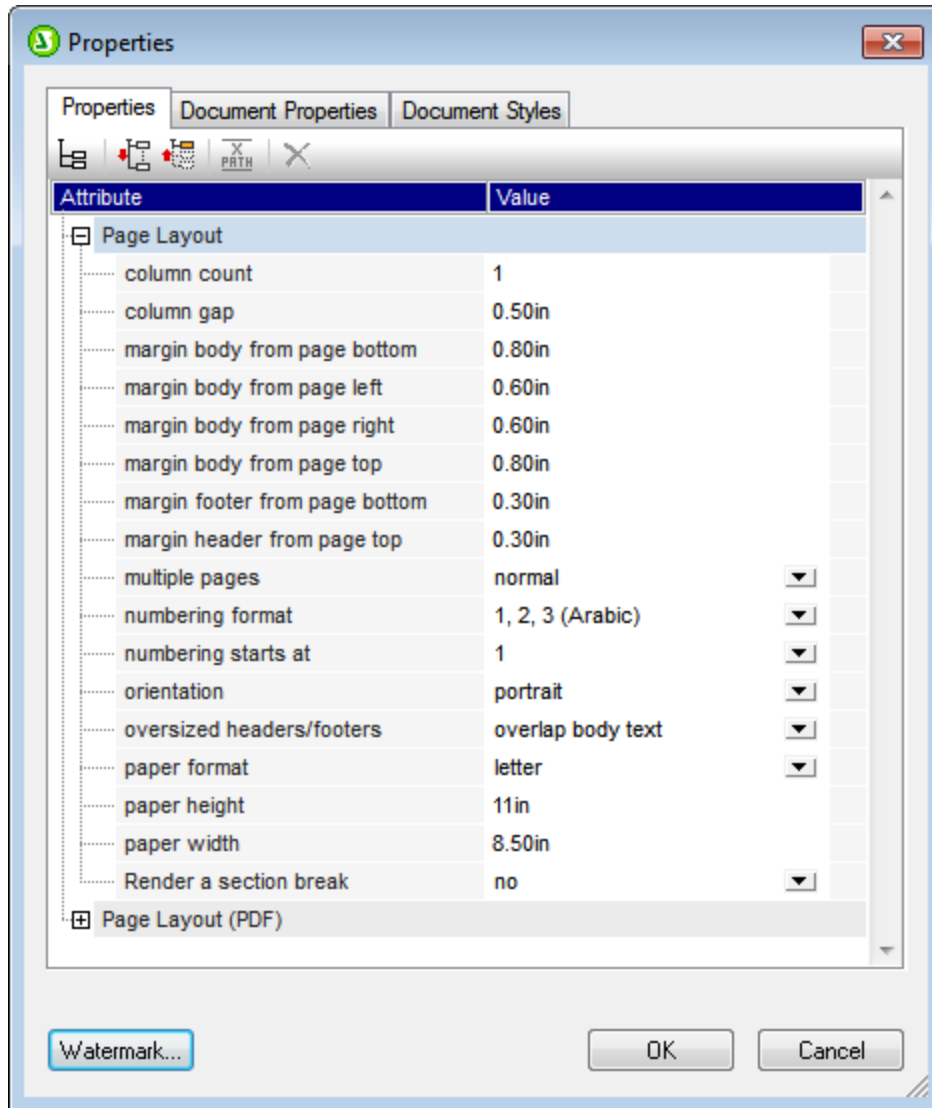
Note: Due to a limitation of FOP, Page Layout properties are available only for the Initial Document Section when FOP is used to generate output.

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](#)³⁹³ 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 (in), centimeters (cm), millimeters (mm), picas (pc), points (pt), [pixels \(px\)](#)⁴⁰⁹, 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.

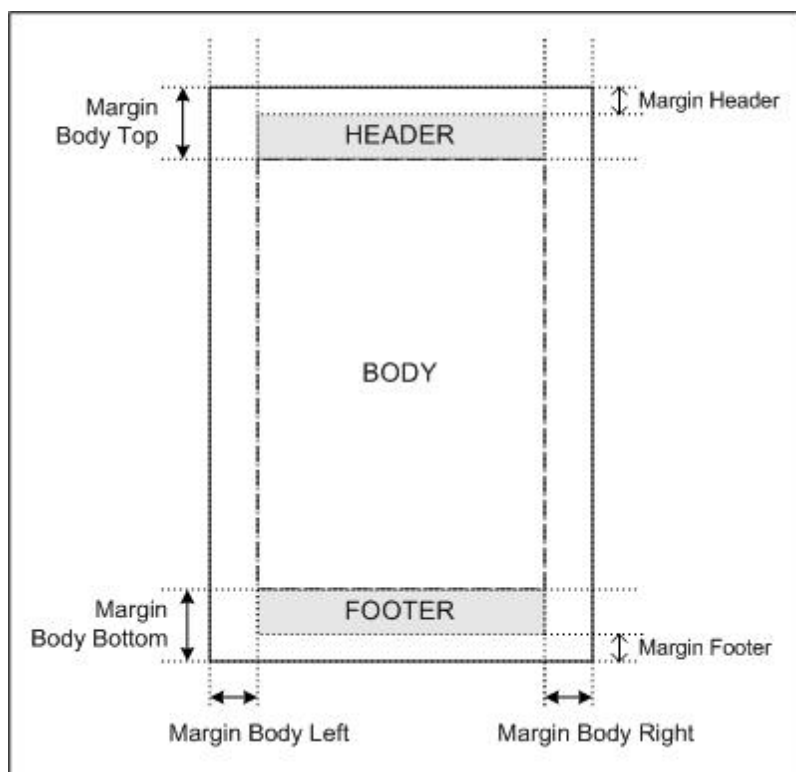
Page margins

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 \(px\)](#)⁴⁰⁹, 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).



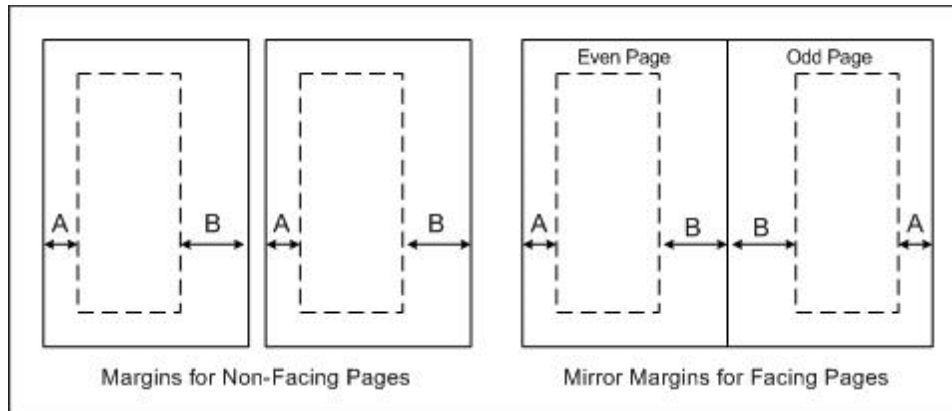
The vertical extents of headers and footers are determined by the actual content of the headers and footers. You should ensure that the vertical extent of a header plus the header margin does not exceed the *Margin body from page top*. Otherwise, the header will be too large to be contained in the space 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 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.

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.



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](#)⁷⁸⁵, 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](#)⁷⁸⁵ 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).

8.7.1.3 Headers and Footers: Part 1

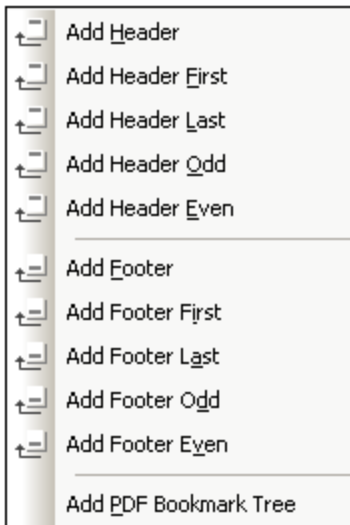
Headers and footers can be added **for each document section** of the document, including the Initial Document Section.

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](#)³⁹³ (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).

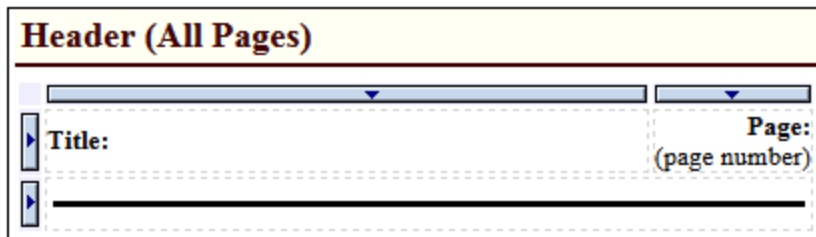


On clicking the required header or footer, a template for the header or footer is created within that document section in [Design View](#)³³. The Design View display of header/s and footer/s in a document section can be toggled on and off by clicking the *Hide Headers/Footers* link in the [Initial Document Section or Document Section item](#)³⁹³ (see screenshot above).

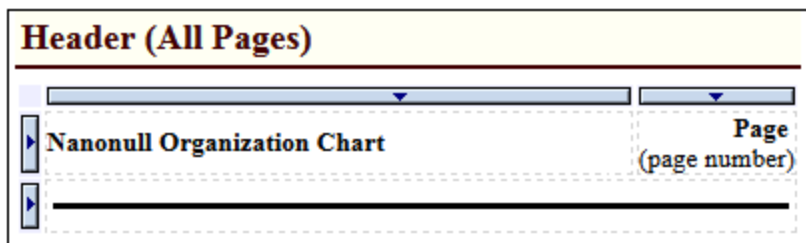
Note: Headers and footers for the last page are supported for PDF output only.

Designing the header/footer in Design View

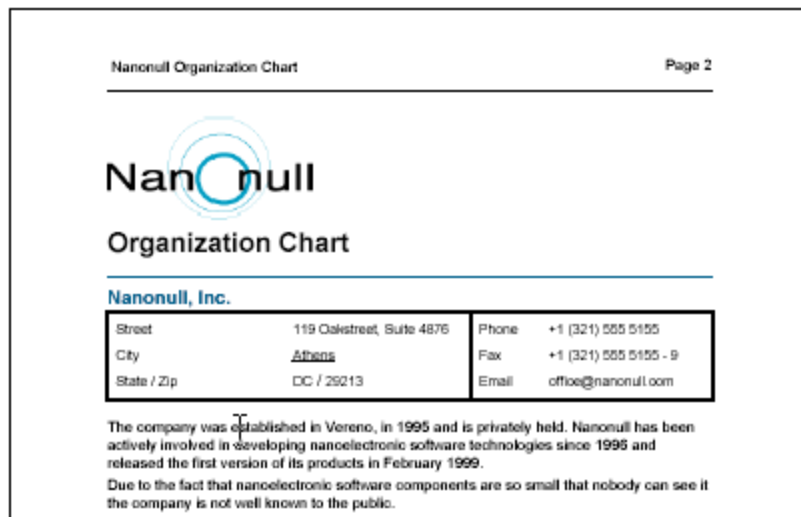
The header/footer template is designed just like any other template. Components can be dragged from the schema tree or entered statically, and then styled. An example is shown below. When a header is added, the template will look something like this:



Change the header as required. Note that you can use both static and dynamic content, and even images.



Click the Preview RTF 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](#)⁴⁰².



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](#) ⁴⁰⁰ 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](#) ⁴⁰⁵ for more information.)
- Page numbering in a document section starts with the number you specify in the [Page Layout Properties](#) ⁴⁰⁰.

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**.

8.7.1.4 Headers and Footers: Part 2

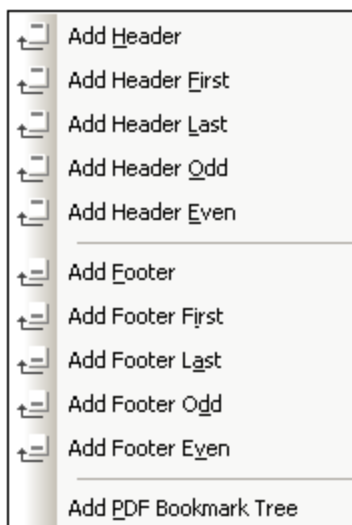
In this section, we describe how to create the following types of headers and footers:

- [Different headers/footers for odd-numbered and even-numbered pages](#) ⁴⁰⁵
- [Different headers/footers for different document sections](#) ⁴⁰⁶
- [Simulating headers/footers inside a page](#) ⁴⁰⁶
- [Headers/footers with subtotals](#) ⁴⁰⁶

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](#)³⁹³. 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](#)⁴⁰².
- The [Page Total](#)⁴⁰² 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](#)¹⁸⁷. The approach would be to design a single page as a layout container. The header and footer are created within [static tables](#)¹⁴⁵ 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](#)⁷⁸⁵ (**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](#)²⁹, `C:\Documents and Settings\\My Documents\Altova\StyleVision2024\StyleVisionExamples\Tutorial\Subtotals\`, demonstrates how running totals can be created and included in headers and footers.

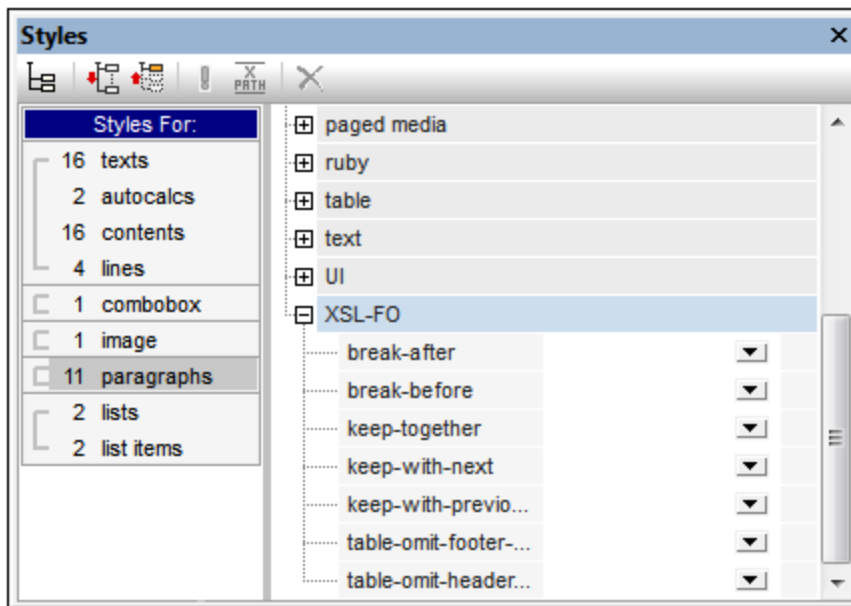
The following strategy was used to design this SPS:

- 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.

8.7.2 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*).

8.7.3 Footnotes

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

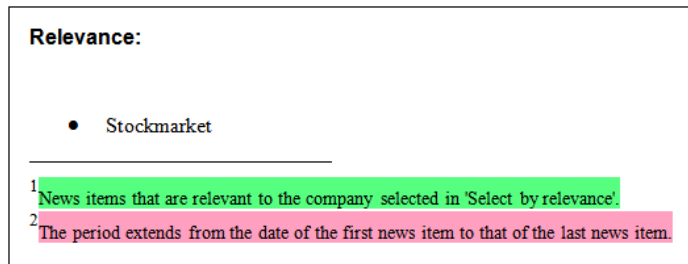
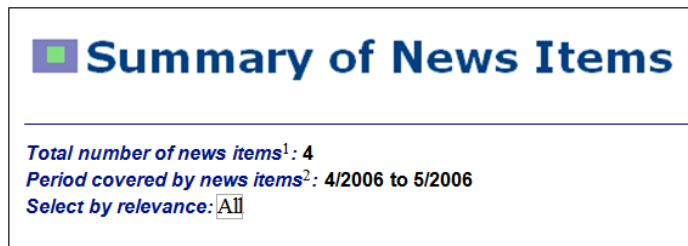
text can be formatted.

- 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.

8.7.4 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.

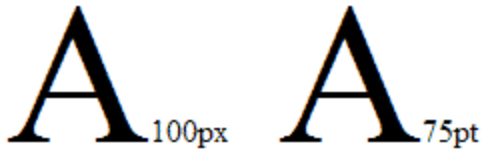
Resolution	Pixels	Points	Factor
72 dpi	100	100	1.00
96 dpi	100	75	0.75
120 dpi	100	60	0.60

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

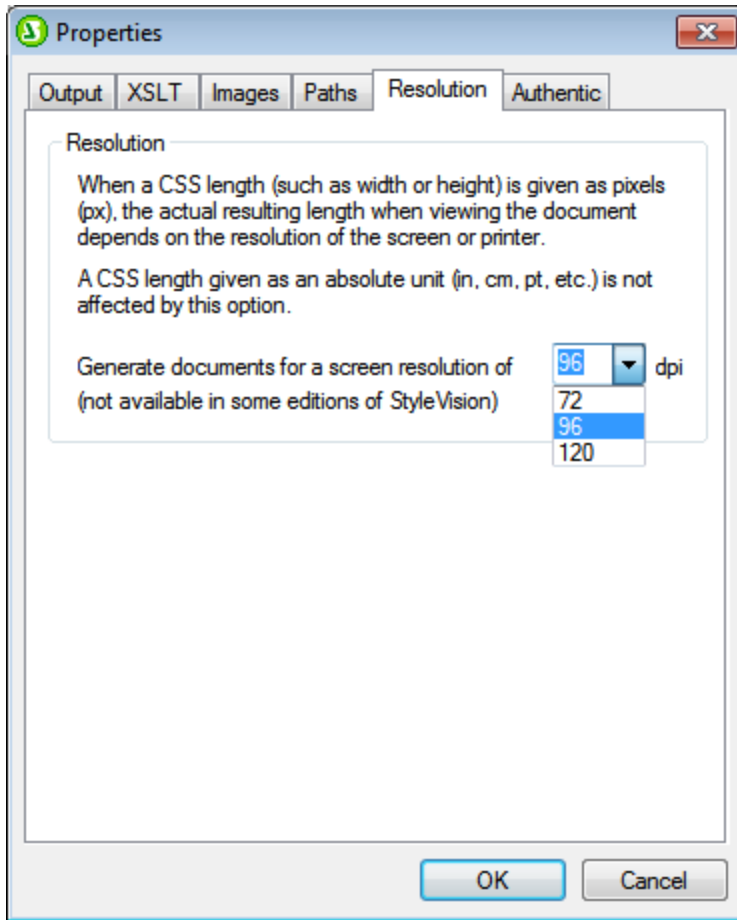
When an SPS is transformed into a print format, such as PDF, RTF, or Word 2007+, non-absolute pixel lengths must be converted to absolute lengths, such as points or inches (because lengths on paper cannot be defined in terms of pixels). The question is: What factor (or screen resolution) should be used 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.

8.7.5 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.

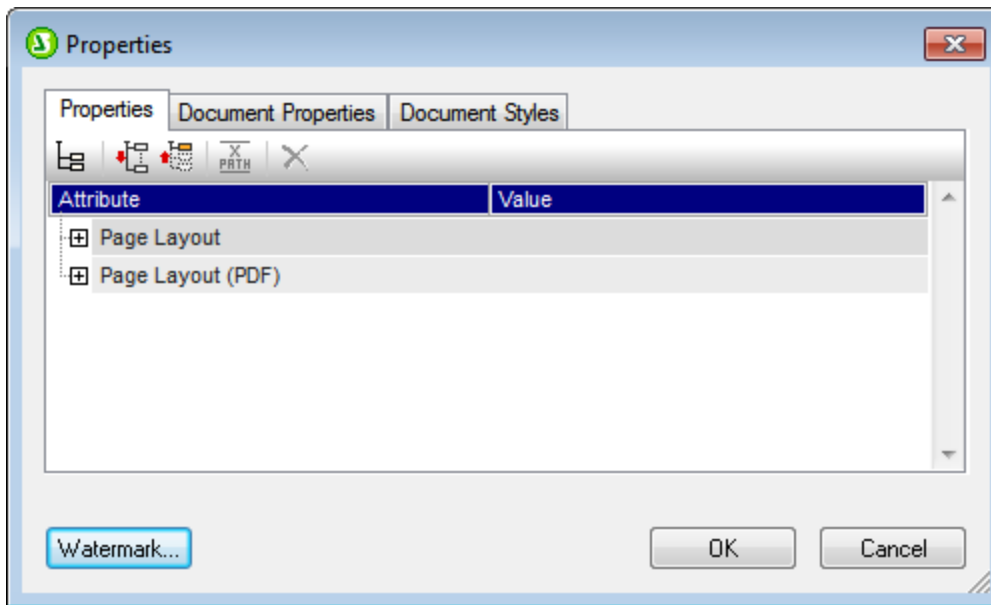
Note: Watermarks are available only for print-output media, not for HTML or Authentic View.

Creating a watermark

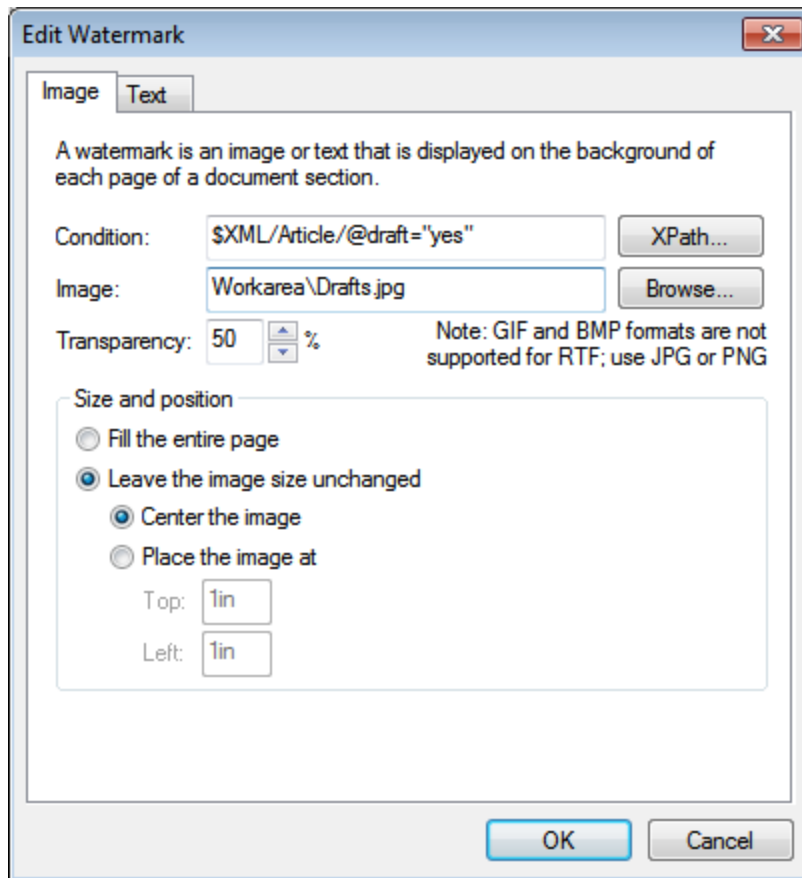
A watermark can be created separately for each document section. To create (or to edit) a watermark, click the *Edit Properties* link of the Initial Section or Document Section items in the design (*screenshot below*).

Document Section [Edit Properties...](#) [Add Header/Footer...](#) [Hide Headers/Footers](#)

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.)



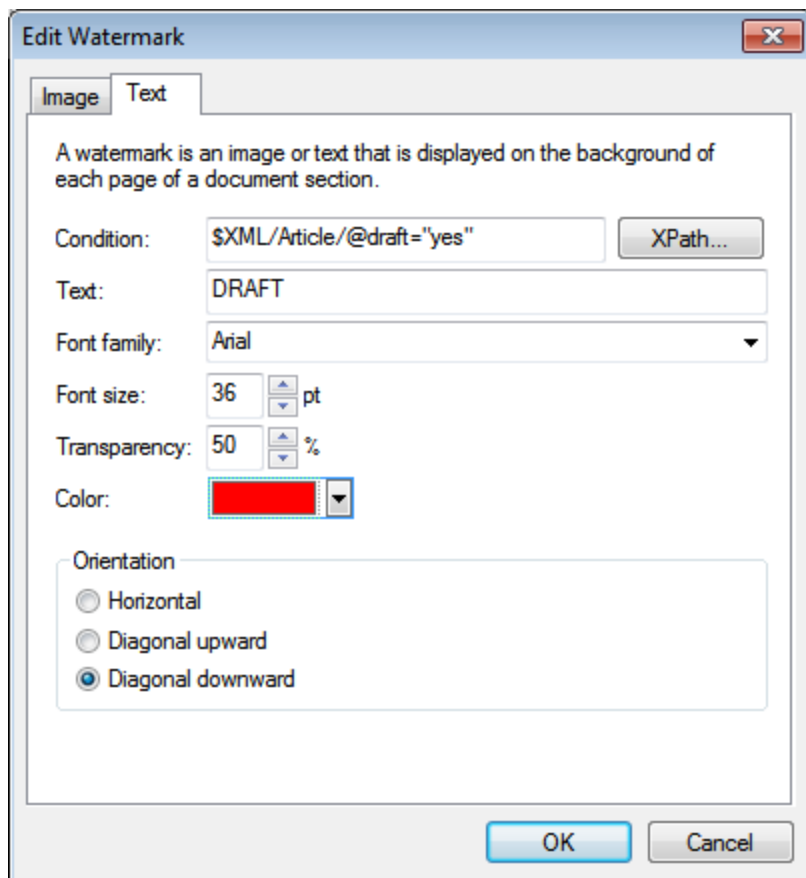
Parameters for image watermarks

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.

9 Additional Functionality

Additional to the [content editing](#)¹²⁴, [structure](#)²⁰⁰, [advanced](#)²⁶⁹, and [presentation](#)³⁴⁹ 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.

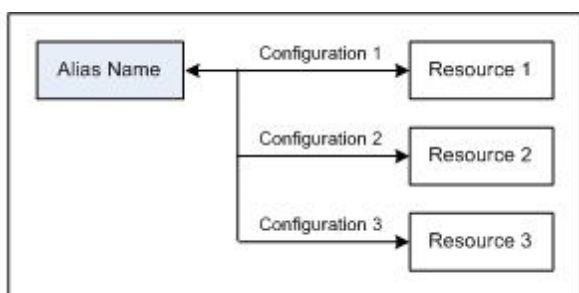
- [Global Resources](#)⁴¹⁶. Global resources provide flexibility in selecting resources. For example, multiple resources (such as files and databases), can be assigned to an alias. When an alias is used as a source (XML, XSD, etc) of an SPS, the resource can be switched among the multiple resources assigned to the alias.
- [Authentic Node Properties](#)⁴³³. 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](#)⁴³⁵. 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](#)⁴³⁸. 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](#)⁴⁶¹. 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](#)⁴⁴⁰. 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](#)⁴⁶⁷. 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](#)⁴⁷². An HTML file can be imported into StyleVision and an XML, XSD, and SPS files can be created from it.
- [New from XSLT](#)⁴⁴². 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](#)⁵⁵
- [Authentic View](#)⁶¹³

9.1 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](#)⁴¹⁶: Resources are defined and the definitions are stored in an XML file. These resources can be shared across multiple Altova applications.
- [Using Global Resources](#)⁴²⁷: 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.

9.1.1 Defining Global Resources

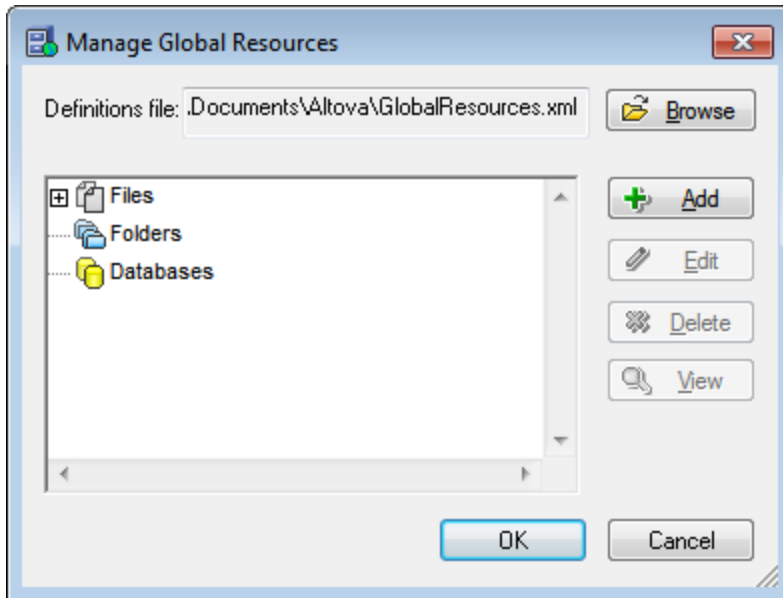
Altova Global Resources are defined in the Manage Global Resources dialog, which can be accessed in two ways:

- 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\\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](#)⁴¹⁹, [folders](#)⁴²⁴, and [databases](#)⁴¹⁶ 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.

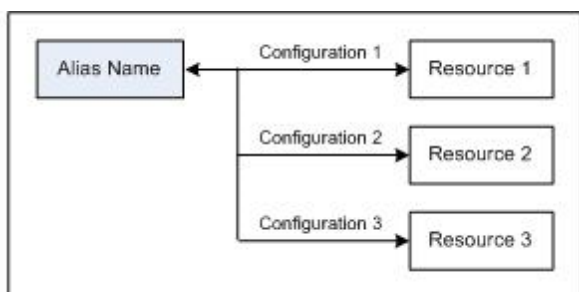
To **edit a global resource**, select it and click **Edit**. This pops up the relevant Global Resource dialog, in which you can make the necessary changes (*see the descriptions of [files](#)⁴¹⁹, [folders](#)⁴²⁴, and [databases](#)⁴²⁵ in the sub-sections of this section*).

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.

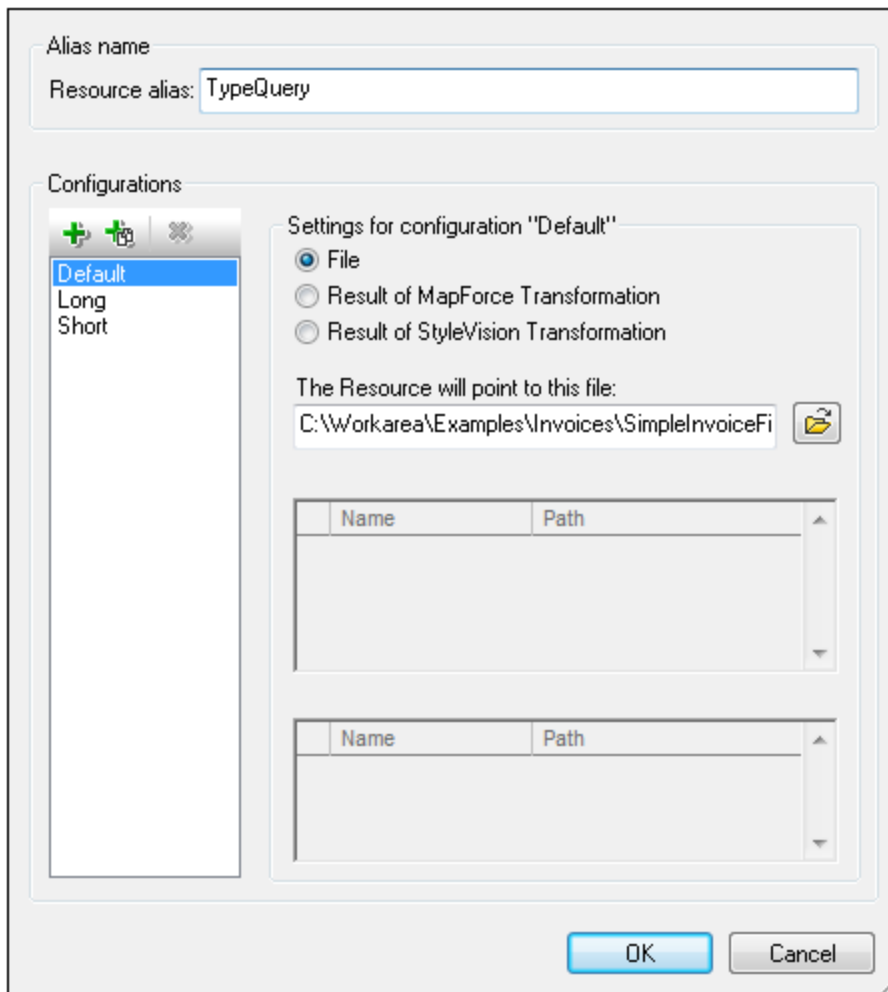
9.1.1.1 Files

The Global Resource dialog for Files (*screenshot below*) is accessed via the **Add | File** command in the [Manage Global Resources dialog](#)⁴¹⁶. 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 | File** command in the [Manage Global Resources dialog](#)⁴¹⁶).

Global Resource dialog

An alias is defined in the Global Resource dialog (*screenshot below*).



Global Resource dialog icons



Add Configuration: Pops up the Add Configuration dialog in which you enter the name of the configuration to be added.



Add Configuration as Copy: Pops up the Add Configuration dialog in which you can enter the name of the configuration to be created as a copy of the selected configuration.



Delete: Deletes the selected configuration.

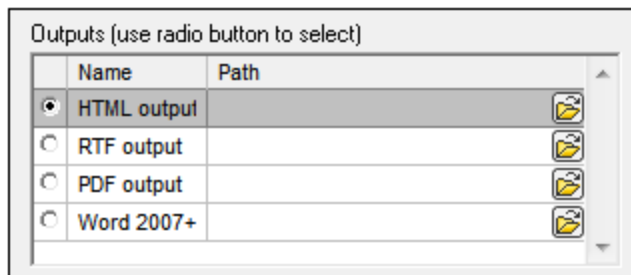



Open: Browse for the file to be created as the global resource.

Defining the alias

Define the alias (its name and configurations) as follows:

1. **Give the alias a name:** Enter the alias name in the *Resource Alias* text box.
2. **Add configurations:** The Configurations pane will have, by default, a configuration named `Default` (see screenshot above), which cannot be deleted or renamed. You can add as many additional configurations as you like by: (i) clicking the **Add Configuration** or **Add Configuration as Copy** icons, and (ii) giving the configuration a name in the dialog that pops up. Each added configuration will be shown in the Configurations list. In the screenshot above, two additional configurations, named `Long` and `Short`, have been added to the Configurations list. The Add Configuration as Copy command enables you to copy the selected configuration and then modify it.
3. **Select a resource type for each configuration:** Select a configuration from the Configurations list, and, in the *Settings for Configuration* pane, specify a resource for the configuration: (i) File, (ii) Output of an Altova MapForce transformation, or (iii) Output of an Altova StyleVision transformation. Select the appropriate radio button. If a MapForce or StyleVision transformation option is selected, then a transformation is carried out by MapForce or StyleVision using, respectively, the `.mfd` or `.sps` file and the respective input file. The result of the transformation will be the resource.
4. **Select a file for the resource type:** If the resource is a directly selected file, browse for the file in the *Resource File Selection* text box. If the resource is the result of a transformation, in the *File Selection* text box, browse for the `.mfd` file (for MapForce transformations) or the `.sps` file (for StyleVision transformations). Where multiple inputs or outputs for the transformation are possible, a selection of the options will be presented. For example, the output options of a StyleVision transformation are displayed according to what edition of StyleVision is installed (*the screenshot below shows the outputs for Enterprise Edition*).



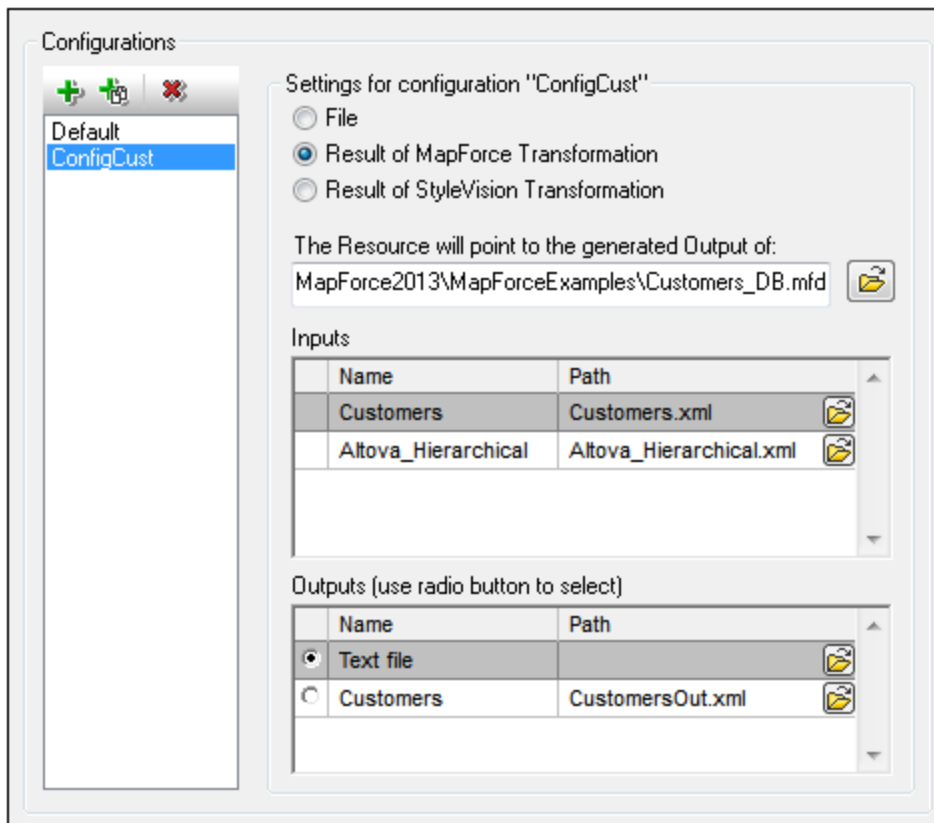
Select the radio button of the desired option (in the screenshot above, 'HTML output' is selected). If the resource is the result of a transformation, then the output can be saved as a file or itself as a global resource. Click the  icon and select, respectively, Global Resource (for saving the output 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](#)⁴¹⁶.

Result of MapForce transformation

Altova MapForce maps one or more (existing) input document schemas to one or more (new) output document schemas. This mapping, which is created by a MapForce user, is known as a MapForce Design (MFD). XML files, text files, databases, etc, that correspond to the input schema/s can be used as data sources. MapForce generates output data files that correspond to the output document schema. This output document is the *Result of MapForce Transformation* file that will become a global resource.

If you wish to set a MapForce-generated data file as a global resource, the following must be specified in the Global Resource dialog (see screenshot below):



- **A .mfd (MapForce Design) file.** You must specify this file in the *Resource will point to generated output of* text box (see screenshot above).
- **One or more input data files.** After the MFD file has been specified, it is analyzed and, based on the input schema information in it, default data file/s are entered in the *Inputs* pane (see screenshot above). You can modify the default file selection for each input schema by specifying another file.
- **An output file.** If the MFD document has multiple output schemas, all these are listed in the *Outputs* pane (see screenshot above) and you must select one of them. If the output file location of an

individual output schema is specified in the MFD document, then this file location is entered for that output schema in the *Outputs* pane. From the screenshot above we can see that the MFD document specifies that the *Customers* output schema has a default XML data file (*CustomersOut.xml*), while the *Text file* output schema does not have a file association in the MFD file. You can use the default file location in the *Outputs* pane or specify one yourself. The result of the MapForce transformation will be saved to the file location of the selected output schema. This is the file that will be used as the global resource

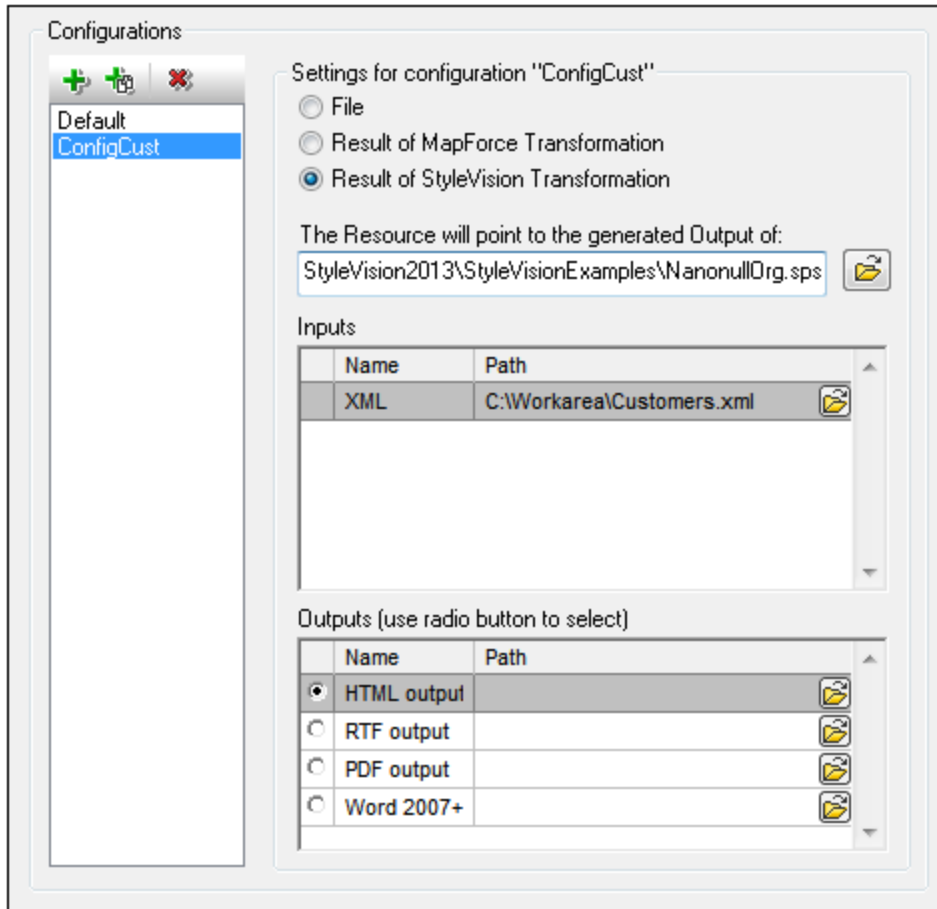
Note: The advantage of this option (Result of MapForce transformation) is that the transformation is carried out at the time the global resource is invoked. This means that the global resource will contain the most up-to-date data (from the input file/s).

Note: Since MapForce is used to run the transformation, you must have Altova MapForce installed for this functionality to work.

Result of StyleVision transformation

Altova StyleVision is used to create StyleVision Power Stylesheet (SPS) files. These SPS files generate XSLT stylesheets that are used to transform XML documents into output documents in various formats (HTML, PDF, RTF, Word 2007+, etc). If you select the option *Result of StyleVision Transformation*, the output document created by StyleVision will be the global resource associated with the selected configuration.

For the *StyleVision Transformation* option in the Global Resource dialog (see screenshot below), the following files must be specified.



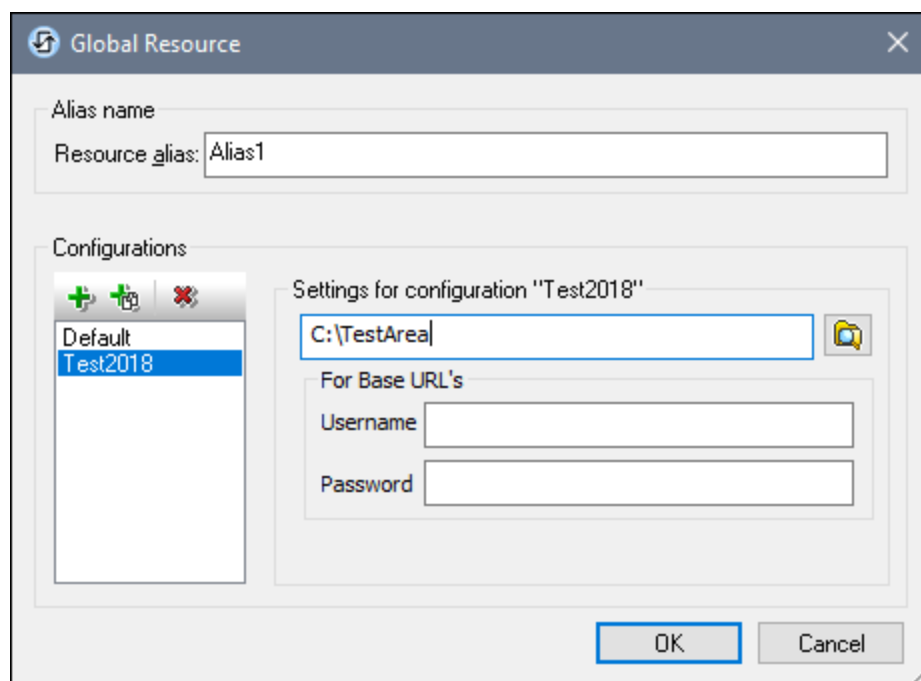
- **A .sps (SPS) file.** You must specify this file in the *Resource will point to generated output* of text box (see screenshot above).
- **Input file/s.** The input file might already be specified in the SPS file. If it is, it will appear automatically in the *Inputs* pane once the SPS file is selected. You can change this entry. If there is no entry, you must add one.
- **Output file/s.** Select the output format in the *Outputs* pane, and specify an output file location for that format.

Note: The advantage of this option (Result of StyleVision transformation) is that the transformation is carried out when the global resource is invoked. This means that the global resource will contain the most up-to-date data (from the input file/s).





Note: Since StyleVision is used to run the transformation, you must have Altova StyleVision installed for this functionality to work.

9.1.1.2 Folders

In the Global Resource dialog for Folders (*screenshot below*), add a folder resource as described below.



Global Resource dialog icons

-  **Add Configuration:** Pops up the Add Configuration dialog in which you enter the name of the configuration to be added.
-  **Add Configuration as Copy:** Pops up the Add Configuration dialog in which you can enter the name of the configuration to be created as a copy of the selected configuration.
-  **Delete:** Deletes the selected configuration.
-  **Open:** Browse for the folder to be created as the global resource.

Defining the alias

Define the alias (its name and configurations) as follows:

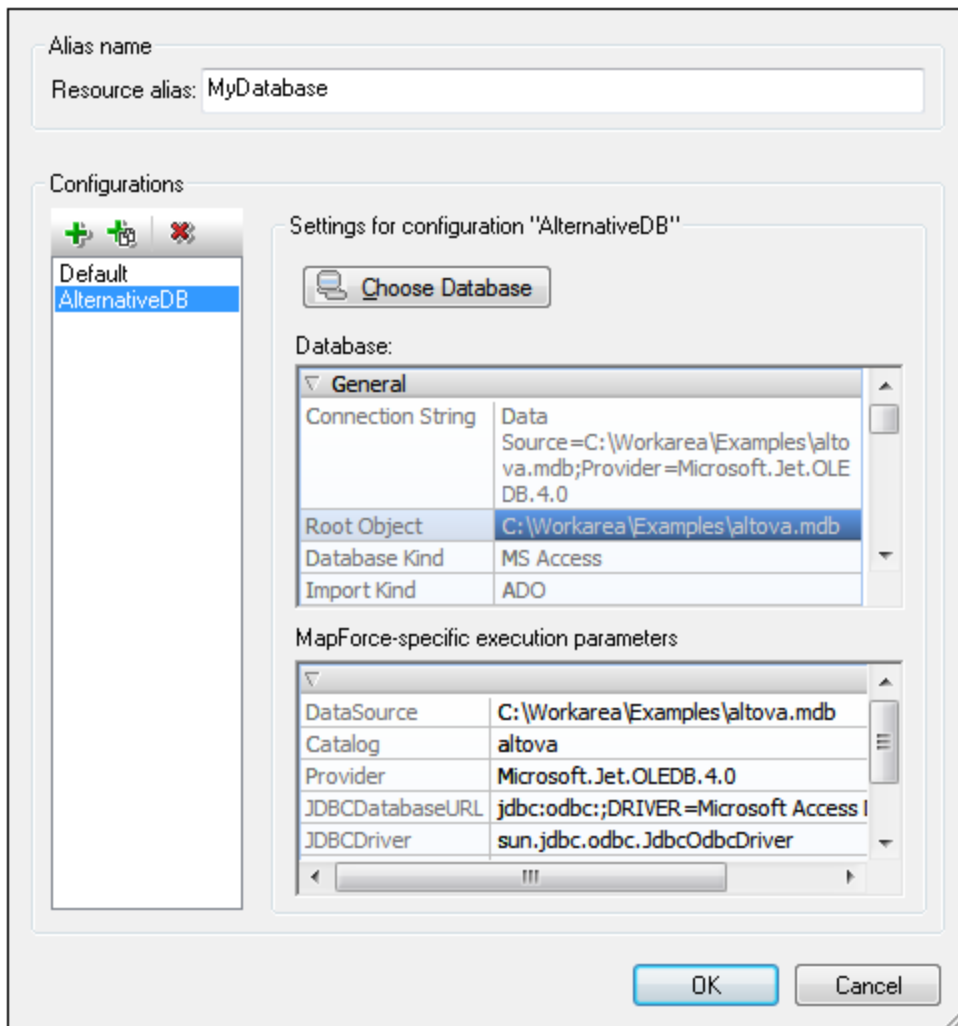
1. **Give the alias a name:** Enter the alias name in the *Resource Alias* text box.
2. **Add configurations:** The Configurations pane will have a configuration named *Default* (see *screenshot above*). This *Default* configuration cannot be deleted nor have its name changed. You can enter as many additional configurations for the selected alias as you like. Add a configuration by clicking the **Add Configuration** or **Add Configuration as Copy** icons. In the dialog which pops up, enter the

configuration name. Click **OK**. The new configuration will be listed in the Configurations pane. Repeat for as many configurations as you want.

3. *Select a folder as the resource of a configuration:* Select one of the configurations in the Configurations pane and browse for the folder you wish to create as a global resource. If security credentials are required to access a folder, then specify these in the *Username* and *Password* fields.
4. *Define multiple configurations if required:* Specify a folder resource for each configuration you have created (that is, repeat Step 3 above for the various configurations you have created). You can add a new configuration to the alias definition at any time.
5. *Save the alias definition:* Click **OK** in the Global Resource dialog to save the alias and all its configurations as a global resource. The global resource will be listed under Folders in the [Manage Global Resources dialog](#)⁴¹⁶.

9.1.1.3 Databases

In the Global Resource dialog for Databases (*screenshot below*), you can add a database resource as follows:



Global Resource dialog icons



Add Configuration: Pops up the Add Configuration dialog in which you enter the name of the configuration to be added.



Add Configuration as Copy: Pops up the Add Configuration dialog in which you can enter the name of the configuration to be created as a copy of the selected configuration.

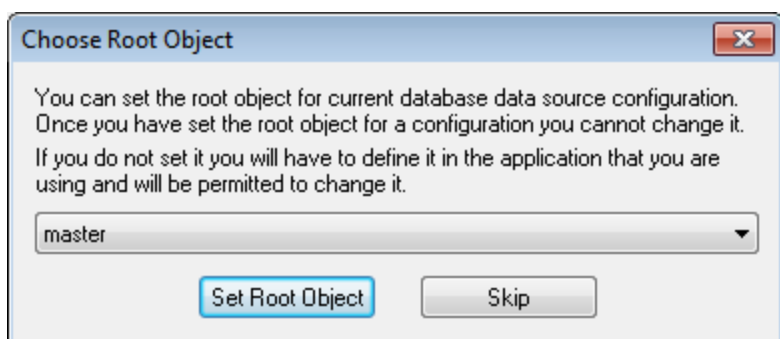


Delete: Deletes the selected configuration.

Defining the alias

Define the alias (its name and configurations) as follows:

1. *Give the alias a name:* Enter the alias name in the *Resource Alias* text box.
2. *Add configurations:* The Configurations pane will have a configuration named Default (see screenshot above). This Default configuration cannot be deleted nor have its name changed. You can enter as many additional configurations for the selected alias as you like. Add a configuration by clicking the **Add Configuration** or **Add Configuration as Copy** icons. In the dialog which pops up, enter the configuration name. Click **OK**. The new configuration will be listed in the Configurations pane. Repeat for as many configurations as you want.
3. *Start selection of a database as the resource of a configuration:* Select one of the configurations in the Configurations pane and click the **Choose Database** icon. This pops up the Create Global Resources Connection dialog.
4. *Connect to the database:* Select whether you wish to create a connection to the database using the Connection Wizard, an existing connection, an ADO Connection, an ODBC Connection, or JDBC Connection. Complete the definition of the connection method as described in the section [Connecting to a Database](#)⁴⁹³. 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.



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.

9.1.2 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](#)⁴²⁷ and [Databases](#)⁴³⁰.

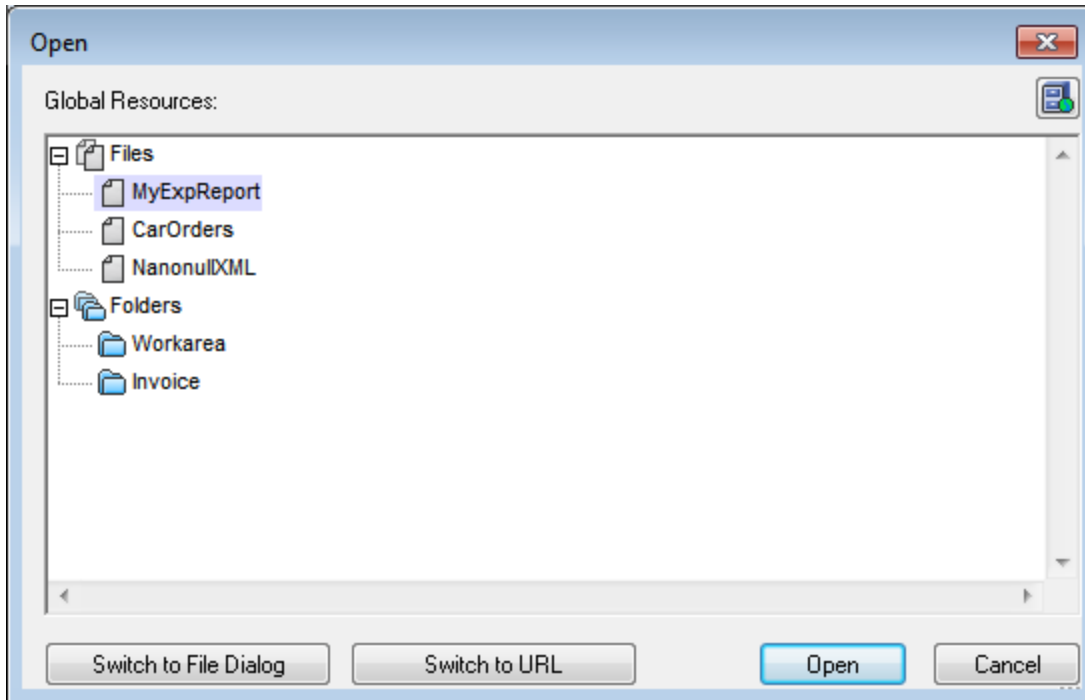
Selections that determine which resource is used

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](#)⁴¹⁶. 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](#)⁸¹⁸ or via the [Global Resources toolbar](#)⁷¹². 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.

9.1.2.1 Assigning Files and Folders

File-type and folder-type global resources are assigned differently. In any one of the [usage scenarios](#)⁴²⁸ below, clicking the **Switch to Global Resources** button displays the Open Global Resource dialog (*screenshot below*).



Manage Global Resources: Displays the [Manage Global Resources](#)⁴¹⁶ 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](#)⁴¹⁶ dialog.

Usage scenarios

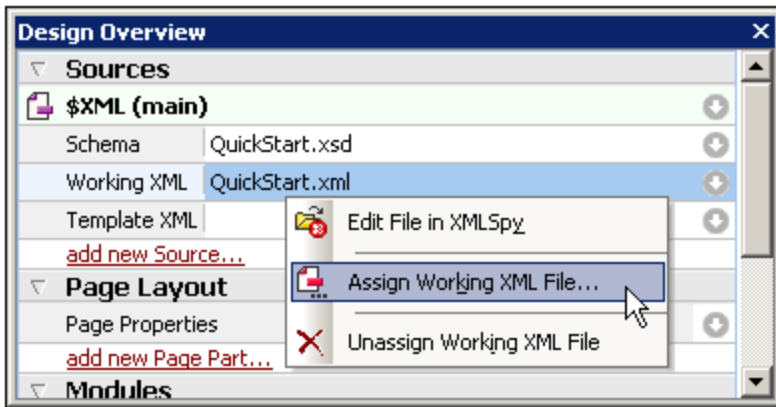
File-type and folder-type global resources can be used in the following scenarios:

- [Adding and modifying schema sources and Working XML Files](#)⁴²⁷ and [Template XML Files](#)⁴²⁷
- [Saving as Global Resource](#)⁴²⁹
- [Adding modules and CSS files](#)⁴²⁹
- [Adding global resources to a project](#)⁴³⁰

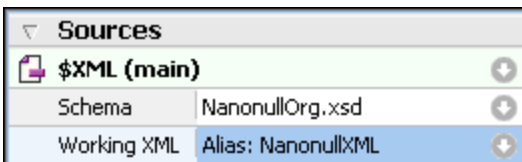
Schema, Working XML File, Template XML File

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](#)⁴² or [Working XML File](#)⁴² 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](#) ⁴¹⁶.)



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*).

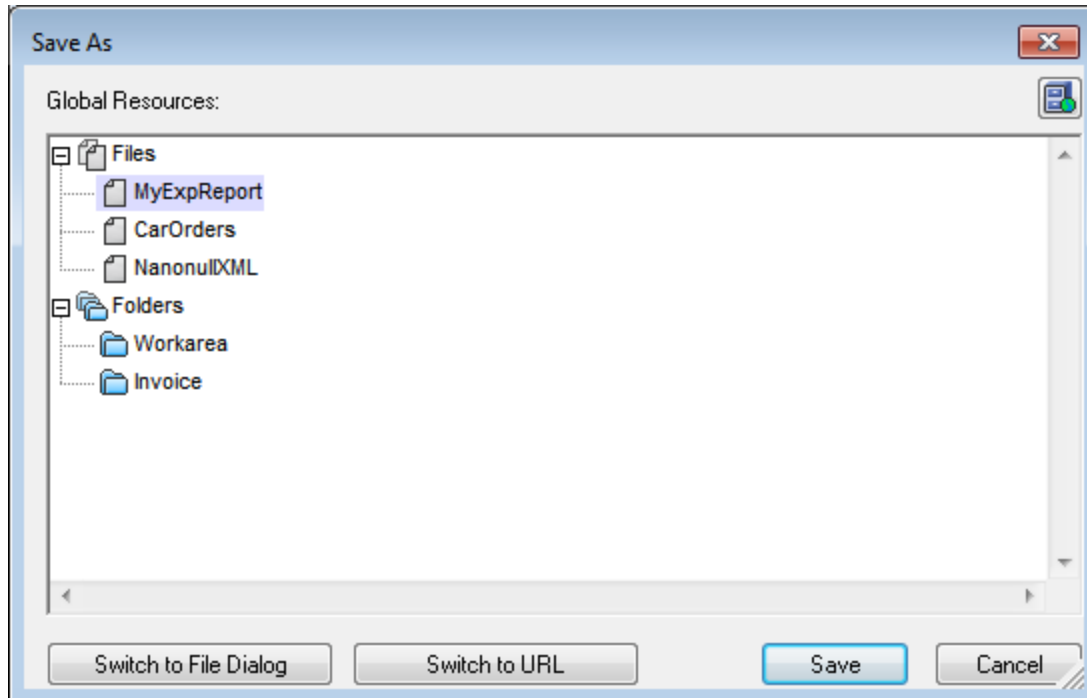


Adding modules and CSS files from a global resource

In the [Design Overview sidebar](#) ⁴², 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](#)⁴¹⁹, the file will be saved directly. If the alias is a [folder alias](#)⁴²⁴, 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](#)⁴³¹.

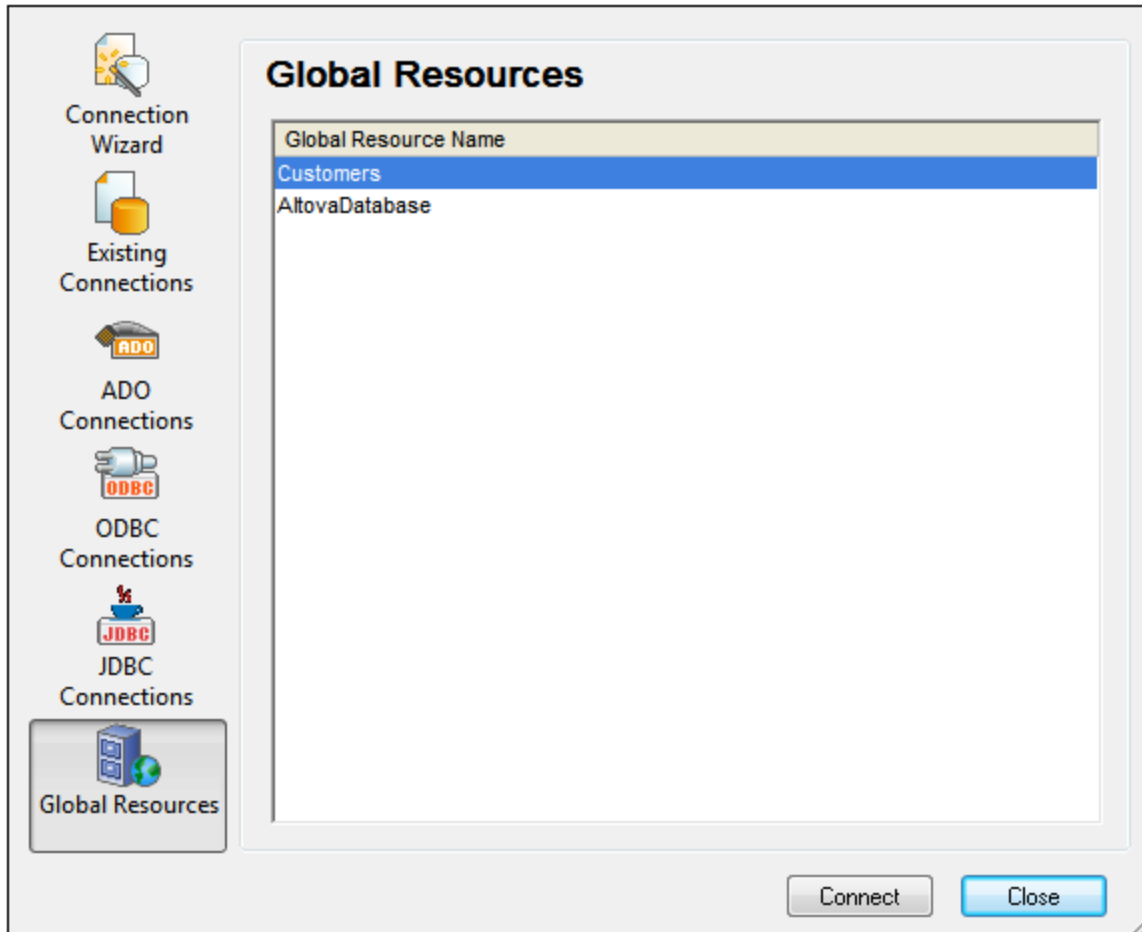
Note: Each configuration points to a specific file location, which is specified in the definition of that configuration. If the file you are saving as a global resource does not have the same filetype extension as the file at the current file location of the configuration, then there might be editing 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](#)⁴¹⁶. 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.

9.1.2.2 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](#)⁴¹⁶ 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.

9.1.2.3 Changing the Active Configuration

One configuration of a global resource can be active at any time. This configuration is called the active configuration, and it is active application-wide. This means that the active configuration is active for all global resources aliases in all currently open files and data source connections. If an alias does not have a configuration with the name of the active configuration, then the default configuration of that alias will be used. As an example of how to change configurations, consider the case in which a file has been assigned via a global resource with multiple configurations. Each configuration maps to a different file. So, which file is selected depends on which configuration is selected as the application's active configuration.

Switching the active configuration can be done in the following ways:

- Via the menu command **Tools | Active Configuration**. Select the configuration from the command's submenu.
- In the combo box of the Global Resources toolbar (*screenshot below*), select the required configuration.

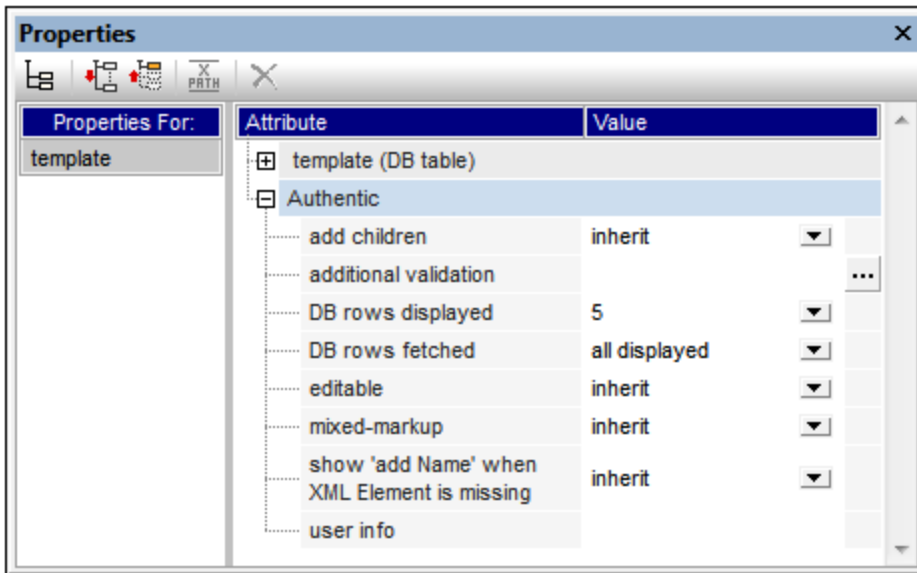


In this way, by changing the active configuration, you can change source files that are assigned via a global resource.

9.2 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](#)⁴³⁸ 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 *DB Records Fetched* property thus enables you to speed up the loading and display time.

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.

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.

Show "add Name" when XML Element is missing

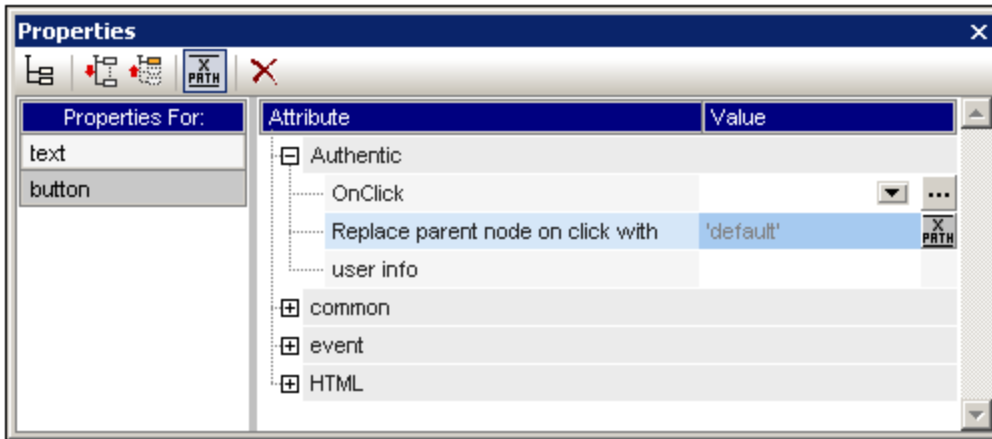
Determines whether a prompt ("Add [element/attribute name]") will appear in Authentic View when the selected element or attribute is missing. By default, the prompt will be displayed. This setting is available when the selected node is an element or an attribute.

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.

9.3 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](#)¹⁸⁰ or [hyperlink](#)³⁴¹.



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:

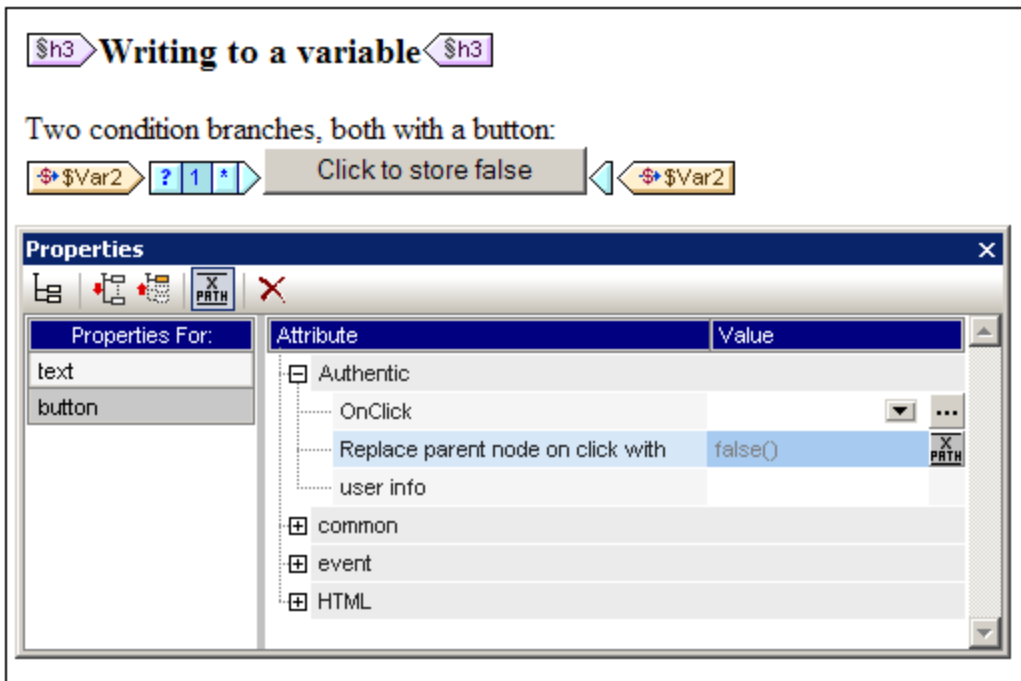
1. In the design, insert the design component (button or hyperlink) within the node for which you wish to insert text content.
2. Select the design component in the design, and, in the Properties sidebar, select that design component (in the *Properties For* column), and enter a value for the *Replace 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](#)²⁸⁰, 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](#)⁴⁶⁷.

Note: For this feature to work correctly, in the schema, the parent node must be defined to allow text content.

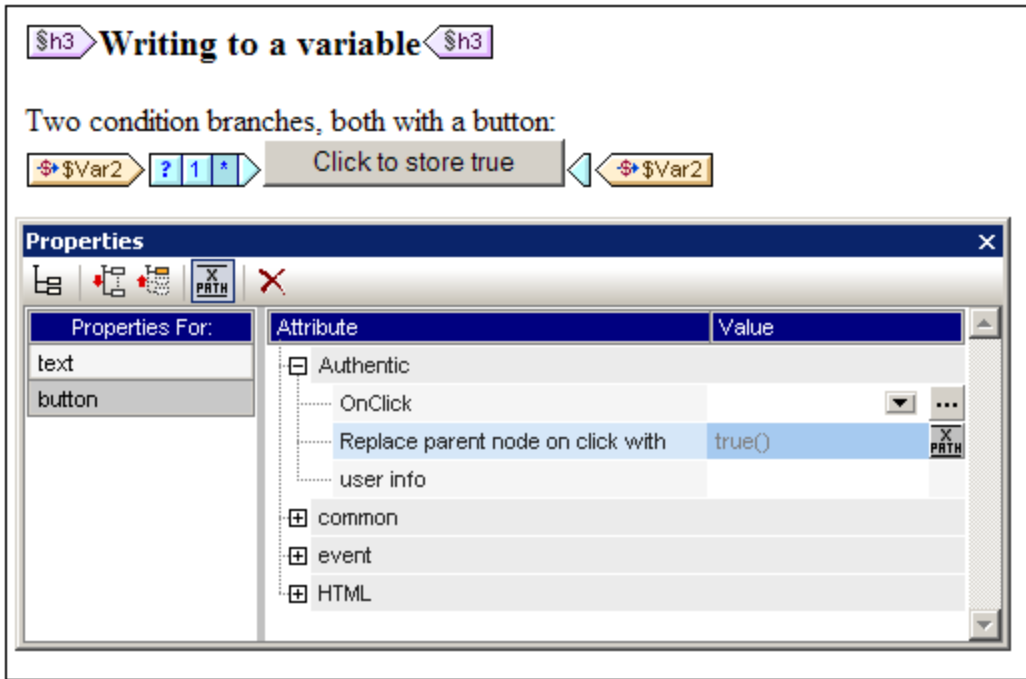
Usage with variables

The parent node can also be a [variable](#)³⁰² 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](#)¹⁸⁰ or [hyperlink](#)³⁴¹), 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](#)²⁸⁰.



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.

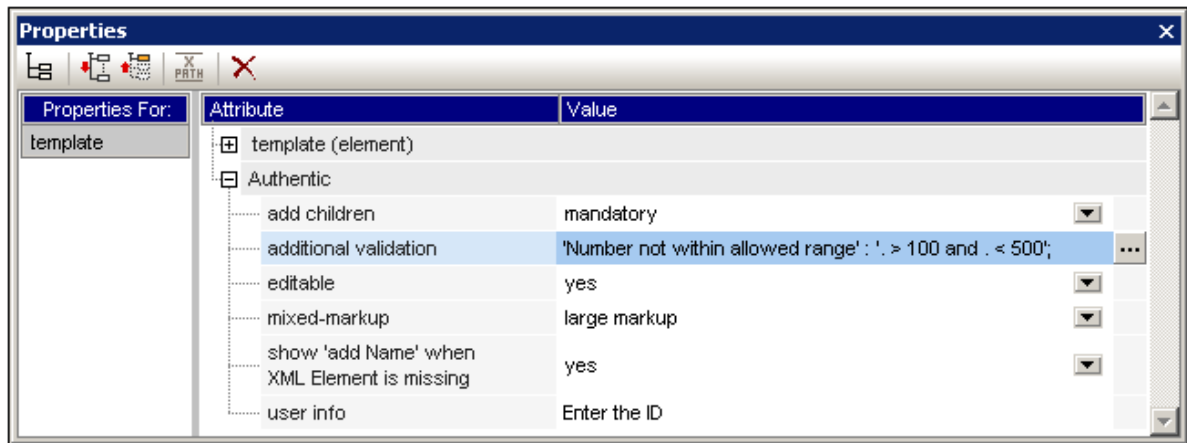
9.4 Additional Validation

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.

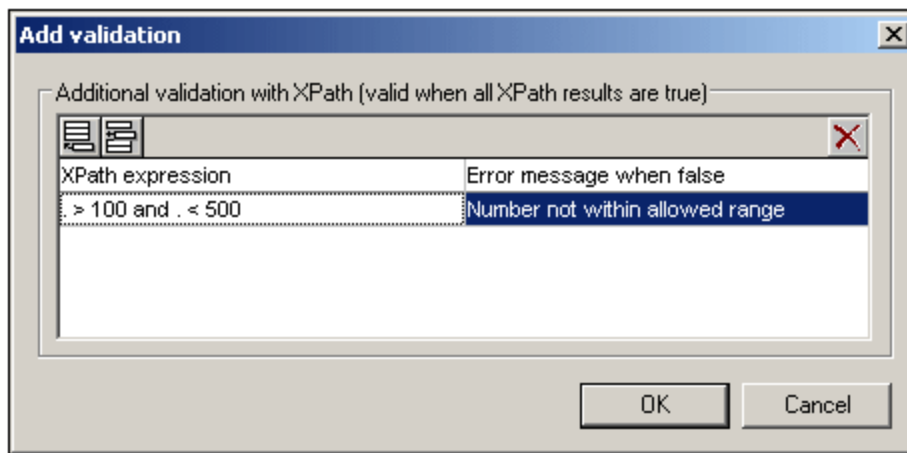
Setting Additional Validation

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  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.

9.5 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); not in Authentic View. Note that unparsed entity URIs are not processed for Text output.

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](#)⁴⁴⁰, 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](#)⁴⁴⁰. This is done by giving the names of the required unparsed entities.
3. In the SPS, unparsed entities can be used to target [images](#)¹⁶⁸ and [hyperlinks](#)³⁴¹ by [correctly accessing the relevant dynamic node values as unparsed entities](#)⁴⁴⁰.

Declaring and referencing unparsed entities

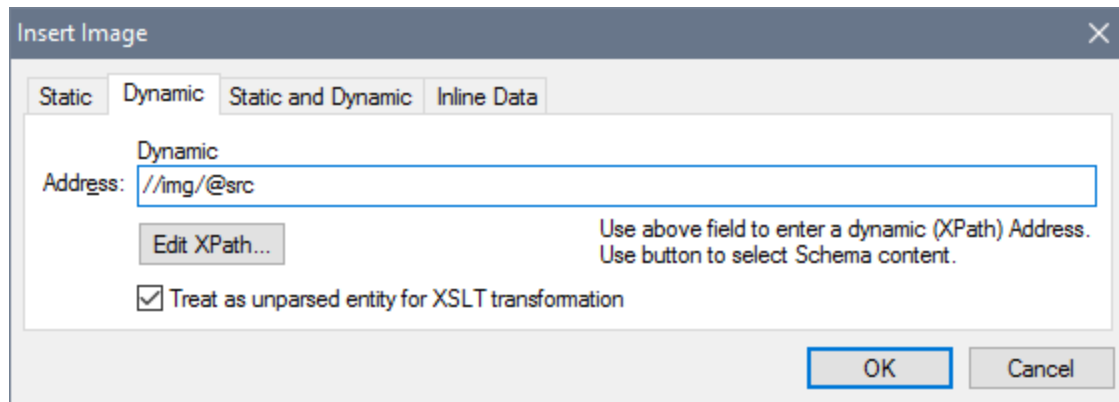
Given below is a cut-down listing of an XML document. It has an internal DTD subset which declares two unparsed entities, one with a GIF notation (indicating a GIF image) and the other with an LNK notation (indicating a hyperlink). The `img/@src` and `link/@href` nodes in the XML code reference the unparsed entities by giving their names.

```
<?xml 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>
  
  <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 object's Edit dialog, 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: If the URI is a relative URI, then the XSLT processor expands it to an absolute URI applying the base URI of the DTD. For example, 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`.

9.6 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](#)³³.
5. You can now modify the SPS in the usual way. For example, you could drag in nodes from the [Schema Tree](#)⁴⁴, modify the styling and presentation or add additional styling, and use StyleVision functionality such as [Auto-Calculations](#)²⁷⁰ and [Conditional Templates](#)²⁸⁰.
6. You can save the SPS and use a [Working XML File](#)⁴² to preview [various output formats](#)³⁶. Subsequently you can [generate stylesheets and output files](#)⁷³⁴ using the [Save Generated Files](#)⁷³⁴ command.

Example

The example discussed below is located in the [\(My\) Documents folder](#)²⁹, C:\Documents and Settings\<>username>\My Documents\Altova\StyleVision2024\StyleVisionExamples\Tutorial\NewFromXSLT. This folder contains the files: SimpleExample.xslt, SimpleExample.xsd, and SimpleExample.xml.

The XML file is shown below.

XML file used in charts example: YearlySales.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>
  </Region>
</Data>
```

```

        <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>

```

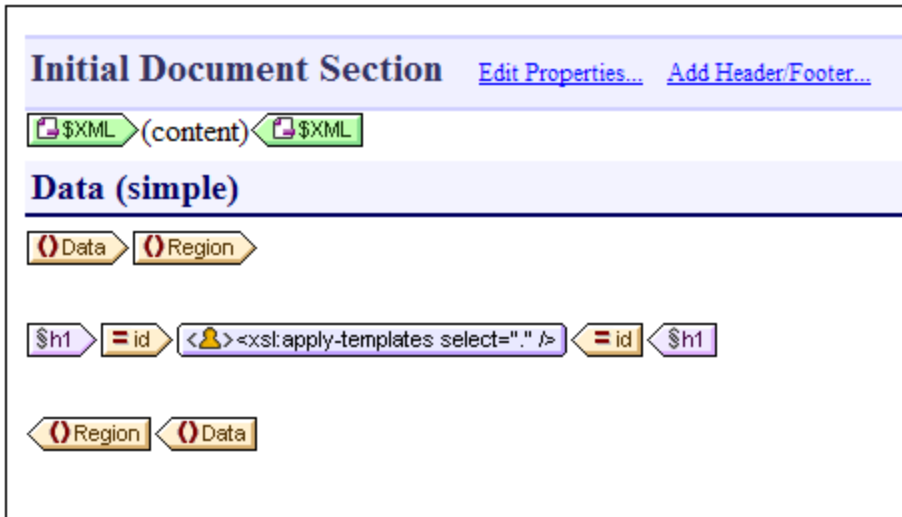
The XSLT file is as follows:

```

<?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:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <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>

```

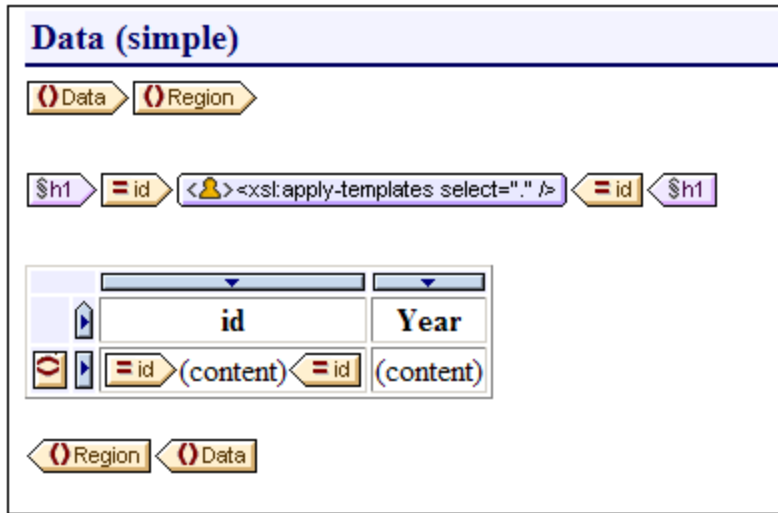
Follow the steps 1 to 4 listed above to obtain the SPS in Design View. The SPS will look something like this:



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.



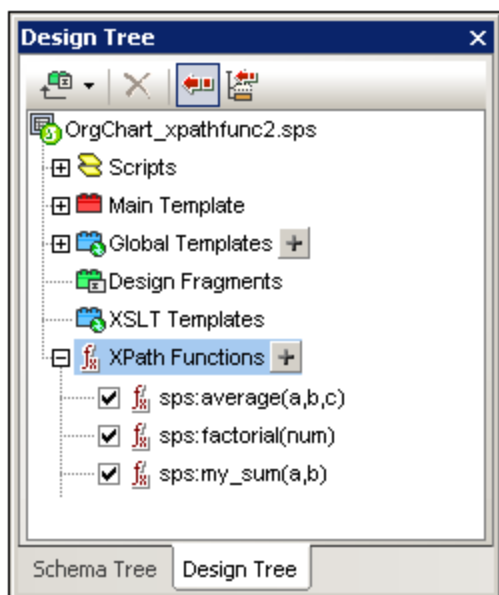
You can make additional changes in the content, structure, and presentation properties of the document, then preview the output and save files using the [Save Generated Files](#)⁷³⁴ command.


9.7 User-Defined XPath Functions

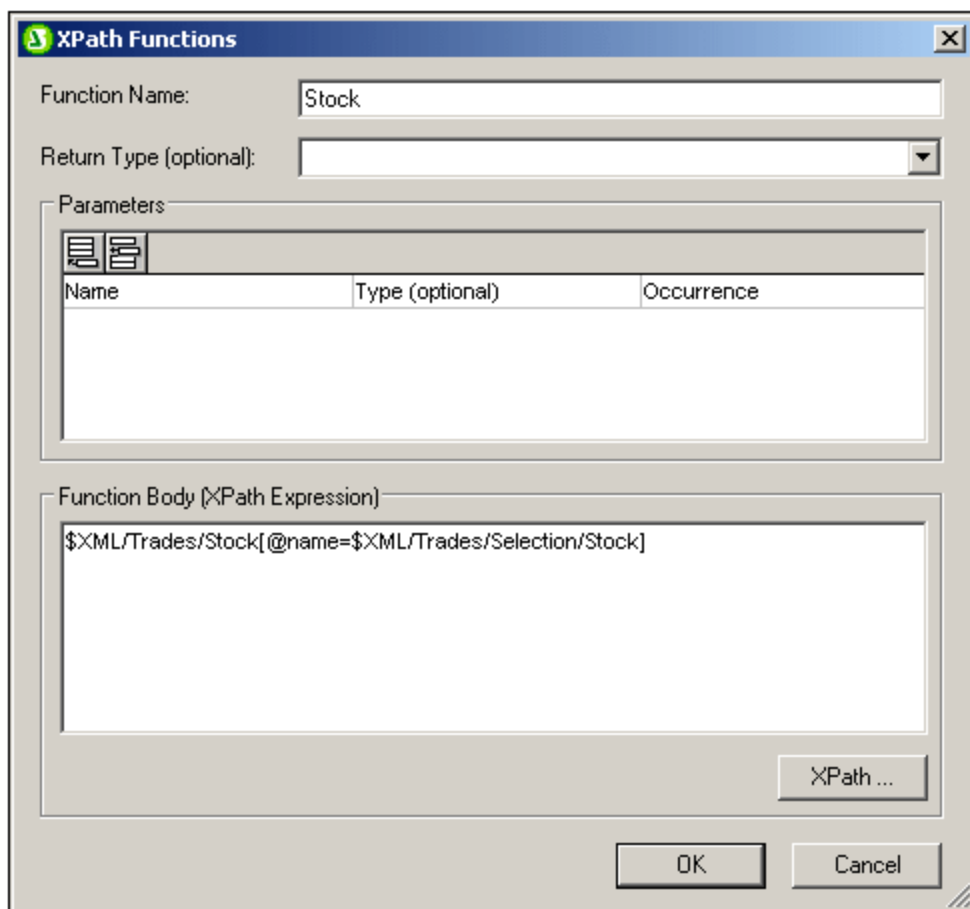
The SPS designer can define customized **XPath 2.0/3.0** functions. A user-defined XPath function can be re-used in any design component that accepts an XPath expression, for example, in Auto-Calculations, conditions, and combo boxes.

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.



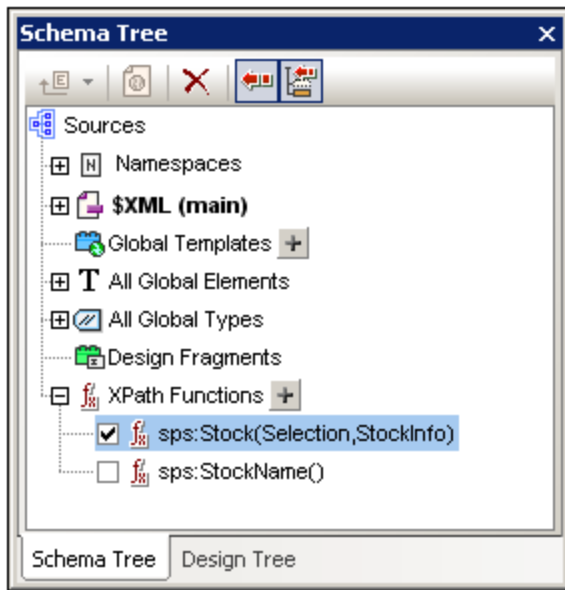
After a user-defined XPath function is created, it is available for use anywhere in the design.

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: `$XML/Trades/Stock[@name=$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.

9.7.1 Defining an XPath Function

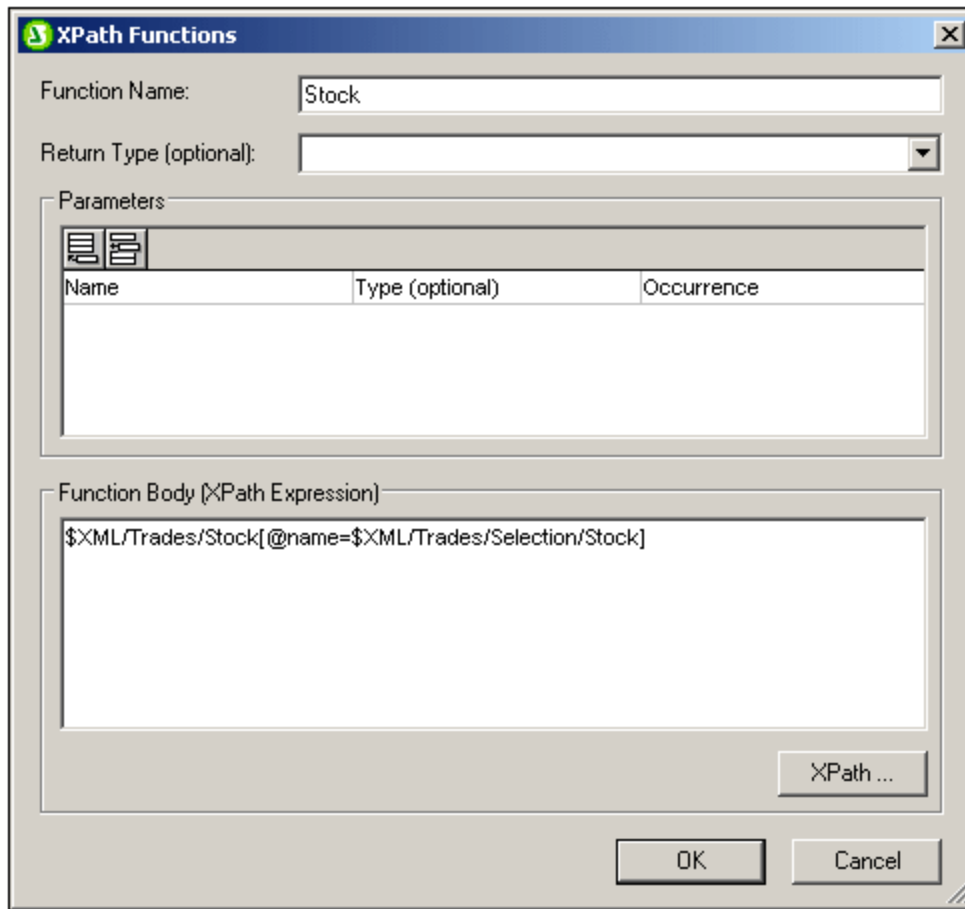
A user-defined XPath function requires: (i) a name (a text string), and (ii) a definition (an XPath expression).

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.

```
<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](#)⁴⁵⁴ and [Parameters and Nodes](#)⁴⁵⁹, 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](#)⁴⁵⁹ below for an explanation.

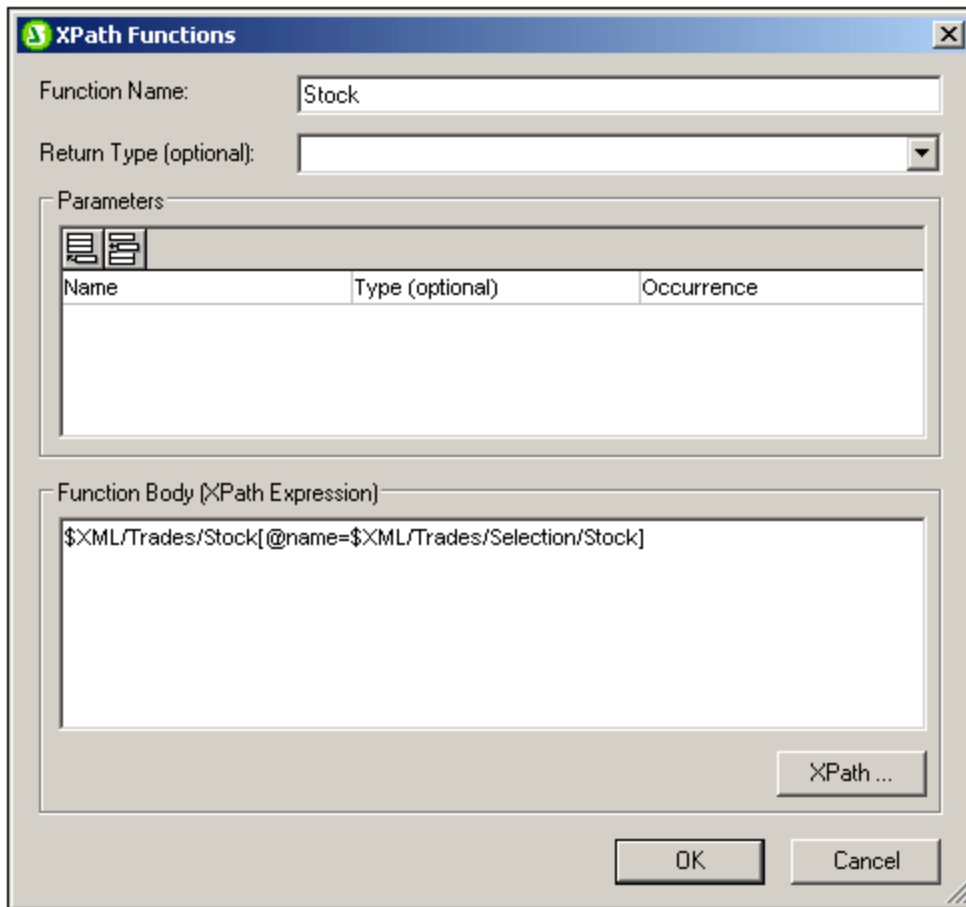
In the following cases, errors are returned:

- 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.

9.7.2 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.

9.7.3 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).

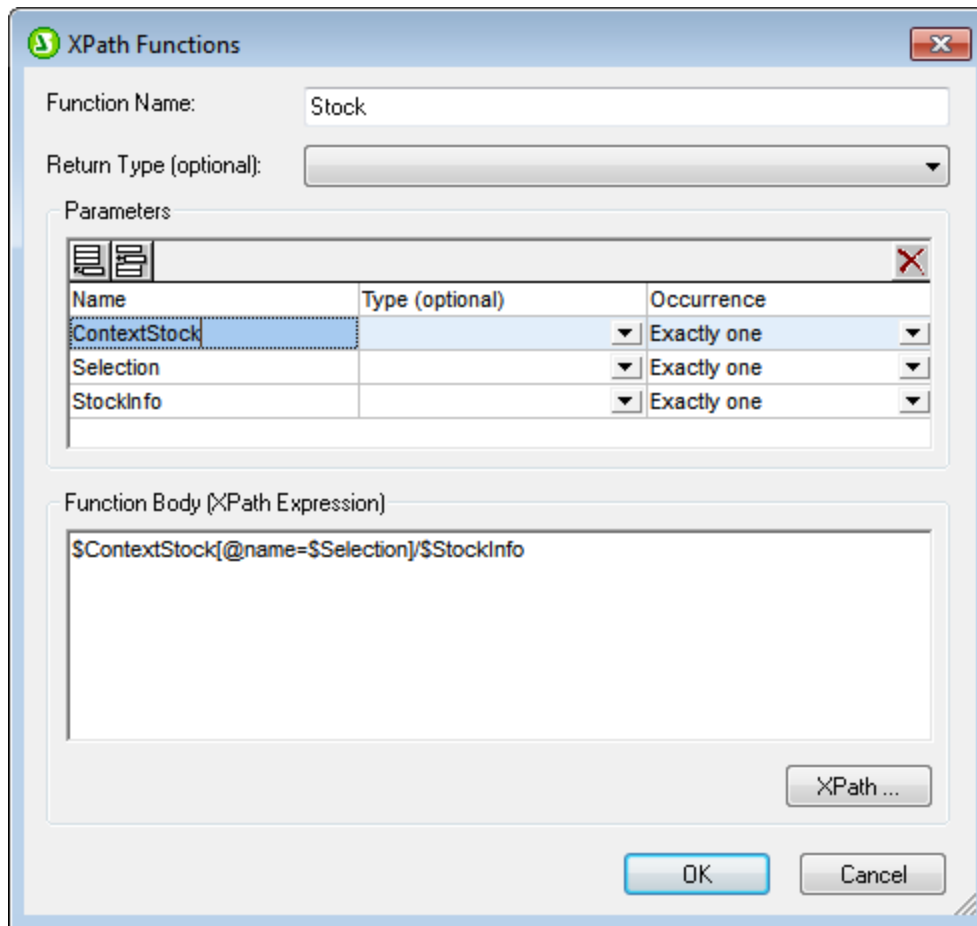
User-defined XPath function mechanism

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:

```
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`.

9.7.3.1 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).*

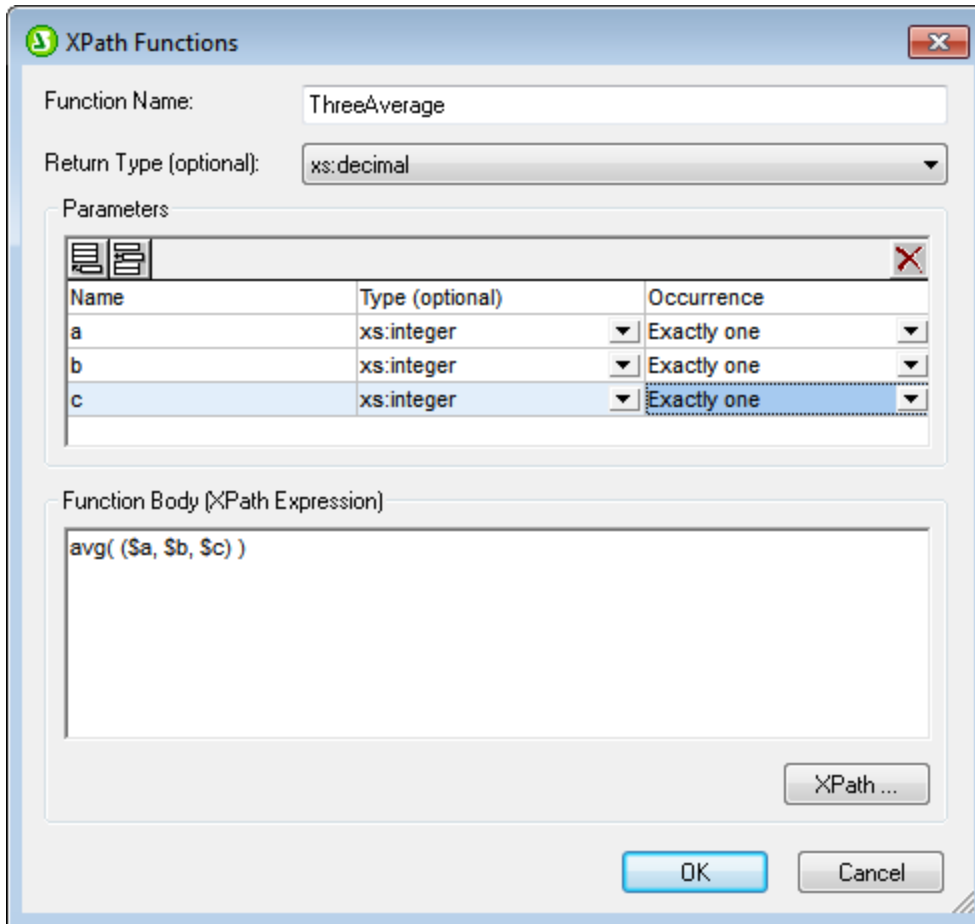
Parentheses are not necessary around a path (or locator) expression (example of a path expression: `//Person/@salary`), because a path expression can be read unambiguously as one parameter or one argument. It is in fact a one-sequence parameter/argument.

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 `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 `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.

- `sps:ThreeAverage(10,20,30)` returns 20. There are three valid arguments in the function call, corresponding to the three XPath parameters.

- `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).
- `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).
- `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').
- `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.

The screenshot shows a dialog box titled "XPath Functions" with the following fields and table:

Function Name:

Return Type (optional):

Parameters:

Name	Type (optional)	Occurrence
a	xs:integer	At least one
b	xs:integer	Exactly one
c	xs:integer	Exactly one

Function Body (XPath Expression):

```
avg( count($a), $b, $c )
```

Buttons: XPath..., OK, Cancel

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 `avg()` function is enclosed in parentheses. This is to specify that the `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 `count()` function; the second and third are the two parameters `b` and `c`.
- The argument of the `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 `Average()` shown in the screenshot above.

- `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 `count()` function operates on it, the function will return the value 2, which will be the first item of the sequence submitted to the `avg()` function.
- `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 *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:

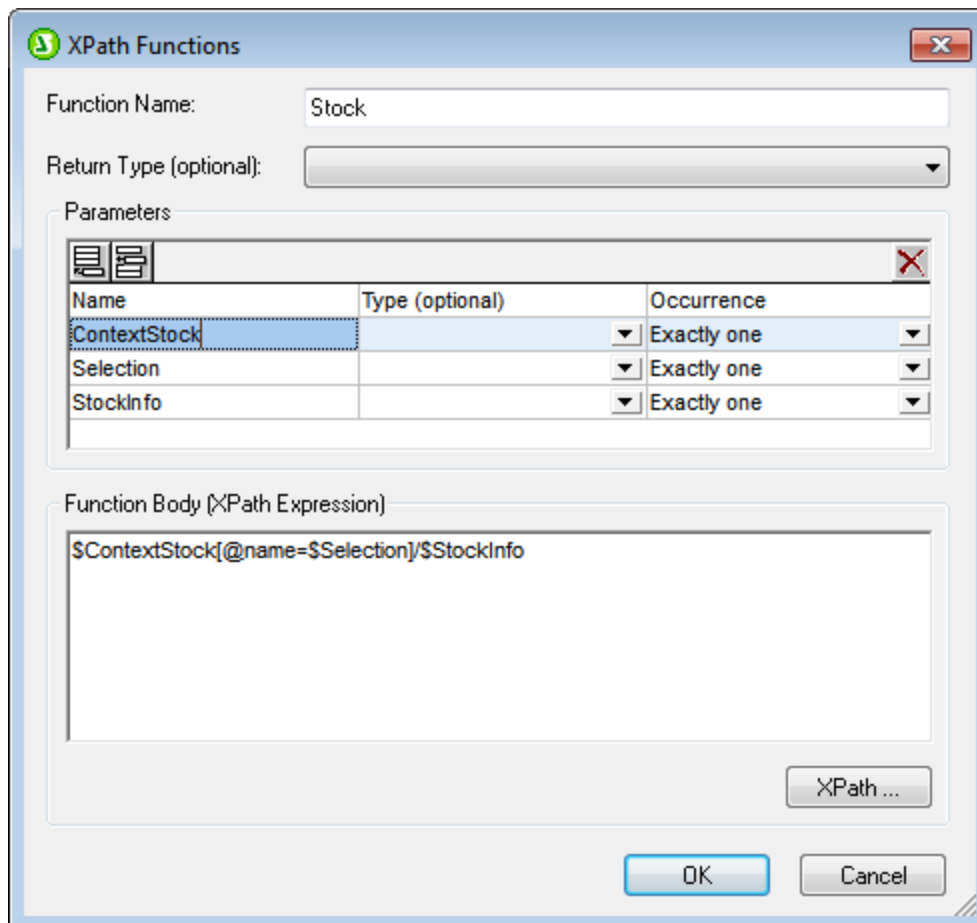
```
if ($num < 2) then 1 else $num * sps:Factorial($num - 1)
```

If this function were called `Factorial()`, then the factorial of, say 6, could be obtained by calling the function with: `sps:Factorial(6)`.

9.7.3.2 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.

9.8 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](#)⁴⁶¹). StyleVision supports the `xs:date` or `xs:dateTime` datatypes by providing:

1. A graphical [date picker](#)⁴⁶² 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](#)⁴⁶³ possibilities via the [Input Formatting](#)³⁵⁴ feature.

These StyleVision features are described in the sub-sections of this section: [Using the Date-Picker](#)⁴⁶² and [Formatting Dates](#)⁴⁶³. 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](#)⁴⁶² to maintain data integrity and use Input Formatting to provide [date formatting](#)⁴⁶³.

Date calculations with XPath 2.0

Data involving dates can be manipulated with XPath 2.0 expressions in [Auto-Calculations](#)²⁷⁰. 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.

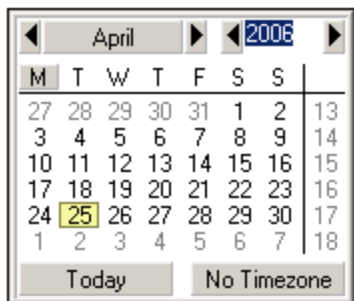
Here is an XPath 2.0 expression in an Auto-Calculation. It calculates a 4% annual interest on an overdue amount on a per-day basis and returns the sum of the principal amount and the accumulated interest:

```
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.

9.8.1 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 \pm) 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 \pm) 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*).

Organization Chart

Location of logo:

Last Updated: 2003-09-01 

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.

Location of logo:

Last Updated: 2003-09-01

Nanonull, Inc.

Location:

◀ September ▶ 2003 ▶

M	T	W	T	F	S	S	
25	26	27	28	29	30	31	35
1	2	3	4	5	6	7	36
8	9	10	11	12	13	14	37
15	16	17	18	19	20	21	38
22	23	24	25	26	27	28	39
29	30	1	2	3	4	5	40

Today No Timezone

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*).

Location of logo:

Last Updated: 2003-09-01+01:00

Nanonull, Inc.

Location:

◀ September ▶ 2003 ▶

M	T	W	T	F	S	S	
25	26	27	28	29	30	31	35
1	2	3	4	5	6	7	36
8	9	10	11	12	13	14	37
15	16	17	18	19	20	21	38
22	23	24	25	26	27	28	39
29	30	1	2	3	4	5	40

Today +01 : 00

9.8.2 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.

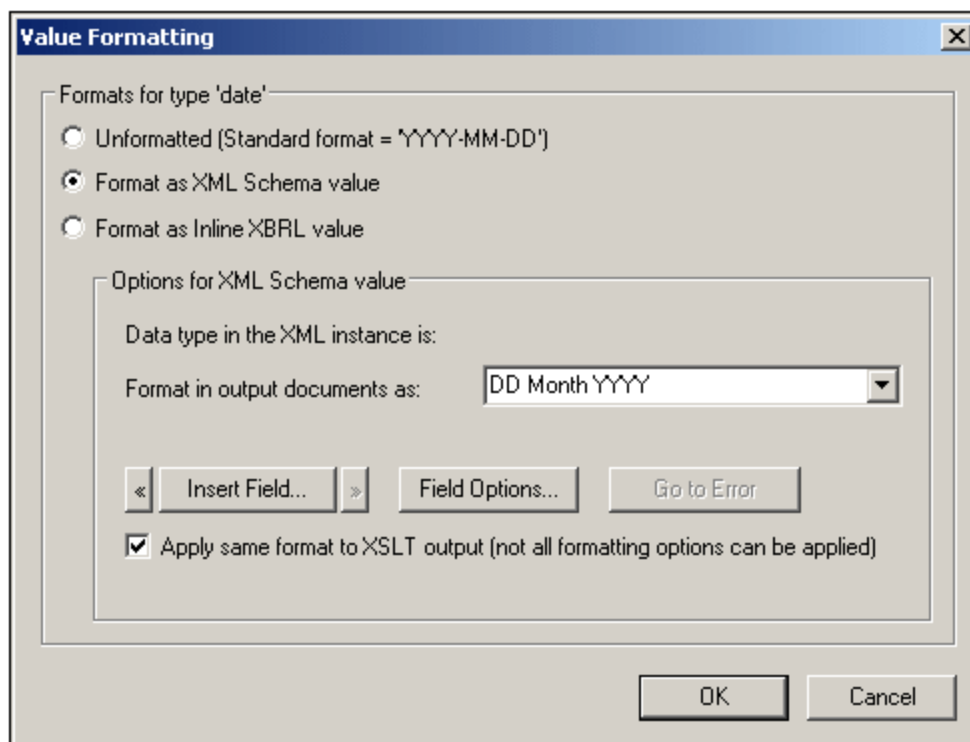
Value Formatting for dates can be used to define custom formats for nodes and Auto-Calculations of the following datatypes:

- `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 `autocalc` item, and then the *AutoCalc* group of properties. Now click the Edit button  of the *Value Formatting* property. This displays the Value Formatting dialog (screenshot below).




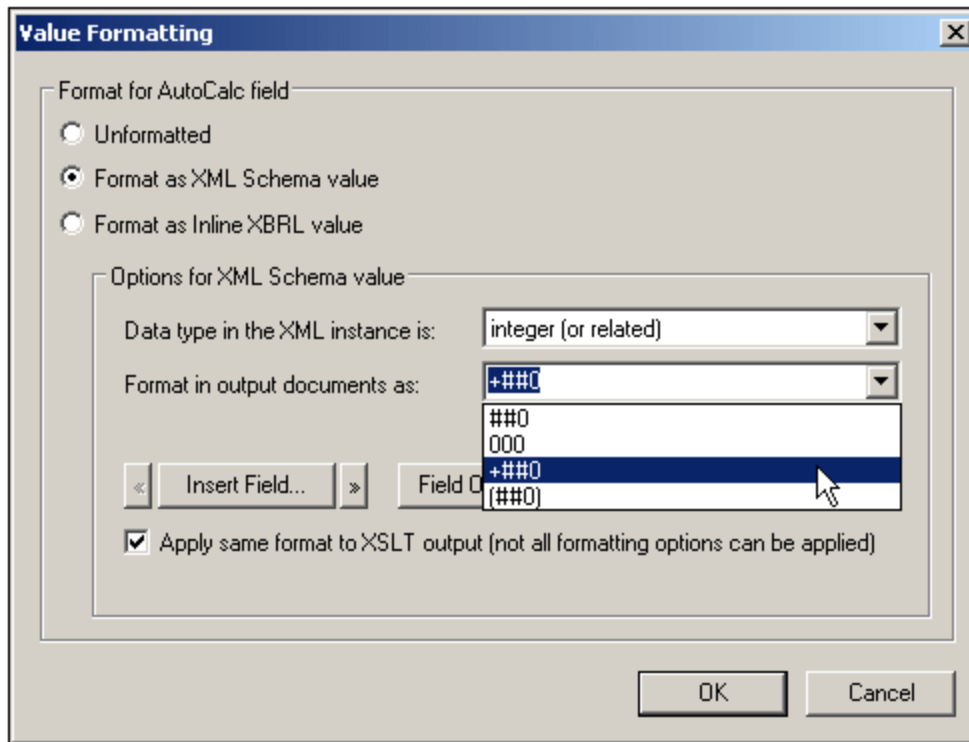
By default, the *Unformatted* radio button (the standard lexical format for the node's datatype) is selected.

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](#)³⁵⁷ 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](#)³⁵⁷ 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, Text, and RTF 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.

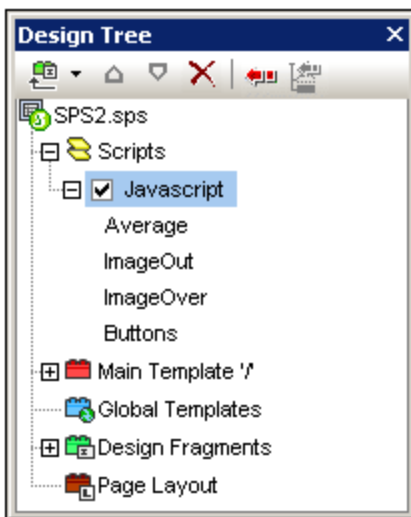
9.9 Using Scripts

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:

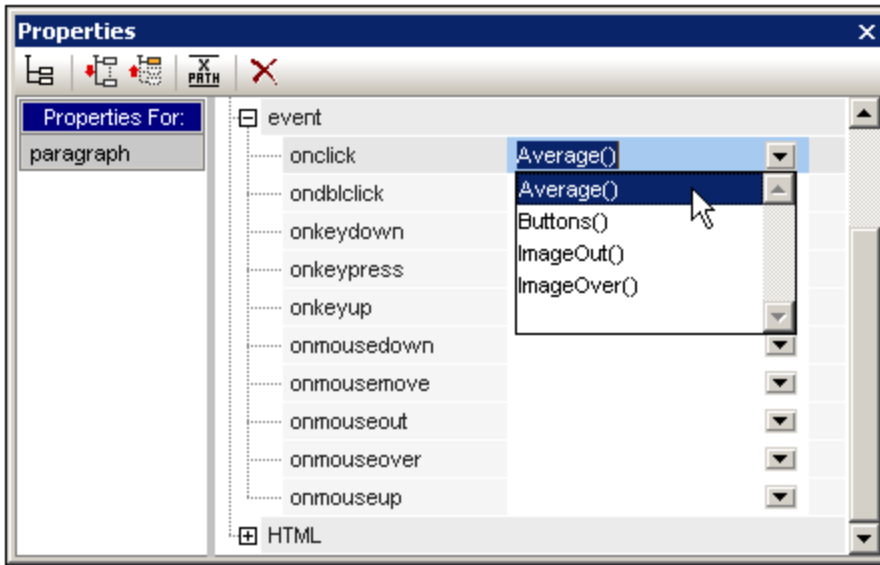
- You wish to achieve a complex result using multiple script statements. In this case it is convenient to write all the required scripts, as separate functions, in one location (the header) and refer to the functions subsequently in the design document.
- You wish to use a particular script at multiple locations in the design document.

How to define functions in the JavaScript Editor is described in the sub-section [Defining JavaScript Functions](#)⁴⁶⁸.

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](#)⁴⁶⁹.



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](#)²³⁰, 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](#)²³⁰.

9.9.1 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*).

```
1  function DisplayTime()
2  {
3    now = new Date();
4    hours = now.getHours();
5    mins = now.getMinutes();
6    secs = now.getSeconds();
7    result = hours + ":" + mins + ":" + secs;
8    alert(result);
9  }
10
11 function ClearStatus()
12 {
13   window.status="";
14 }
```

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:

```
<script language="javascript">

<!-- function DisplayTime ()
{
    now = new Date ();
    hours = now.getHours ();
    mins = now.getMinutes ();
    secs = now.getSeconds ();
    result = hours + "." + mins + "." + secs;
    alert (result)
}

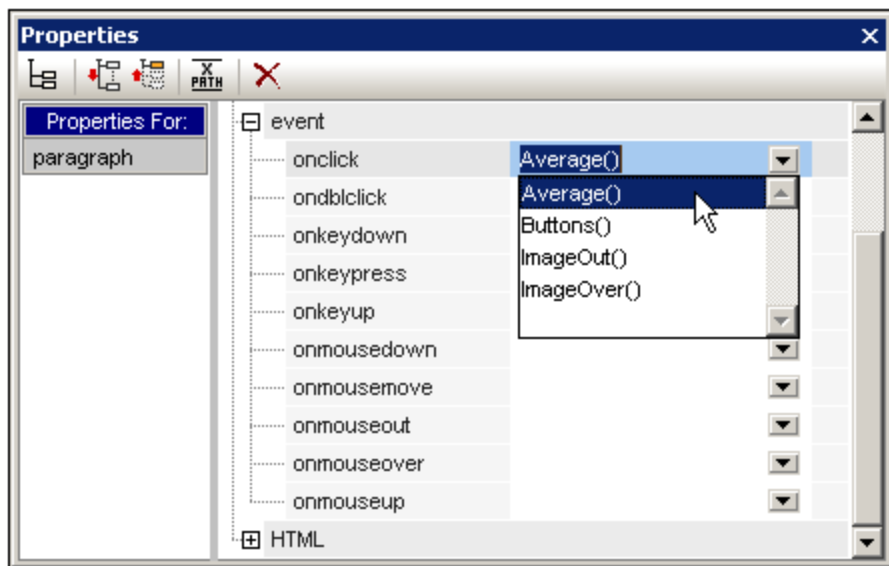
function ClearStatus ()
{
    window.status="";
}
-->

</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](#)⁴⁶⁹ for details.

9.9.2 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.

9.9.3 External JavaScript Files

An SPS can access external JavaScript files in two ways:

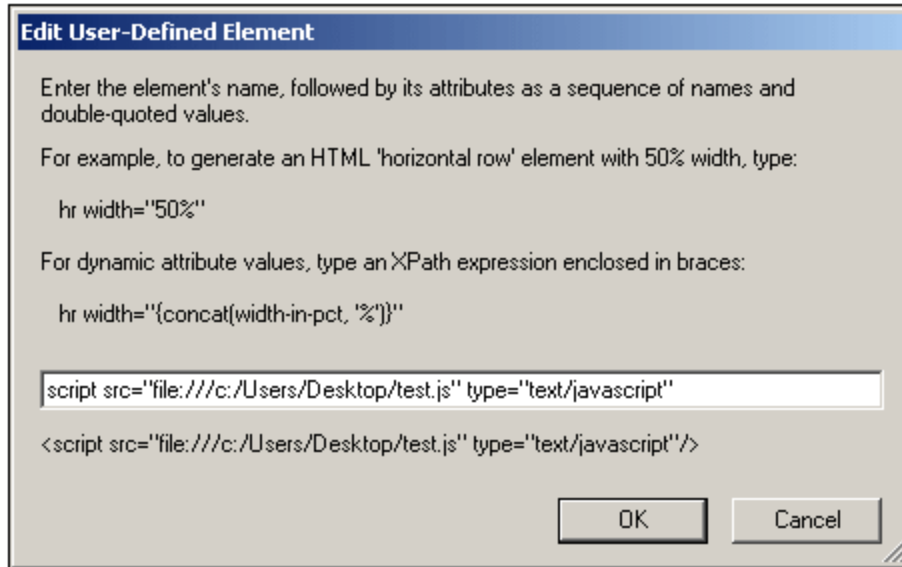
1. [By creating a User-Defined Element or User-Defined XML Block](#)⁴⁷⁰. These design objects can contain a `SCRIPT` element that accesses the external JavaScript file. Note that location of the User-Defined Element or User-Defined XML Block is within the `BODY` element of the design (and therefore within the `BODY` element of the HTML output, not within the `HEAD` element).
2. [By adding a script in the Javascript Editor](#)⁴⁷¹ 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](#)¹⁴⁰ and [User-Defined XML Blocks](#)¹⁴¹. 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](#)¹⁴⁰ 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](#) ¹⁴⁰.



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](#) ¹⁴¹ 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](#) ¹⁴¹ is inserted instead of a [User-Defined Element](#) ¹⁴⁰, 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](#) ⁴⁶⁸ 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.

```
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.

9.10 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:

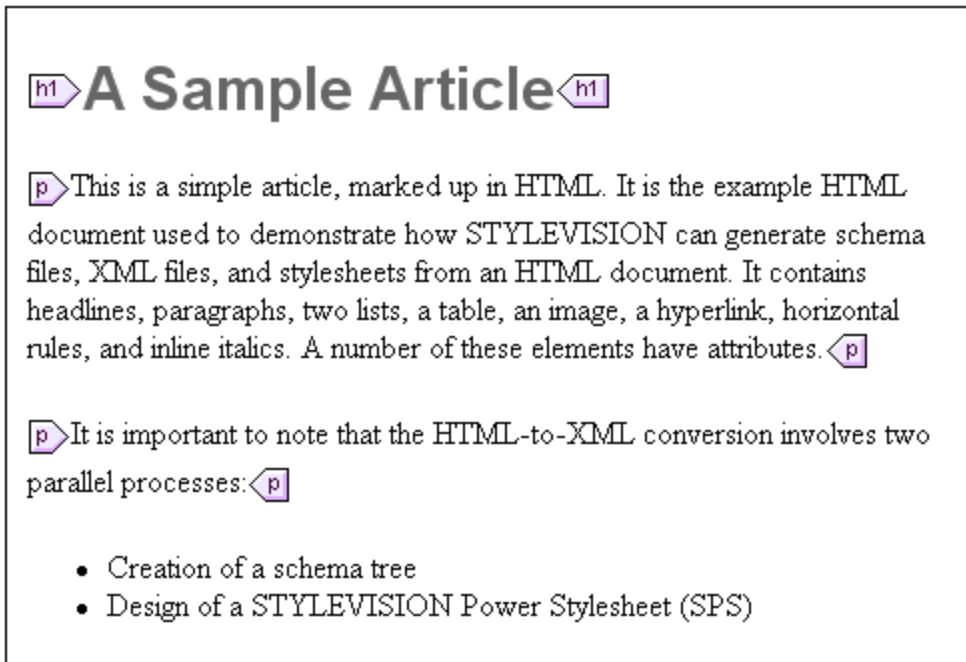
1. [Creating New SPS via HTML Import](#)⁴⁷². When an HTML file is imported into StyleVision, a new SPS document is created. The HTML document is displayed in Design View with HTML markup tags. A user-defined XML Schema with a document element called `UserRoot` is created in the Schema Tree window. This 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](#)⁴⁷⁴. 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](#)⁴⁷⁸ 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.

9.10.1 Creating New SPS via HTML Import

To create a new SPS file from an HTML document, do the following:

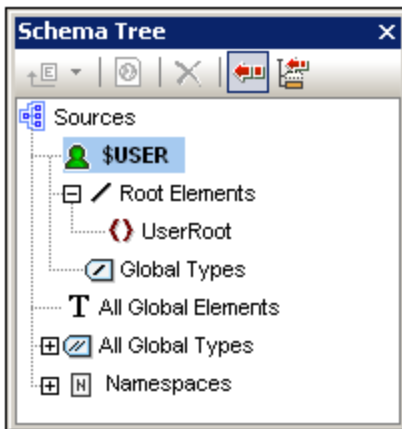
1. Select the menu command **File | New | New from HTML File**.
2. In the Open dialog that pops up, browse for the HTML file you wish to import. Select it and click **Open**.
3. You will be asked whether relative paths should be converted to absolute paths. Make your choice by clicking either **Yes** 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*).



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 `(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 `(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 `(content)` placeholder for this node is explicitly inserted in the design.

The significance of the `(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).

9.10.2 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](#) ⁴⁷⁶.

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

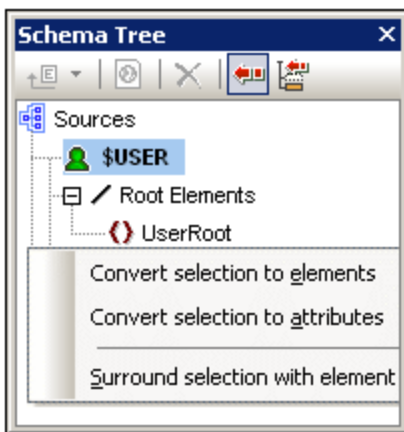
On dragging the selection over the desired schema tree node, the mouse pointer will be changed to one of the following symbols.

- 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](#) ⁴⁷⁶.

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.

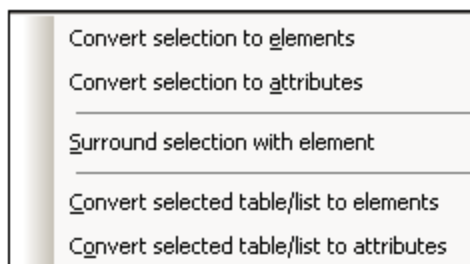
9.10.3 Creating Tables and Lists as Elements/Attributes

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.

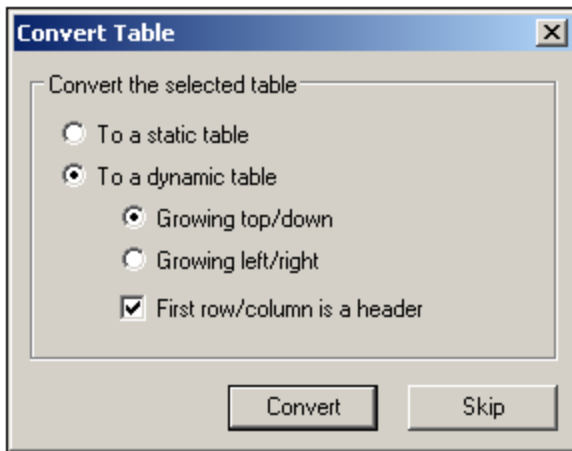
Converting a table to elements/attributes

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.



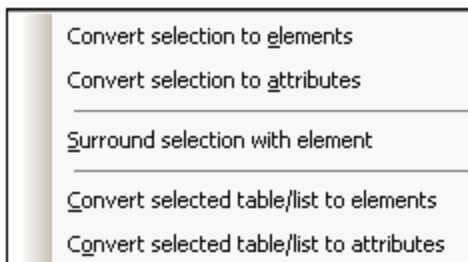
If the **static table** option is selected, then for each cell in the table, a schema node is created. In the design, each node is inserted with the `(content)` placeholder. The data in the table cells is copied to the temporary internal XML document (and to the generated XML document). The **dynamic 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.

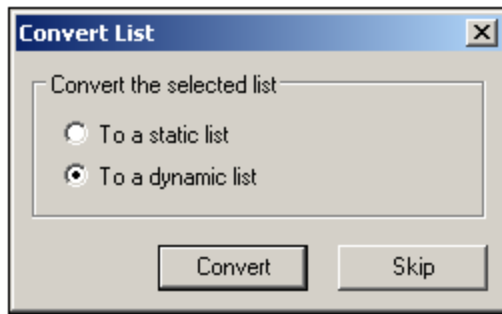
Converting a list to elements/attributes

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.



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.



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.

9.10.4 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, Text, and RTF output.
- HTML, Text, and RTF output.

9.11 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.

9.11.1 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.

9.12 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 (PXF) 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.

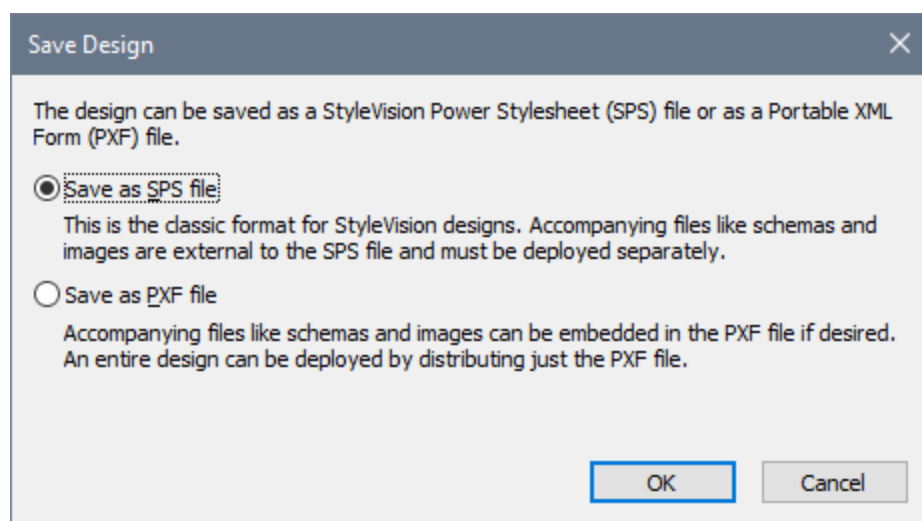
This section describing the use of PXF files is organized in two parts:

- [Creating a PXF file](#) ⁴⁸²
- [Editing a PXF File](#) ⁴⁸⁵
- [Deploying a PXF file](#) ⁴⁸⁶

Note: PXF files can be created only from SPSs designed with XSLT 2.0 or 3.0.

9.12.1 Creating a PXF File

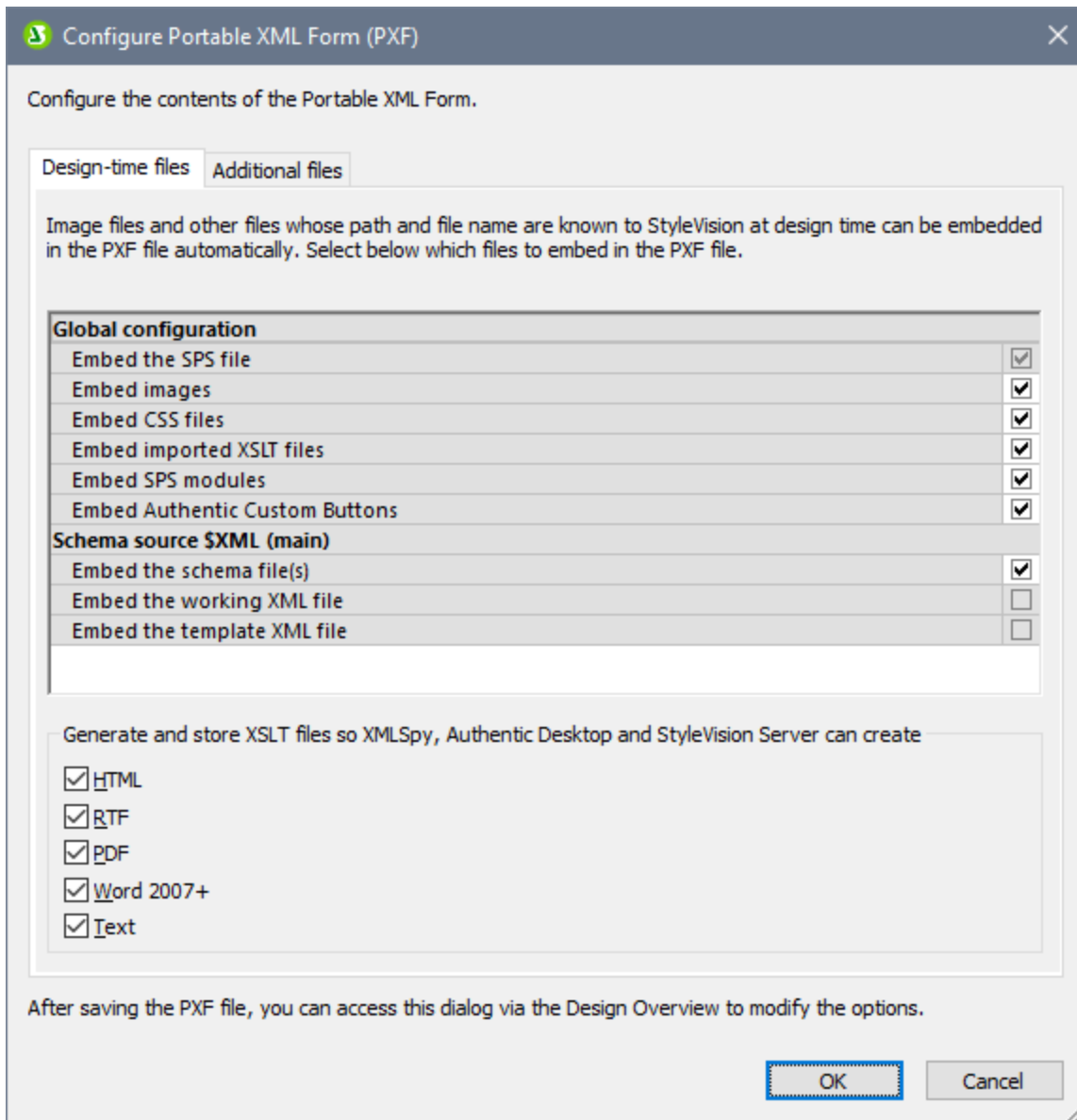
To create a PXF file that will contain an SPS design plus related files, open the SPS design in StyleVision and select the command **File | Save As**. This pops up the Save Design dialog (*screenshot below*).



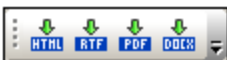
The **SPS format** is the standard Altova format for StyleVision designs. In this section we are concerned with the PXF format and so will not consider the SPS format here. Saving a file as an SPS is described in detail in the [User Reference section](#) ⁷³⁰.

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](#) ⁷²⁵ [command](#) ⁷²⁵—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.



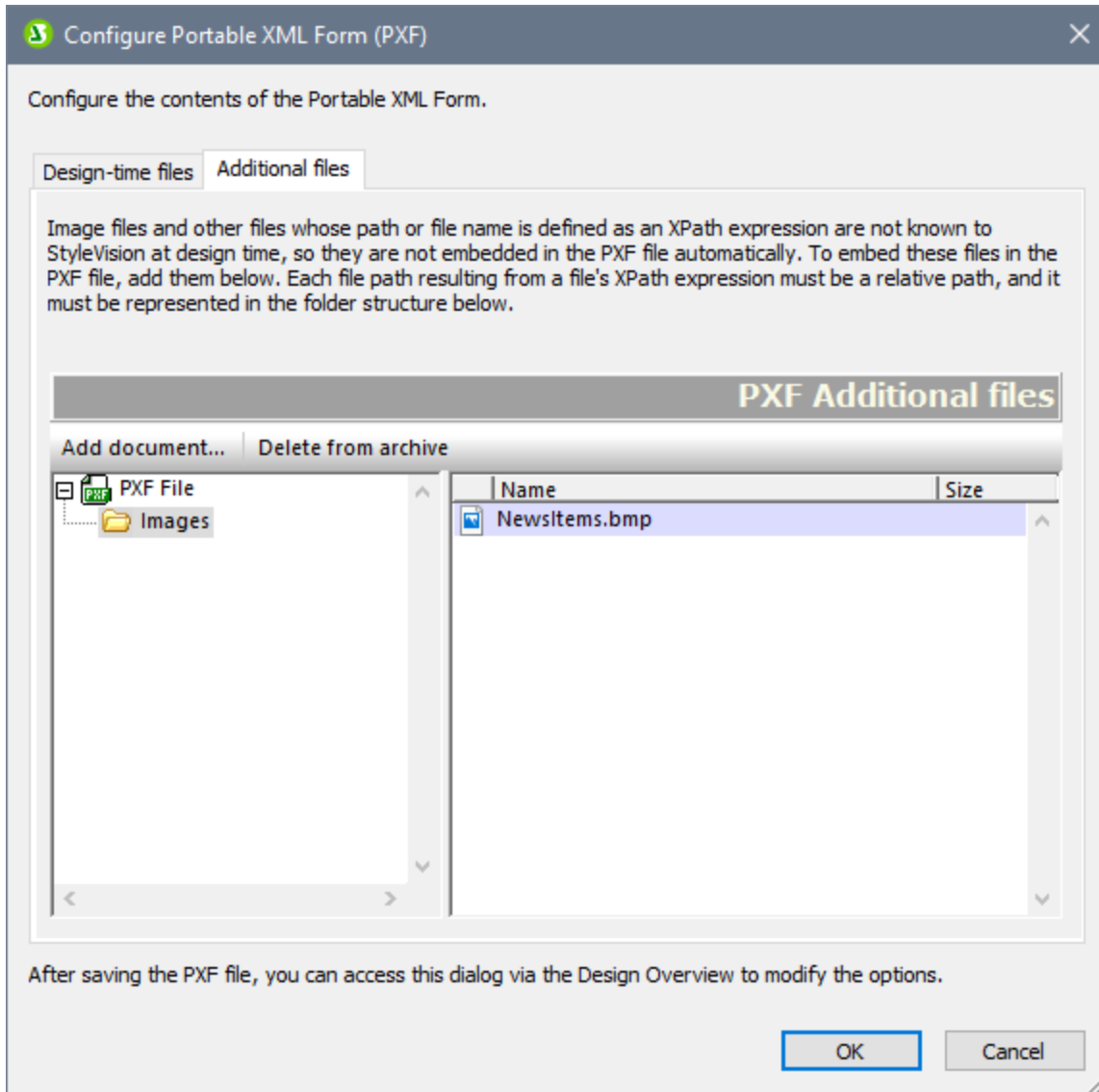
In the Global Configuration pane of the *Design-time Files* tab, you can select/deselect the design-related source files to be embedded/omitted. You can additionally choose to embed XSLT files generated from the design. In the XSLT files pane, select the output formats for which you wish to generate 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*).



Note: If XSLT files for outputs supported only in a higher edition of StyleVision (*high to low: Enterprise, Professional, Basic*) were created in a PXF file and if that PXF file is then opened in a lower edition, then on saving the PXF file the XSLT files for outputs not supported in the lower edition will not be saved. A prompt appears, asking whether you wish to continue saving the PXF file. You can then save

without the unsupported formats, or abort the save and retain the unsupported formats.

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.

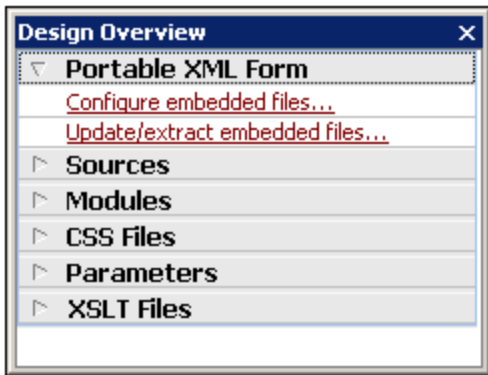
9.12.2 Editing a PXF File

A PXF file can be opened in StyleVision via the [File | Open](#) ⁷²⁰ 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*).

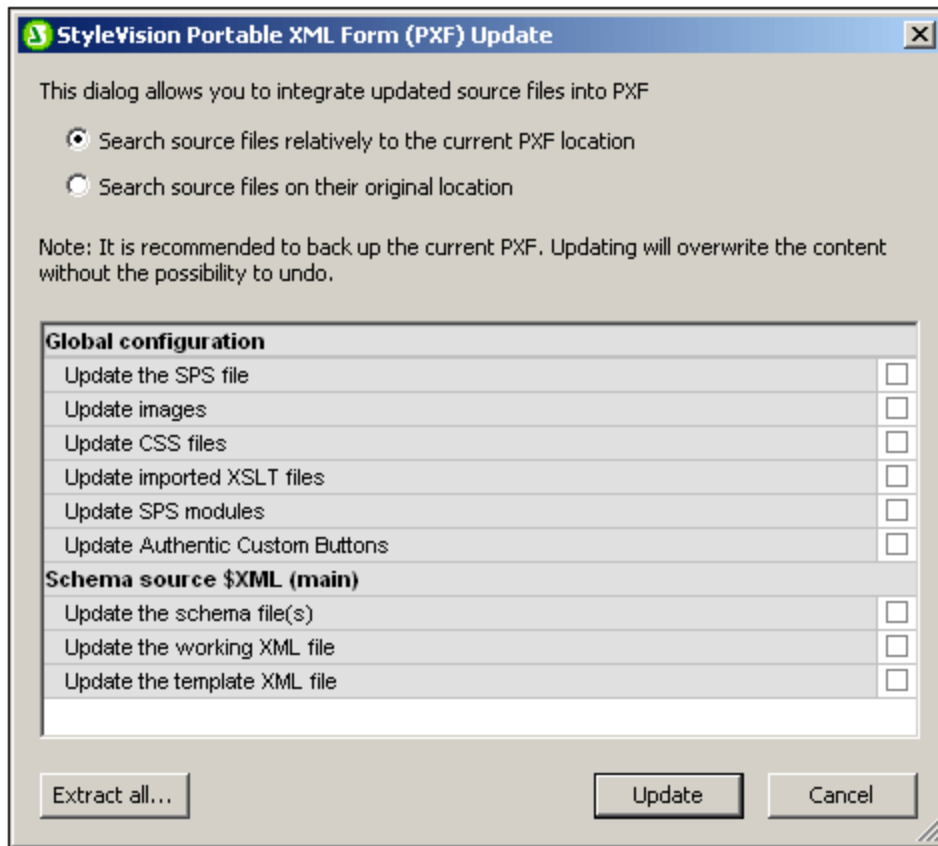


Configure embedded files

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](#) ⁴⁸².

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.

9.12.3 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](#)²⁴. 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 Plug-in, 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`.

10 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](#)²⁴.
- 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, Text, and RTF output.
- To generate DB reports (based on the SPS design) in HTML, Text, and RTF 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](#)⁴⁹⁰, which provides an [overview of how DBs work in StyleVision](#)⁴⁹⁰, 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](#)⁴⁹²: Describes how to connect to non-XML DBs, including IBM DB2 versions below 9..
- [DB Data Selection](#)⁵⁷⁵: 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](#)⁵⁸⁶: 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](#)⁵⁸⁹: DB data that is loaded into the temporary XML file can be filtered.
- [SPS Design Features for DB](#)⁵⁹⁴: In the SPS, special DB functionality, such as DB controls and DB Queries, are available.
- [Generating Output Files](#)⁵⁹⁸: A wide range of DB report-related files can be generated by StyleVision.

Supported databases

The table below lists all the supported databases. 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.

Database	Notes
Firebird 2.x, 3.x, 4.x	
IBM DB2 8.x, 9.x, 10.x, 11.x	

Database	Notes
IBM Db2 for i 6.x, 7.4, 7.5	Logical files are supported and shown as views.
IBM Informix 11.70 and later	
MariaDB 10 and later	MariaDB supports native connections. No separate drivers are required.
Microsoft Access 2003 and later	At the time of writing (early September 2019), there is no Microsoft Access Runtime available for Access 2019. You can connect to an Access 2019 database from Altova products only if Microsoft Access 2016 Runtime is installed and only if the database does not use the "Large Number" data type.
Microsoft Azure SQL Database	SQL Server 2016 codebase
Microsoft SQL Server 2005 and later Microsoft SQL Server on Linux	
MySQL 5 and later	MySQL 5.7 and later supports native connections. No separate drivers are required.
Oracle 9i and later	
PostgreSQL 8 and later	PostgreSQL connections are supported both as native connections and driver-based connections through interfaces (drivers) such as ODBC or JDBC. Native connections do not require any drivers.
Progress OpenEdge 11.6	
SQLite 3.x	SQLite connections are supported as native, direct connections to the SQLite database file. No separate drivers are required. In Authentic view, data coming from a SQLite database is not editable. When you attempt to save SQLite data from the Authentic view, a message box will inform you of this known limitation.
Sybase ASE 15, 16	
Teradata 16	

 **See also**

- [What Is an SPS?](#) ²³

Altova website:  [Database Reporting](#)

10.1 DBs and StyleVision

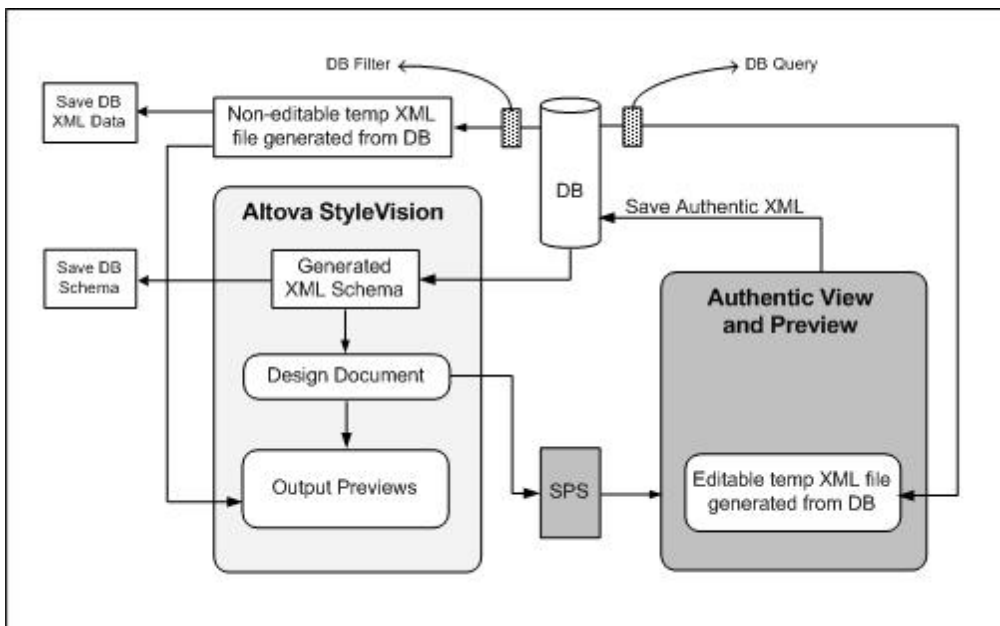
In StyleVision, you can create DB-based SPSs. These stylesheets enable you to do two things:

- 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 and RTF 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 Text output (based on design of SPS)
- XSLT stylesheet for RTF output (based on design of SPS)
- HTML output
- Text output
- RTF 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, Text, and RTF reports from the DB.

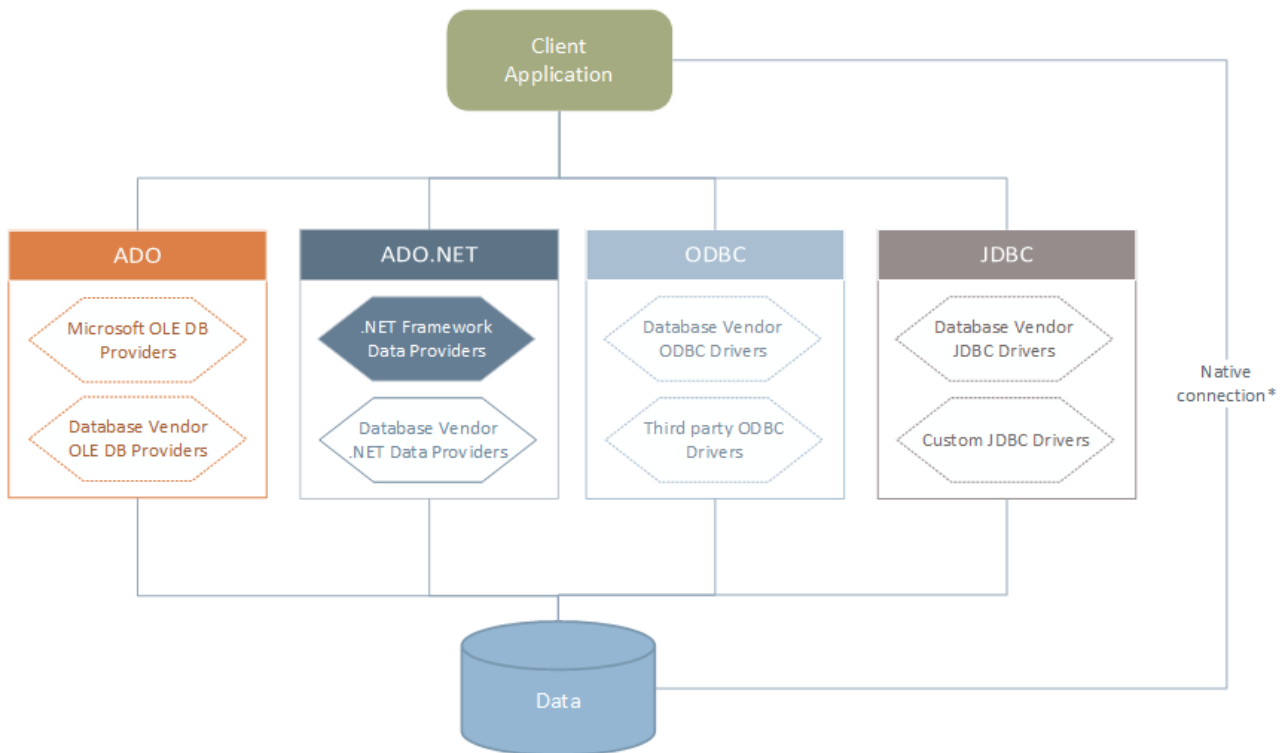
- [Connect to the DB](#) ⁴⁹² with StyleVision. During the connection process you can specify what data tables in the DB should be filtered out from the XML Schema..
- When the connection is made, a [temporary XML Schema](#) ⁵⁸⁶ is generated based on the structure of the DB and that schema is displayed 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](#) ⁵⁸⁶. 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](#) ⁵⁸⁹ to restrict the data imported into the non-editable XML File, i.e. for the output views and the reports.
- A [DB Query](#) ⁵⁹⁴ 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.

10.2 Connecting to a Data Source

In the most simple case, a database can be a local file such as a Microsoft Access or SQLite database file. In a more advanced scenario, a database may reside on a remote or network database server which does not necessarily use the same operating system as the application that connects to it and consumes data. For example, while StyleVision runs on a Windows operating system, the database from which you want to access data (for example, MySQL) might run on a Linux machine.

To interact with various database types, both remote and local, StyleVision relies on the data connection interfaces and database drivers that are already available on your operating system or released periodically by the major database vendors. In the constantly evolving landscape of database technologies, this approach caters for better cross-platform flexibility and interoperability.

The following diagram illustrates, in a simplified way, data connectivity options available between StyleVision (illustrated as a generic client application) and a data store (which may be a database server or database file).



* Direct native connections are supported for SQLite, MySQL, MariaDB, PostgreSQL databases. To connect to such databases, no additional drivers are required to be installed on your system.

As shown in the diagram above, StyleVision can access any of the major database types through the following data access technologies:

- ADO (Microsoft® ActiveX® Data Objects), which, in its turn, uses an underlying OLE DB (Object Linking and Embedding, Database) provider
- ADO.NET (A set of libraries available in the Microsoft .NET Framework that enable interaction with data)

- JDBC (Java Database Connectivity)
- ODBC (Open Database Connectivity)

Note: Some ADO.NET providers are not supported or have limited support. See [ADO.NET Support Notes](#)⁶⁰⁹.

About data access technologies

The data connection interface you should choose largely depends on your existing software infrastructure. You will typically choose the data access technology and the database driver which integrates tighter with the database system to which you want to connect. For example, to connect to a Microsoft Access 2013 database, you would build an ADO connection string that uses a native provider such as the **Microsoft Office Access Database Engine OLE DB Provider**. To connect to Oracle, on the other hand, you may want to download and install the latest JDBC, ODBC, or ADO.NET interfaces from the Oracle website.

While drivers for Windows products (such as Microsoft Access or SQL Server) may already be available on your Windows operating system, they may not be available for other database types. Major database vendors routinely release publicly available database client software and drivers which provide cross-platform access to the respective database through any combination of ADO, ADO.NET, ODBC, or JDBC. In addition to this, several third party drivers may be available for any of the above technologies. In most cases, there is more than one way to connect to the required database from your operating system, and, consequently, from StyleVision. The available features, performance parameters, and the known issues will typically vary based on the data access technology or drivers used.

10.2.1 Start Database Connection Wizard

StyleVision provides a Database Connection Wizard that guides you through the steps required to set up a connection to a data source. Before you go through the wizard steps, be aware that for some database types it is necessary to install and separately configure several database prerequisites, such as a database driver or database client software. These are normally provided by the respective database vendors, and include documentation tailored to your specific Windows version. For a list of database drivers grouped by database type, see [Database Drivers Overview](#)⁴⁹⁵.

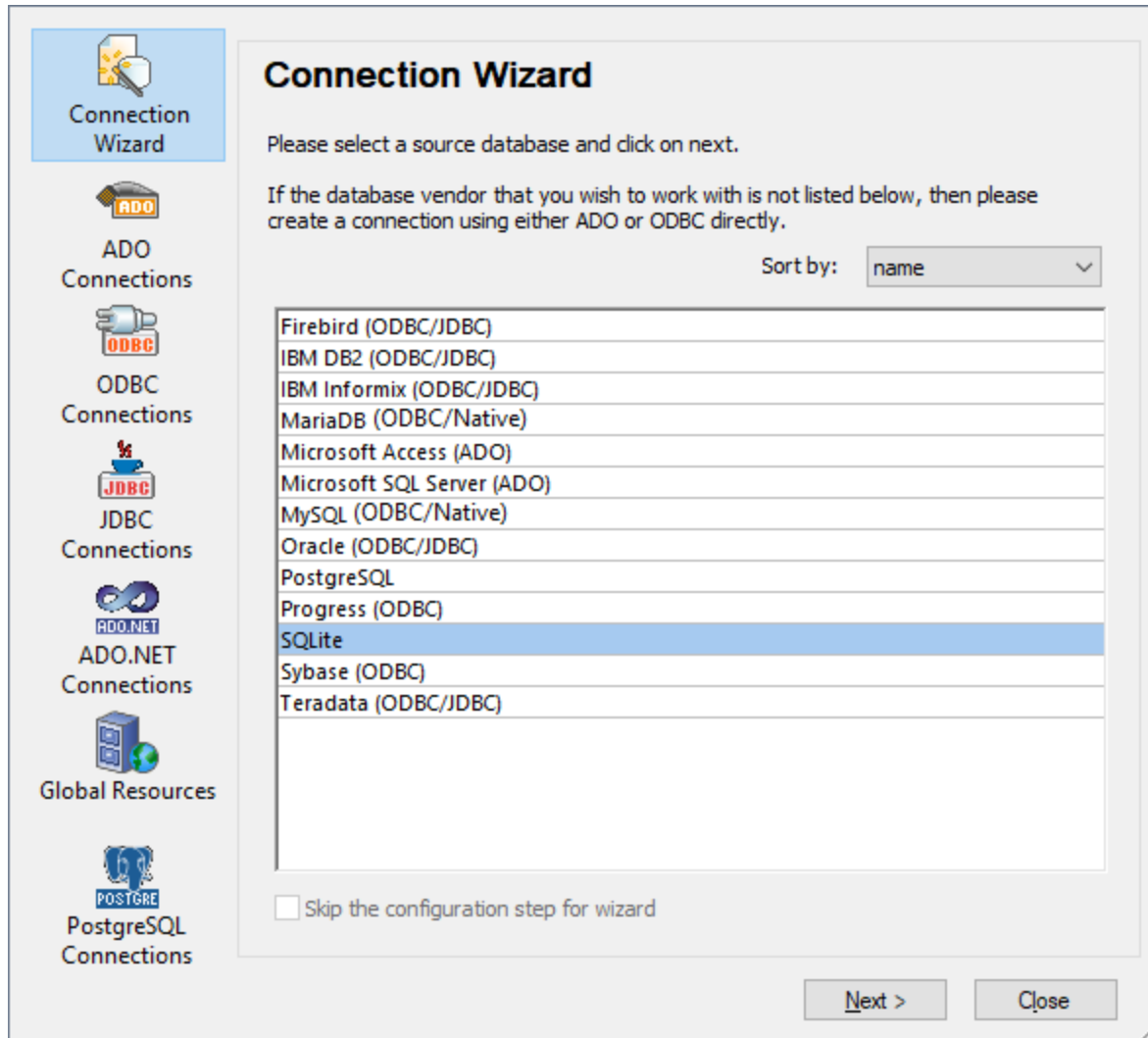
To start the Database Connection Wizard (see *screenshot below*), do the following:

- On the **File** menu, click **New**, and then click **New From DB**.

The Database Connection Wizard (*screenshot below*) is started. On the left hand side of the window, you can select the most suitable from the following ways to connect to your database:

- Connection Wizard, which prompts you to choose your database type and then guides you through the steps for connecting to a database of that type
- Select a data access technology: ADO, ADO.NET, ODBC, or JDBC
- Use an Altova global resource in which database connection is stored
- A native PostgreSQL connection

In the Connection Wizard pane (see *screenshot below*) databases can be sorted alphabetically by the name of the database type or by recent usage. Select the option you want in the *Sort By* combo box. After you have selected the database type to which you want to connect, click **Next**.



The wizard will take you through the next steps according to the database type, connection technology (ADO, ADO.NET, ODBC, JDBC), and driver that will be used. For examples applicable to each database type, see [Database Connection Examples](#)⁵²¹.

Alternatively to using Connection Wizard, you can use one of the following database access technologies:

- [Setting up an ADO Connection](#)⁴⁹⁸
- [Setting up an ADO.NET Connection](#)⁵⁰³
- [Setting up an ODBC Connection](#)⁵¹⁰
- [Setting up a JDBC Connection](#)⁵¹³

10.2.2 Database Drivers Overview

The following table lists common database drivers you can use to connect to a particular database through a particular data access technology. Note that this list does not aim to be either exhaustive or prescriptive; you can use other native or third party alternatives in addition to the drivers shown below.

Even though a number of database drivers might be already available on your Windows operating system, you may still need to download an alternative driver. For some databases, the latest driver supplied by the database vendor is likely to perform better than the driver that shipped with the operating system.

Database vendors may provide drivers either as separate downloadable packages, or bundled with database client software. In the latter case, the database client software normally includes any required database drivers, or provides you with an option during installation to select the drivers and components you wish to install. Database client software typically consists of administration and configuration utilities used to simplify database administration and connectivity, as well as documentation on how to install and configure the database client and any of its components.

Configuring the database client correctly is crucial for establishing a successful connection to the database. Before installing and using the database client software, it is strongly recommended to read carefully the installation and configuration instructions of the database client; these may vary for each database version and for each Windows version.

To understand the capabilities and limitations of each data access technology with respect to each database type, refer to the documentation of that particular database product and also test the connection against your specific environment. To avoid common connectivity issues, note the following:

- Some ADO.NET providers are not supported or have limited support. See [ADO.NET Support Notes](#) ⁵⁰⁹.
- When installing a database driver, it is recommended that it has the same platform as the Altova application (32-bit or 64-bit). For example, if you are using a 32-bit Altova application on a 64-bit operating system, install the 32-bit driver, and set up your database connection using the 32-bit driver, see also [Viewing the Available ODBC Drivers](#) ⁵¹².
- When setting up an ODBC data source, it is recommended to create the data source name (DSN) as System DSN instead of User DSN. For more information, see [Setting up an ODBC Connection](#) ⁵¹⁰.
- When setting up a JDBC data source, ensure that JRE (Java Runtime Environment) or Java Development Kit (JDK) is installed and that the CLASSPATH environment variable of the operating system is configured. For more information, see [Setting up a JDBC Connection](#) ⁵¹³.
- For the installation instructions and support details of any drivers or database client software that you install from a database vendor, check the documentation provided with the installation package.

Database	Interface	Drivers
Firebird	ADO.NET	Firebird ADO.NET Data Provider (https://www.firebirdsql.org/en/additional-downloads/)
	JDBC	Firebird JDBC driver (https://www.firebirdsql.org/en/jdbc-driver/)
	ODBC	Firebird ODBC driver (https://www.firebirdsql.org/en/odbc-driver/)
IBM DB2	ADO	IBM OLE DB Provider for DB2
	ADO.NET	IBM Data Server Provider for .NET

Database	Interface	Drivers
	JDBC	IBM Data Server Driver for JDBC and SQLJ
	ODBC	IBM DB2 ODBC Driver
IBM DB2 for i	ADO	<ul style="list-style-type: none"> • IBM DB2 for i5/OS IBMDA400 OLE DB Provider • IBM DB2 for i5/OS IBMDARLA OLE DB Provider • IBM DB2 for i5/OS IBMDASQL OLE DB Provider
	ADO.NET	.NET Framework Data Provider for IBM i
	JDBC	IBM Toolbox for Java JDBC Driver
	ODBC	iSeries Access ODBC Driver
IBM Informix	ADO	IBM Informix OLE DB Provider
	JDBC	IBM Informix JDBC Driver
	ODBC	IBM Informix ODBC Driver
Microsoft Access	ADO	<ul style="list-style-type: none"> • Microsoft Jet OLE DB Provider • Microsoft Access Database Engine OLE DB Provider
	ADO.NET	.NET Framework Data Provider for OLE DB
	ODBC	<ul style="list-style-type: none"> • Microsoft Access Driver
MariaDB	ADO.NET	In the absence of a dedicated .NET connector for MariaDB, use Connector/.NET for MySQL (https://dev.mysql.com/downloads/connector/net/).
	JDBC	MariaDB Connector/J (https://downloads.mariadb.org/)
	ODBC	MariaDB Connector/ODBC (https://downloads.mariadb.org/)
	Native connection	Available. No drivers are required.
Microsoft SQL Server	ADO	<ul style="list-style-type: none"> • Microsoft OLE DB Driver for SQL Server (MSOLEDBSQL) • Microsoft OLE DB Provider for SQL Server (SQLOLEDB) • SQL Server Native Client (SQLNCLI)
	ADO.NET	<ul style="list-style-type: none"> • .NET Framework Data Provider for SQL Server • .NET Framework Data Provider for OLE DB
	JDBC	<ul style="list-style-type: none"> • Microsoft JDBC Driver for SQL Server (https://docs.microsoft.com/en-us/sql/connect/jdbc/microsoft-jdbc-driver-for-sql-server)
	ODBC	<ul style="list-style-type: none"> • ODBC Driver for Microsoft SQL Server (https://docs.microsoft.com/en-us/SQL/connect/odbc/download-odbc-driver-for-sql-server)
MySQL	ADO.NET	<ul style="list-style-type: none"> • Connector/.NET (https://dev.mysql.com/downloads/connector/net/)
	JDBC	Connector/J (https://dev.mysql.com/downloads/connector/j/)

Database	Interface	Drivers
	ODBC	Connector/ODBC (https://dev.mysql.com/downloads/connector/odbc/)
	Native connection	Available for MySQL 5.7 and later. No drivers are required.
Oracle	ADO	<ul style="list-style-type: none"> • Oracle Provider for OLE DB • Microsoft OLE DB Provider for Oracle
	ADO.NET	Oracle Data Provider for .NET (http://www.oracle.com/technetwork/topics/dotnet/index-085163.html)
	JDBC	<ul style="list-style-type: none"> • JDBC Thin Driver • JDBC Oracle Call Interface (OCI) Driver These drivers are typically installed during the installation of your Oracle database client. Connect through the OCI Driver (not the Thin Driver) if you are using the Oracle XML DB component.
	ODBC	<ul style="list-style-type: none"> • Microsoft ODBC for Oracle • Oracle ODBC Driver (typically installed during the installation of your Oracle database client)
PostgreSQL	JDBC	PostgreSQL JDBC Driver (https://jdbc.postgresql.org/download.html)
	ODBC	psqlODBC (https://odbc.postgresql.org/)
	Native connection	Available. No drivers are required.
Progress OpenEdge	JDBC	JDBC Connector (https://www.progress.com/jdbc/openedge)
	ODBC	ODBC Connector (https://www.progress.com/odbc/openedge)
SQLite	Native connection	Available. No drivers are required.
Sybase	ADO	Sybase ASE OLE DB Provider
	JDBC	jConnect™ for JDBC
	ODBC	Sybase ASE ODBC Driver
Teradata	ADO.NET	.NET Data Provider for Teradata (https://downloads.teradata.com/download/connectivity/net-data-provider-for-teradata)
	JDBC	Teradata JDBC Driver (https://downloads.teradata.com/download/connectivity/jdbc-driver)
	ODBC	Teradata ODBC Driver for Windows (https://downloads.teradata.com/download/connectivity/odbc-driver/windows)

10.2.3 ADO Connection

Microsoft ActiveX Data Objects (ADO) is a data access technology that enables you to connect to a variety of data sources through OLE DB. OLE DB is an alternative interface to ODBC or JDBC; it provides uniform access to data in a COM (Component Object Model) environment. ADO is a precursor of the newer [ADO.NET](#)⁵⁰³ and is still one of the possible ways to connect to Microsoft native databases such as Microsoft Access or SQL Server, although you can also use it for other data sources.

Importantly, you can choose between multiple ADO providers, and some of them must be downloaded and installed on your workstation before you can use them. For example, for connecting to SQL Server, the following ADO providers are available:

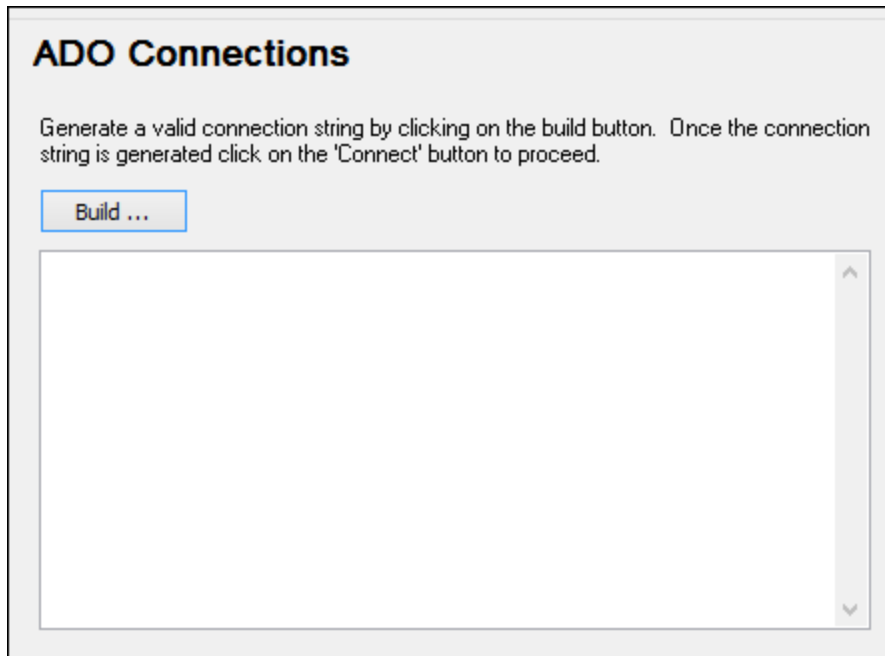
- Microsoft OLE DB *Driver* for SQL Server (MSOLEDBSQL)
- Microsoft OLE DB *Provider* for SQL Server (SQLOLEDB)
- SQL Server Native Client (SQLNCLI)

From the providers listed above, the recommended one is MSOLEDBSQL; you can download it from <https://docs.microsoft.com/en-us/sql/connect/oledb/download-oledb-driver-for-sql-server?view=sql-server-ver15>. Note that it must match the platform of StyleVision (32-bit or 64-bit). The SQLOLEDB and SQLNCLI providers are considered deprecated and thus are not recommended.

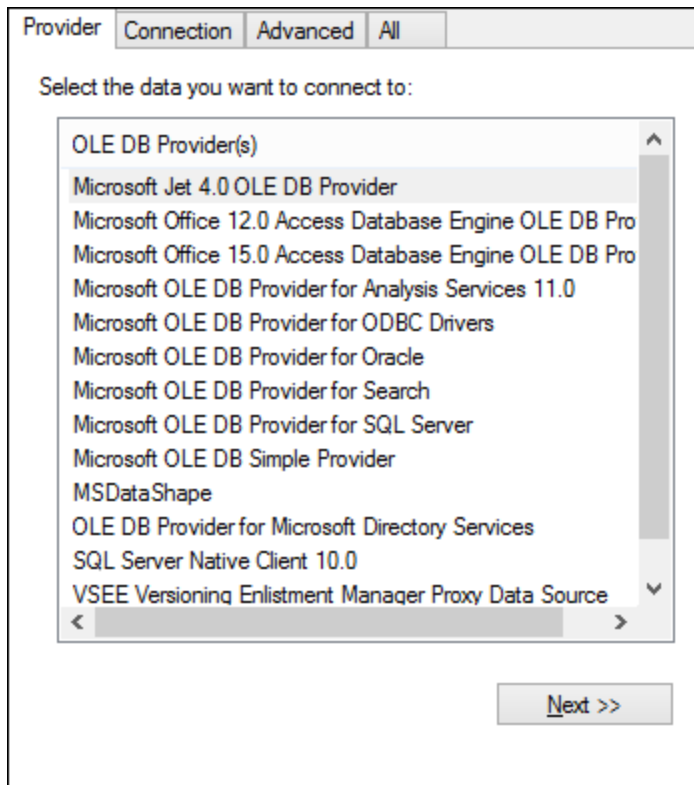
The **Microsoft OLE DB Provider for SQL Server (SQLOLEDB)** is known to have issues with parameter binding of complex queries like Common Table Expressions (CTE) and nested SELECT statements.

To set up an ADO connection:

1. [Start the database connection wizard](#)⁴⁹³.
2. Click **ADO Connections**.



3. Click **Build**.



4. Select the data provider through which you want to connect. The table below lists a few common scenarios.

To connect to this database...	Use this provider...
Microsoft Access	<ul style="list-style-type: none"> • Microsoft Office Access Database Engine OLE DB Provider (recommended) • Microsoft Jet OLE DB Provider <p>If the Microsoft Office Access Database Engine OLE DB Provider is not available in the list, make sure that you have installed either Microsoft Access or the Microsoft Access Database Engine Redistributable (https://www.microsoft.com/en-us/download/details.aspx?id=54920) on your computer.</p>
SQL Server	<ul style="list-style-type: none"> • Microsoft OLE DB Driver for SQL Server (MSOLEDBSQL) - this is the recommended OLE DB provider. In order for this provider to appear in the list, it must be downloaded from https://docs.microsoft.com/en-us/sql/connect/oledb/download-oledb-driver-for-sql-server?view=sql-server-ver15 and installed. • Microsoft OLE DB Provider for SQL Server (OLEDBSQL) • SQL Server Native Client (SQLNCLI)
Other database	<p>Select the provider applicable to your database.</p> <p>If an OLE DB provider to your database is not available, install the required driver from the database vendor (see Database Drivers Overview⁴⁹⁵). Alternatively, set up an ADO.NET, ODBC, or JDBC connection.</p> <p>If the operating system has an ODBC driver to the required database, you could also use the Microsoft OLE DB Provider for ODBC Drivers, or preferably opt for an ODBC connection⁶¹⁰.</p>

5. Having selected the provider of choice, click **Next** and complete the wizard.

The subsequent wizard steps are specific to the provider you chose. For SQL Server, you will need to provide or select the host name of the database server, the authentication method, the database name, as well as the database username and password. For an example, see [Connecting to Microsoft SQL Server \(ADO\)](#)⁶⁴³. For Microsoft Access, you will be asked to browse for or provide the path to the database file. For an example, see [Connecting to Microsoft Access \(ADO\)](#)⁵⁴⁰.

The complete list of initialization properties (connection parameters) is available in the **All** tab of the connection dialog box—these properties vary depending on the chosen provider and may need to be set explicitly in order for the connection to be possible. The following sections provide guidance on configuring the basic initialization properties for Microsoft Access and SQL Server databases:

- [Setting up the SQL Server Data Link Properties](#)⁵⁰¹
- [Setting up the Microsoft Access Data Link Properties](#)⁵⁰²

10.2.3.1 Connecting to an Existing Microsoft Access Database

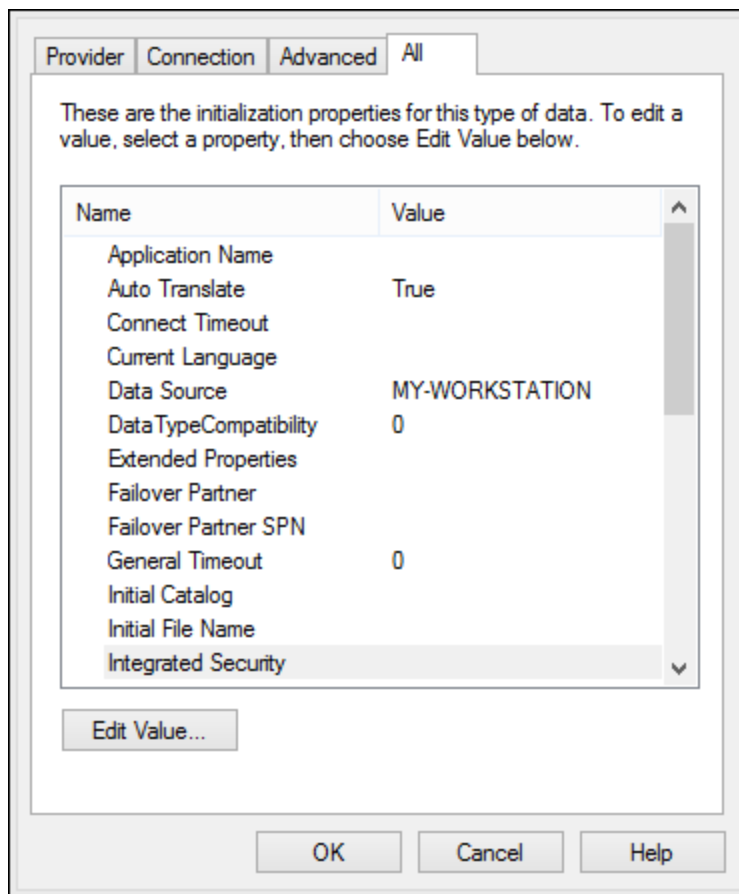
This approach is suitable when you want to connect to a Microsoft Access database which is not password-protected. If the database is password-protected, set up the database password as shown in [Connecting to Microsoft Access \(ADO\)](#)⁵⁴⁰.

To connect to an existing Microsoft Access database:

1. Run the database connection wizard (see [Starting the Database Connection Wizard](#)⁴⁹³).
2. Select **Microsoft Access (ADO)**, and then click **Next**.
3. Browse for the database file, or enter the path to it (either relative or absolute).
4. Click **Connect**.

10.2.3.2 Setting up the SQL Server Data Link Properties

When you connect to a Microsoft SQL Server database through [ADO](#)⁴⁹⁸, you may need to set the following connection properties in the **All** tab of the Data Link Properties dialog box.

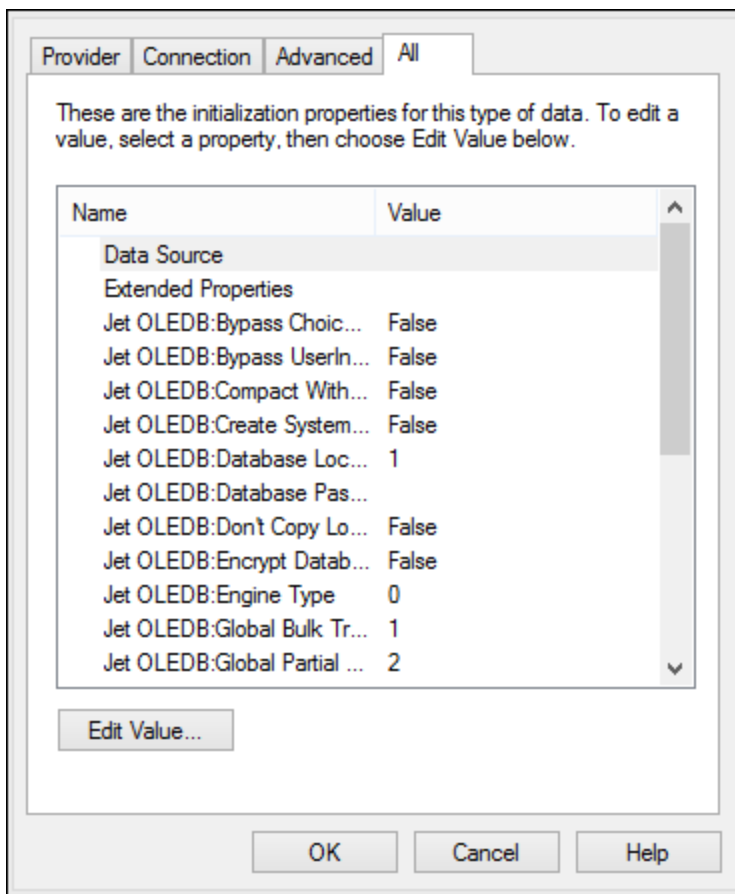


Data Link Properties dialog box

Property	Notes
Integrated Security	If you selected the SQL Server Native Client data provider on the Provider tab, set this property to a space character.
Persist Security Info	Set this property to True .

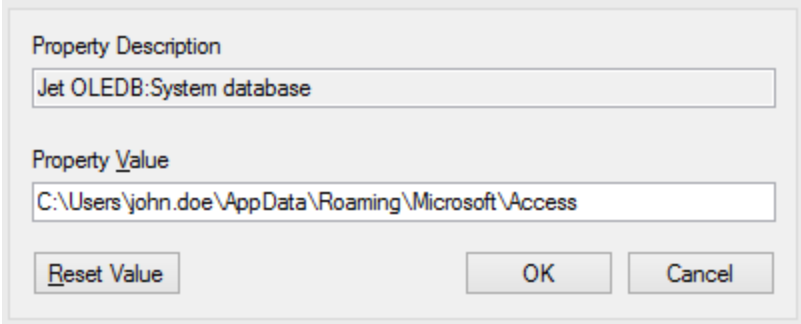
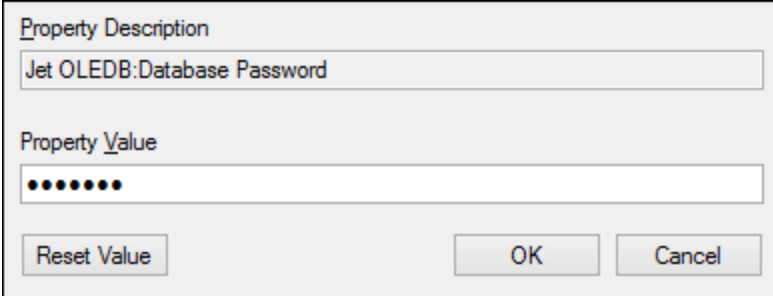
10.2.3.3 Setting up the Microsoft Access Data Link Properties

When you connect to a Microsoft Access database through [ADO](#)⁴⁹⁸, you may need to set the following connection properties in the **All** tab of the Data Link Properties dialog box.



Data Link Properties dialog box

Property	Notes
Data Source	This property stores the path to the Microsoft Access database file. To avoid database connectivity issues, it is recommended to use the UNC (Universal Naming Convention) path format, for example: <code>\\anyserver\share\$\filepath</code>

Property	Notes
Jet OLEDB:System Database	<p>This property stores the path to the workgroup information file. You may need to explicitly set the value of this property before you can connect to a Microsoft Access database.</p> <p>If you cannot connect due to a "workgroup information file" error, locate the workgroup information file (System.MDW) applicable to your user profile, and set the property value to the path of the System.MDW file.</p> 
Jet OLEDB:Database Password	<p>If the database is password-protected, set the value of this property to the database password.</p> 

10.2.4 ADO.NET Connection

ADO.NET is a set of Microsoft .NET Framework libraries designed to interact with data, including data from databases. To connect to a database from StyleVision through ADO.NET, Microsoft .NET Framework 4 or later is required. As shown below, you connect to a database through ADO.NET by selecting a .NET provider and supplying a connection string.

A .NET data provider is a collection of classes that enables connecting to a particular type of data source (for example, a SQL Server, or an Oracle database), executing commands against it, and fetching data from it. In other words, with ADO.NET, an application such as StyleVision interacts with a database through a data provider. Each data provider is optimized to work with the specific type of data source that it is designed for. There are two types of .NET providers:

1. Supplied by default with Microsoft .NET Framework.

2. Supplied by major database vendors, as an extension to the .NET Framework. Such ADO.NET providers must be installed separately and can typically be downloaded from the website of the respective database vendor.

Note: Certain ADO.NET providers are not supported or have limited support. See [ADO.NET Support Notes](#) ⁵⁰⁹.

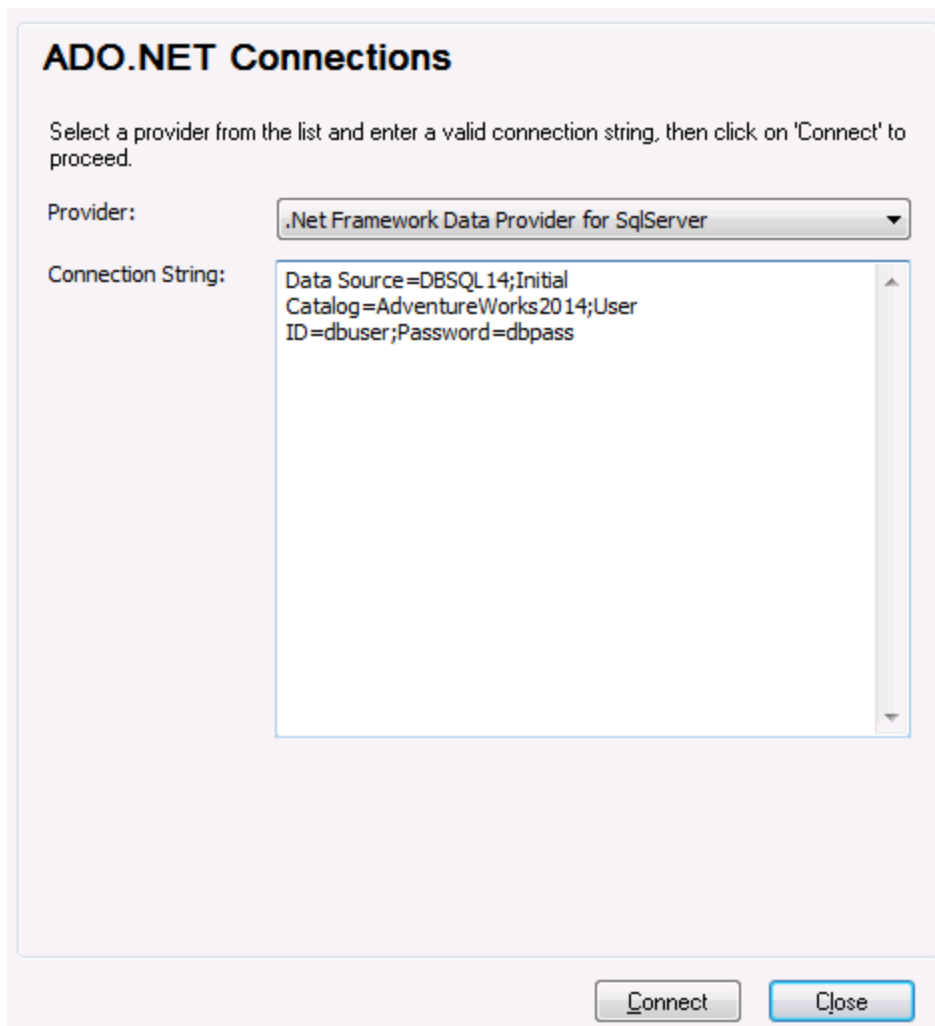
To set up an ADO.NET connection:

1. [Start the database connection wizard](#) ⁴⁹³.
2. Click **ADO.NET Connections**.
3. Select a .NET data provider from the list.

The list of providers available by default with the .NET Framework appears in the "Provider" list. Vendor-specific .NET data providers are available in the list only if they are already installed on your system. To become available, vendor-specific .NET providers must be installed into the GAC (Global Assembly Cache), by running the .msi or .exe file supplied by the database vendor.

4. Enter a database connection string. A connection string defines the database connection information, as semicolon-delimited key/value pairs of connection parameters. For example, a connection string such as `Data Source=DBSQLSERV;Initial Catalog=ProductsDB;UserID=dbuser;Password=dbpass` connects to the SQL Server database `ProductsDB` on server `DBSQLSERV`, with the user name `dbuser` and password `dbpass`. You can create a connection string by typing the key/value pairs directly into the "Connection String" dialog box. Another option is to create it with Visual Studio (see [Creating a Connection String in Visual Studio](#) ⁵⁰⁵).

The syntax of the connection string depends on the provider selected from the "Provider" list. For examples, see [Sample ADO.NET Connection Strings](#) ⁵⁰⁸.



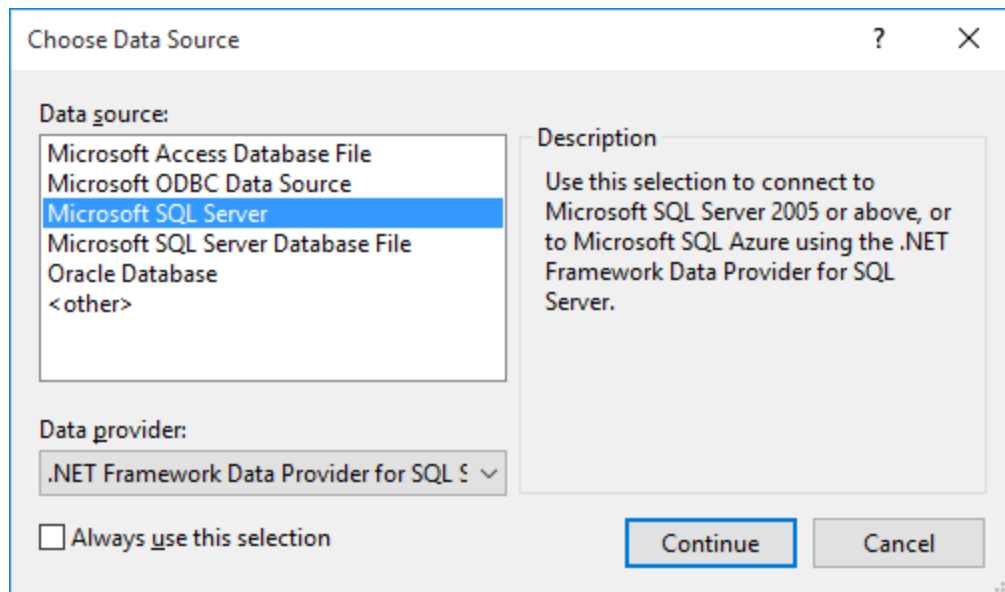
5. Click **Connect**.

10.2.4.1 Creating a Connection String in Visual Studio

In order to connect to a data source using ADO.NET, a valid database connection string is required. The following instructions show you how to create a connection string from Visual Studio.

To create a connection string in Visual Studio:

1. On the **Tools** menu, click **Connect to Database**.
2. Select a data source from the list (in this example, Microsoft SQL Server). The Data Provider is filled automatically based on your choice.



3. Click **Continue**.

Modify Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:
Microsoft SQL Server (SqlClient) Change...

Server name:
DBSQLSERV Refresh

Log on to the server

Use Windows Authentication

Use SQL Server Authentication

User name: dbuser

Password: ●●●●●●

Save my password

Connect to a database

Select or enter a database name:
ProductsDB

Attach a database file:
Browse...

Logical name:

Advanced...

Test Connection OK Cancel

4. Enter the server host name and the user name and password to the database. In this example, we are connecting to the database `ProductsDB` on server `DBSQLSERV`, using SQL Server authentication.
5. Click **OK**.

If the database connection is successful, it appears in the Server Explorer window. You can display the Server Explorer window using the menu command **View | Server Explorer**. To obtain the database connection string, right-click the connection in the Server Explorer window, and select **Properties**. The connection string is now displayed in the Properties window of Visual Studio. Note that, before pasting the string into the "Connection String" box of StyleVision, you will need to replace the asterisk (*) characters with the actual password.

10.2.4.2 Sample ADO.NET Connection Strings

To set up an ADO.NET connection, you need to select an ADO.NET provider from the database connection dialog box and enter a connection string (see also [Setting up an ADO.NET Connection](#)⁵⁰³). Sample ADO.NET connection strings for various databases are listed below under the .NET provider where they apply.

.NET Data Provider for Teradata

This provider can be downloaded from Teradata website (<https://downloads.teradata.com/download/connectivity/net-data-provider-for-teradata>). A sample connection string looks as follows:

```
Data Source=ServerAddress;User Id=USER;Password=password;
```

.NET Framework Data Provider for IBM i

This provider is installed as part of *IBM i Access Client Solutions - Windows Application Package*. A sample connection string looks as follows:

```
DataSource=ServerAddress;UserID=user;Password=password;DataCompression=True;
```

For more information, see the ".NET Provider Technical Reference" help file included in the installation package above.

.NET Framework Data Provider for MySQL

This provider can be downloaded from MySQL website (<https://dev.mysql.com/downloads/connector/net/>). A sample connection string looks as follows:

```
Server=127.0.0.1;Uid=root;Pwd=12345;Database=test;
```

See also: <https://dev.mysql.com/doc/connector-net/en/connector-net-programming-connecting-connection-string.html>

.NET Framework Data Provider for SQL Server

A sample connection string looks as follows:

```
Data Source=DBSQLSERV;Initial Catalog=ProductsDB;User ID=dbuser;Password=dbpass
```

See also: [https://msdn.microsoft.com/en-us/library/ms254500\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/ms254500(v=vs.110).aspx)

IBM DB2 Data Provider 10.1.2 for .NET Framework 4.0

```
Database=PRODUCTS;UID=user;Password=password;Server=localhost:50000;
```

Note: This provider is typically installed with the IBM DB2 Data Server Client package. If the provider is missing from the list of ADO.NET providers after installing IBM DB2 Data Server Client package, refer to the following technical note: <https://www-01.ibm.com/support/docview.wss?uid=swg21429586>.

See also:

https://www.ibm.com/support/knowledgecenter/en/SSEPGG_10.1.0/com.ibm.swg.im.dbclient.adonet.ref.doc/doc/DB2ConnectionClassConnectionStringProperty.html

Oracle Data Provider for .NET (ODP.NET)

The installation package which includes the ODP.NET provider can be downloaded from the Oracle website (see <http://www.oracle.com/technetwork/topics/dotnet/downloads/index.html>). A sample connection string looks as follows:

```
Data Source=DSORCL;User Id=user;Password=password;
```

Where `DSORCL` is the name of the data source which points to an Oracle service name defined in the `tnsnames.ora` file, as described in [Connecting to Oracle \(ODBC\)](#)⁶⁵⁵.

To connect without configuring a service name in the `tnsnames.ora` file, use a string such as:

```
Data Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=host)(PORT=port)))
(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=MyOracleSID)));User
Id=user;Password=password;
```

See also: https://docs.oracle.com/cd/B28359_01/win.111/b28375/featConnecting.htm

10.2.4.3 ADO.NET Support Notes

The following table lists known ADO.NET database drivers that are currently not supported or have limited support in StyleVision.

Database	Driver	Support notes
All databases	.Net Framework Data Provider for ODBC	Limited support. Known issues exist with Microsoft Access connections. It is recommended to use ODBC direct connections instead.
	.Net Framework Data Provider for OleDb	Limited support. Known issues exist with Microsoft Access connections. It is recommended to use ADO direct connections instead.
Firebird	Firebird ADO.NET Data Provider	Limited support. It is recommended to use ODBC or JDBC instead.
Informix	IBM Informix Data Provider for	Not supported. Use DB2 Data Server

Database	Driver	Support notes
	.NET Framework 4.0	Provider instead.
IBM DB2 for i (iSeries)	.Net Framework Data Provider for i5/OS	Not supported. Use .Net Framework Data Provider for IBM i instead, installed as part of the <i>IBM i Access Client Solutions - Windows Application Package</i> .
Oracle	.Net Framework Data Provider for Oracle	Limited support. Although this driver is provided with the .NET Framework, its usage is discouraged by Microsoft, because it is deprecated.
PostgreSQL	-	No ADO.NET drivers for this vendor are supported. Use a native connection instead.
Sybase	-	No ADO.NET drivers for this vendor are supported.

10.2.5 ODBC Connection

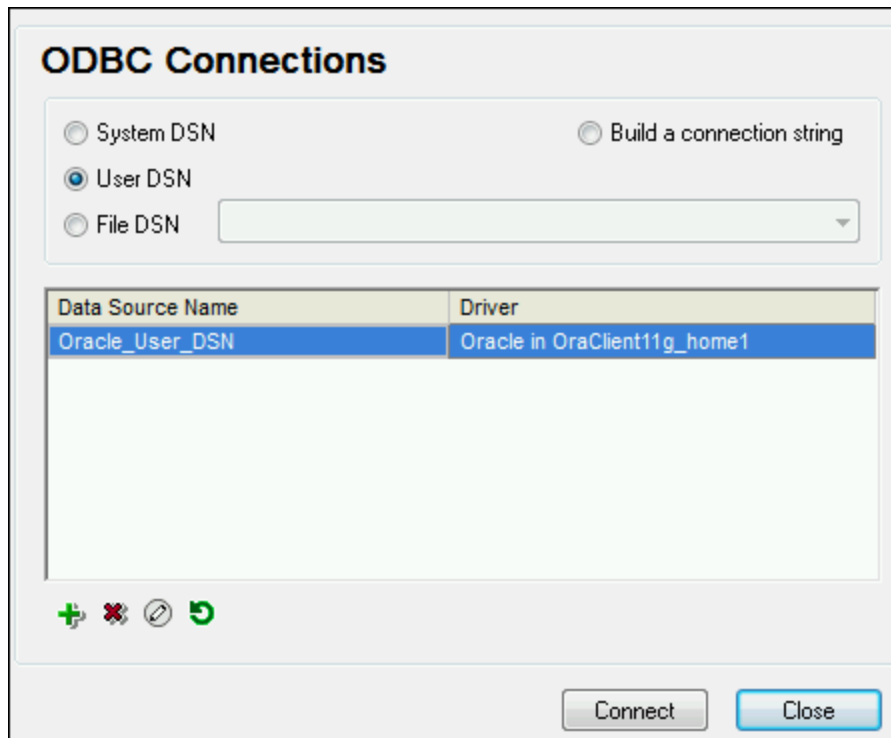
ODBC (Open Database Connectivity) is a widely used data access technology that enables you to connect to a database from StyleVision. It can be used either as primary means to connect to a database, or as an alternative to native, OLE DB, or JDBC-driven connections.

To connect to a database through ODBC, first you need to create an ODBC data source name (DSN) on the operating system. This step is not required if the DSN has already been created, perhaps by another user of the operating system. The DSN represents a uniform way to describe the database connection to any ODBC-aware client application on the operating system, including StyleVision. DSNs can be of the following types:

- System DSN
- User DSN
- File DSN

A *System* data source is accessible by all users with privileges on the operating system. A *User* data source is available to the user who created it. Finally, if you create a *File DSN*, the data source will be created as a file with the *.dsn* extension which you can share with other users, provided that they have installed the drivers used by the data source.

Any DSNs already available on your machine are listed by the database connection dialog box when you click **ODBC connections** on the ODBC connections dialog box.




ODBC Connections dialog box

If a DSN to the required database is not available, the StyleVision database connection wizard will assist you to create it; however, you can also create it directly on your Windows operating system. In either case, before you proceed, ensure that the ODBC driver applicable for your database is in the list of ODBC drivers available to the operating system (see [Viewing the Available ODBC Drivers](#)⁵¹²).

To connect by using a new DSN:

1. [Start the database connection wizard](#)⁴⁹³.
2. On the database connection dialog box, click **ODBC Connections**.
3. Select a data source type (User DSN, System DSN, File DSN).

To create a System DSN, you need administrative rights on the operating system, and StyleVision must be run as administrator.

4. Click **Add** .
5. Select a driver, and then click **User DSN** or **System DSN** (depending on the type of the DSN you want to create). If the driver applicable to your database is not listed, download it from the database vendor and install it (see [Database Drivers Overview](#)⁴⁹⁵).
6. On the dialog box that pops up, fill in any driver specific connection information to complete the setup.

For the connection to be successful, you will need to provide the host name (or IP address) of the database server, as well as the database username and password. There may be other optional connection parameters—these parameters vary between database providers. For detailed information about the parameters specific to

each connection method, consult the documentation of the driver provider. Once created, the DSN becomes available in the list of data source names. This enables you to reuse the database connection details any time you want to connect to the database. Note that User DSNs are added to the list of User DSNs whereas System DSNs are added to the list of System DSNs.

To connect by using an existing DSN:

1. [Start the database connection wizard](#) ⁴⁹³.
2. Click **ODBC Connections**.
3. Choose the type of the existing data source (User DSN, System DSN, File DSN).
4. Click the existing DSN record, and then click **Connect**.

To build a connection string based on an existing .dsn file:

1. [Start the database connection wizard](#) ⁴⁹³.
2. Click **ODBC Connections**.
3. Select **Build a connection string**, and then click **Build**.
4. If you want to build the connection string using a File DSN, click the **File Data Source** tab. Otherwise, click the **Machine Data Source** tab. (System DSNs and User DSNs are known as "Machine" data sources.)
5. Select the required .dsn file, and then click **OK**.

To connect by using a prepared connection string:

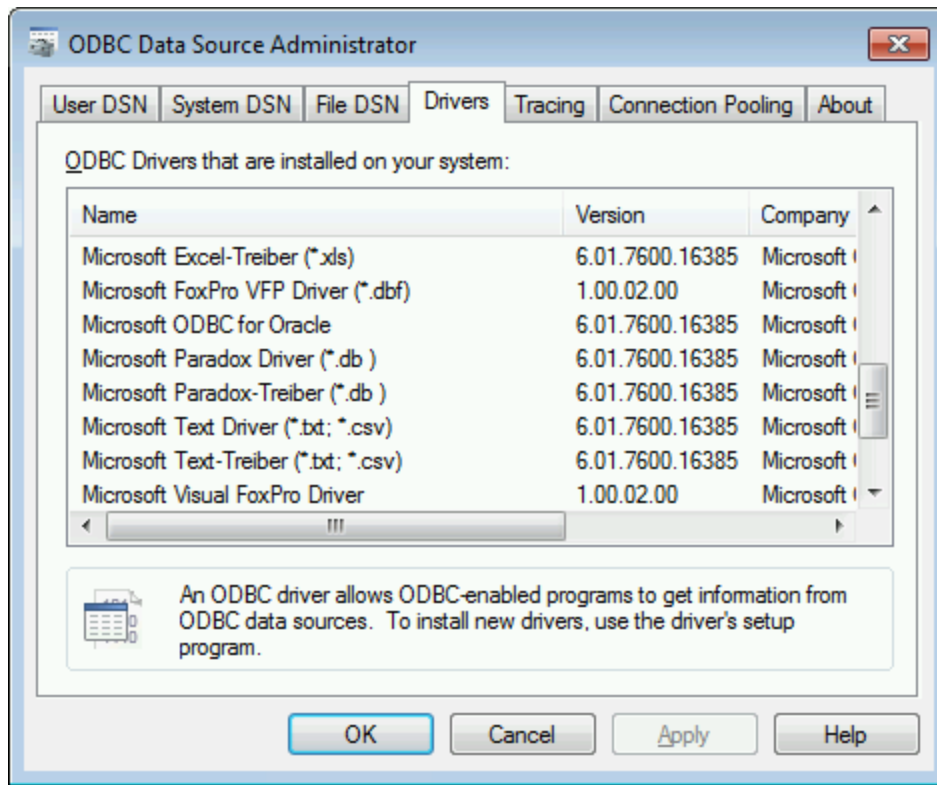
1. [Start the database connection wizard](#) ⁴⁹³.
2. Click **ODBC Connections**.
3. Select **Build a connection string**.
4. Paste the connection string into the provided box, and then click **Connect**.

10.2.5.1 Available ODBC Drivers

You can view the ODBC drivers available on your operating system in the ODBC Data Source Administrator. You can access the ODBC Data Source Administrator (**Odbcad32.exe**) from the Windows Control Panel, under **Administrative Tools**. On 64-bit operating systems, there are two versions of this executable:

- The 32-bit version of the **Odbcad32.exe** file is located in the **C:\Windows\SysWoW64** directory (assuming that **C:** is your system drive).
- The 64-bit version of the **Odbcad32.exe** file is located in the **C:\Windows\System32** directory.

Any installed 32-bit database drivers are visible in the 32-bit version of ODBC Data Source Administrator, while 64-bit drivers—in the 64-bit version. Therefore, ensure that you check the database drivers from the relevant version of ODBC Data Source Administrator.



ODBC Data Source Administrator

If the driver to your target database does not exist in the list, or if you want to add an alternative driver, you will need to download it from the database vendor (see [Database Drivers Overview](#)⁴⁹⁵). Once the ODBC driver is available on your system, you are ready to create ODBC connections with it (see [Setting up an ODBC Connection](#)⁵¹⁰).

10.2.6 JDBC Connection

JDBC (Java Database Connectivity) is a database access interface which is part of the Java software platform from Oracle. JDBC connections are generally more resource-intensive than ODBC connections but may provide features not available through ODBC.

Prerequisites

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. StyleVision will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.
- Make sure that the platform of StyleVision (32-bit, 64-bit) matches that of the JRE/JDK.
- The JDBC drivers from the database vendor must be installed. These may be JDBC drivers installed as part of a database client installation, or JDBC libraries (.jar files) downloaded separately, if available and supported by the database, see also [Database Connection Examples](#)⁵²¹.

- The `CLASSPATH` environment variable must include the path to the JDBC driver (one or several `.jar` files) on your Windows operating system. When you install some database clients, the installer may configure this variable automatically. See also [Configuring the CLASSPATH](#)⁵¹⁶.

Connecting to SQL Server via JDBC with Windows credentials

If you connect to SQL Server through JDBC with Windows credentials (integrated security), note the following:

- The `sqljdbc_auth.dll` file included in the JDBC driver package must be copied to a directory that is on the system `PATH` environment variable. There are two such files, one for the x86 and one for x64 platform. Make sure that you add to the `PATH` the one that corresponds to your JDK platform.
- The JDBC connection string must include the property `integratedSecurity=true`.

For further information, refer to *Microsoft JDBC driver for SQL Server* documentation, <https://docs.microsoft.com/en-us/sql/connect/jdbc/building-the-connection-url>.

Setting up a JDBC connection

1. [Start the database connection wizard](#)⁴⁹³.
2. Click **JDBC Connections**.
3. Optionally, enter a semicolon-separated list of `.jar` file paths in the "Classpaths" text box. The `.jar` libraries entered here will be loaded into the environment in addition to those already defined in the `CLASSPATH` environment variable. When you finish editing the "Classpaths" text box, any JDBC drivers found in the source `.jar` libraries are automatically added to the "Driver" list (see the next step).

The screenshot shows a dialog box for configuring a JDBC connection. It has the following fields and values:

- Classpaths:** `C:\jdbc\instantclient_12_1\ojdbc7.jar`
- Driver:** `oracle.jdbc.driver.OracleDriver`
- Username:** `johndoe`
- Password:** Masked with 7 dots.
- Database URL:** `jdbc:oracle:thin:@//ora12c:1521/orcl12c`

At the bottom of the dialog are two buttons: **Connect** and **Close**.

- Next to "Driver", select a JDBC driver from the list, or enter a Java class name. Note that this list contains any JDBC drivers configured through the CLASSPATH environment variable (see [Configuring the CLASSPATH](#) ⁵¹⁶), as well as those found in the "Classpaths" text box.

The JDBC driver paths defined in the CLASSPATH variable, as well as any .jar file paths entered directly in the database connection dialog box are all supplied to the Java Virtual Machine (JVM). The JVM then decides which drivers to use in order to establish a connection. It is recommended to keep track of Java classes loaded into the JVM so as not to create potential JDBC driver conflicts and avoid unexpected results when connecting to the database.

- Enter the username and password to the database in the corresponding boxes.
- In the Database URL text box, enter the JDBC connection URL (string) in the format specific to your database type. The following table describes the syntax of JDBC connection URLs (strings) for common database types.

Database	JDBC Connection URL
Firebird	<code>jdbc:firebirdsql://<host>[:<port>]/<database path or alias></code>
IBM DB2	<code>jdbc:db2://<hostName>:<port>/<databaseName></code>
IBM DB2 for i	<code>jdbc:as400://<host></code>
IBM Informix	<code>jdbc:informix-sqli://<hostName>:<port>/<databaseName>:INFORMIXSERVER=<myserver></code>
MariaDB	<code>jdbc:mariadb://<hostName>:<port>/<databaseName></code>
Microsoft SQL Server	<code>jdbc:sqlserver://<hostName>:<port>;<databaseName>=name</code>
MySQL	<code>jdbc:mysql://<hostName>:<port>/<databaseName></code>
Oracle	<code>jdbc:oracle:thin:@<hostName>:<port>:<SID></code> <code>jdbc:oracle:thin:@//<hostName>:<port>/<service></code>
Oracle XML DB	<code>jdbc:oracle:oci:@//<hostName>:<port>:<service></code>
PostgreSQL	<code>jdbc:postgresql://<hostName>:<port>/<databaseName></code>
Progress OpenEdge	<code>jdbc:datadirect:openedge://<host>:<port>;<databaseName>=<db_name></code>
Sybase	<code>jdbc:sybase:Tds:<hostName>:<port>/<databaseName></code>
Teradata	<code>jdbc:teradata://<databaseServerName></code>

Note: Syntax variations to the formats listed above are also possible (for example, the database URL may exclude the port or may include the username and password to the database). Check the documentation of the database vendor for further details.

- Click **Connect**.

10.2.6.1 Configuring the CLASSPATH

The `CLASSPATH` environment variable is used by the Java Runtime Environment (JRE) or the Java Development Kit (JDK) to locate Java classes and other resource files on your operating system. When you connect to a database through JDBC, this variable must be configured to include the path to the JDBC driver on your operating system, and, in some cases, the path to additional library files specific to the database type you are using.

The following table lists sample file paths that must be typically included in the `CLASSPATH` variable. Importantly, you may need to adjust this information based on the location of the JDBC driver on your system, the JDBC driver name, as well as the JRE/JDK version present on your operating system. To avoid connectivity problems, check the installation instructions and any pre-installation or post-installation configuration steps applicable to the JDBC driver installed on your operating system.

Database	Sample CLASSPATH entries
Firebird	C:\Program Files\Firebird\Jaybird-2.2.8-JDK_1.8\jaybird-full-2.2.8.jar
IBM DB2	C:\Program Files (x86)\IBM\SQLLIB\java\db2jcc.jar;C:\Program Files (x86)\IBM\SQLLIB\java\db2jcc_license_cu.jar;
IBM DB2 for i	C:\jt400\jt400.jar;
IBM Informix	C:\Informix_JDBC_Driver\lib\ifxjdbc.jar;
Microsoft SQL Server	C:\Program Files\Microsoft JDBC Driver 4.0 for SQL Server\sqljdbc_4.0\enu\sqljdbc.jar
MariaDB	<installation directory>\mariadb-java-client-2.2.0.jar
MySQL	<installation directory>\mysql-connector-java- <i>version</i> -bin.jar;
Oracle	ORACLE_HOME\jdbc\lib\ojdbc6.jar;
Oracle (with XML DB)	ORACLE_HOME\jdbc\lib\ojdbc6.jar;ORACLE_HOME\LIB\xmlparserv2.jar; ORACLE_HOME\RDBMS\jlib\xdb.jar;
PostgreSQL	<installation directory>\postgresql.jar
Progress OpenEdge	%DLC%\java\openedge.jar;%DLC%\java\pool.jar; Note: Assuming the Progress OpenEdge SDK is installed on the machine, %DLC% is the directory where OpenEdge is installed.
Sybase	C:\sybase\jConnect-7_0\classes\jconn4.jar
Teradata	<installation directory>\tdgssconfig.jar;<installation directory>\terajdbc4.jar

- Changing the CLASSPATH variable may affect the behavior of Java applications on your machine. To understand possible implications before you proceed, refer to the Java documentation.
- Environment variables can be user or system. To change system environment variables, you need administrative rights on the operating system.
- After you change the environment variable, restart any running programs for settings to take effect. Alternatively, log off or restart your operating system.

To configure the CLASSPATH on Windows 7:

1. Open the **Start** menu and right-click **Computer**.
2. Click **Properties**.
3. Click **Advanced system settings**.
4. In the **Advanced** tab, click **Environment Variables**.
5. Locate the CLASSPATH variable under user or system environment variables, and then click **Edit**. If the CLASSPATH variable does not exist, click **New** to create it.
6. Edit the variable value to include the path on your operating system where the JDBC driver is located. To separate the JDBC driver path from other paths that may already be in the CLASSPATH variable, use the semi-colon separator (;).

To configure the CLASSPATH on Windows 10:

1. Press the Windows key and start typing "environment variables".
2. Click the suggestion **Edit the system environment variables**.
3. Click **Environment Variables**.
4. Locate the CLASSPATH variable under user or system environment variables, and then click **Edit**. If the CLASSPATH variable does not exist, click **New** to create it.
5. Edit the variable value to include the path on your operating system where the JDBC driver is located. To separate the JDBC driver path from other paths that may already be in the CLASSPATH variable, use the semi-colon separator (;).

10.2.7 SQLite Connection

[SQLite](#) is a file-based, self-contained database type, which makes it ideal in scenarios where portability and ease of configuration is important. Since SQLite databases are natively supported by StyleVision, you do not need to install any drivers to connect to them.

SQLite database support notes

- On Linux, statement execution timeout for SQLite databases is not supported.
- Full text search tables are not supported.
- SQLite allows values of different data types in each row of a given table. All processed values must be compatible with the declared column type; therefore, unexpected values can be retrieved and run-time errors may occur if your SQLite database has row values which are not the same as the declared column type.

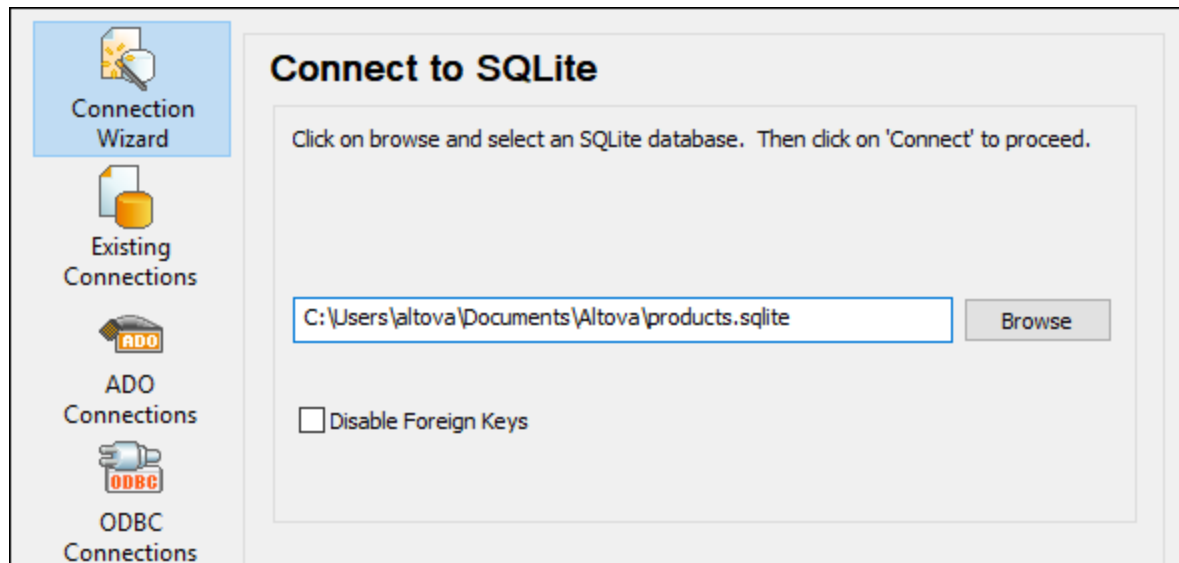
Important

It is recommended to create tables with the `STRICT` keyword to ensure more predictable behavior of your data. Otherwise, the data may not be read or written correctly when values of different types are mixed in one column. To find out more about STRICT tables, see the [SQLite documentation](#).

10.2.7.1 Connect to an Existing SQLite Database

To connect to an existing SQLite database:

1. Run the database connection wizard (see [Starting the Database Connection Wizard](#)⁴⁹³).
2. Select **SQLite**, and then click **Next**.
3. Browse for the SQLite database file, or enter the path (either relative or absolute) to the database. The **Connect** button becomes enabled once you enter the path to a SQLite database file.

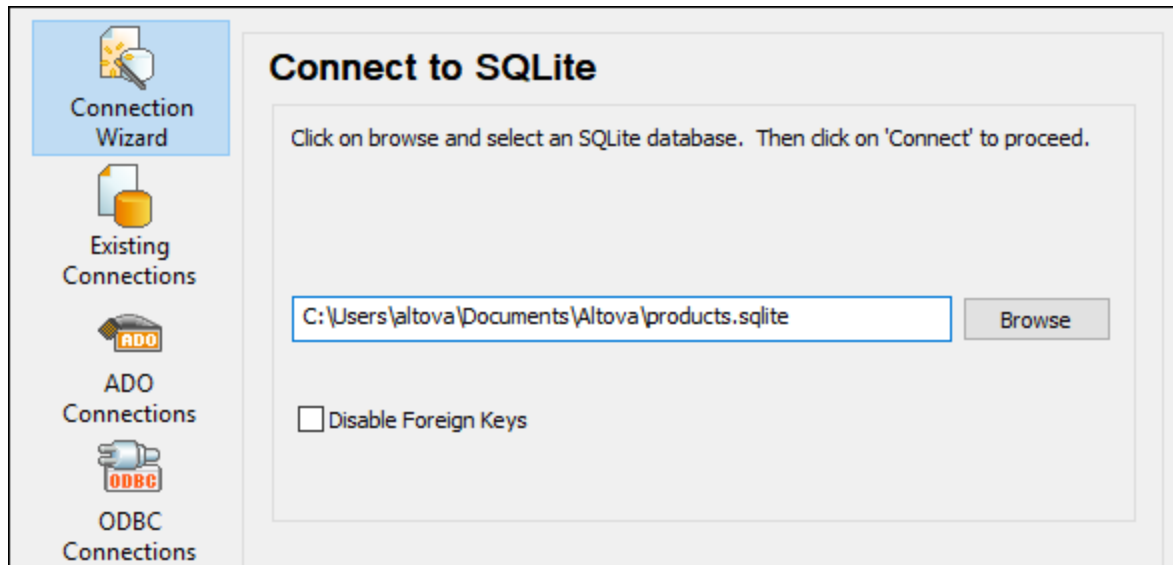


4. Optionally, select the **Disable Foreign Keys** check box, see [Foreign Key Constraints](#)⁵¹⁸.
5. Click **Connect**.

10.2.7.2 Foreign Key Constraints

When you connect to an existing SQLite database from StyleVision, foreign key constraints are enabled by default. Foreign key constraints help preserve the integrity of data in your database. For example, when foreign keys are enabled, it is not possible to delete a record from a table if it has dependencies in another table.

In certain cases, you may want to temporarily override this behavior and disable foreign keys, perhaps, in order to update or insert multiple rows of data without getting data validation errors. To explicitly disable foreign keys before connecting to the SQLite database, select the **Disable Foreign Keys** option available on the database connection wizard.



"Connect to SQLite" wizard page

When foreign keys are disabled, you will be able to perform operations against data that would otherwise not be possible due to validation checks. At the same time, however, there is also the risk of introducing incorrect data into the database, or creating "orphaned" rows. (An example of an "orphaned" row would be an address in the "addresses" table not linked to any person in the "person" table, because the person was deleted but its associated address was not.)

10.2.8 Native Connection

Native connections are direct connections to the DB that do not need drivers to be installed. Also, if you intend to deploy files for execution on a Linux or macOS server, you do not need to install drivers on the target server either.

You can set up native connections for the following DBs:

- MariaDB
- MySQL
- SQLite
- PostgreSQL

If you prefer to establish a connection by means of a driver, see the following topics:

- [Setting up a JDBC Connection](#) ⁵¹³
- [SQLite Connection](#) ⁵¹⁷
- [Connecting to PostgreSQL \(ODBC\)](#) ⁵⁵⁹

Connection setup

To set up a native connection, follow the steps below. You will need the following information: host name, port, database name, username, and password.

1. [Start the database connection wizard](#)⁴⁹³.
2. Select the DB you want to connect to (MariaDB, MySQL, PostgreSQL, or SQLite).
3. In the dialog that appears, enter the host (for example, *localhost*), optionally the port (typically 5432), SSL Mode in the case of MySQL, the database name, username, and password in the corresponding boxes.
4. Click **Connect**.

SQLite conections

For detailed information about SQLite connections, see the topic [SQLite Connection](#)⁵¹⁷.

Notes for PostgreSQL

If the PostgreSQL database server is on a different machine, note the following:

- The PostgreSQL database server must be configured to accept connections from clients. Specifically, the **pg_hba.conf** file must be configured to allow non-local connections. Secondly, the **postgresql.conf** file must be configured to listen on specified IP address(es) and port. For more information, check the PostgreSQL documentation (<https://www.postgresql.org/docs/9.5/static/client-authentication-problems.html>).
- The server machine must be configured to accept connections on the designated port (typically, 5432) through the firewall. For example, on a database server running on Windows, a rule may need to be created to allow connections on port 5432 through the firewall, from **Control Panel > Windows Firewall > Advanced Settings > Inbound Rules**.

10.2.9 Global Resources

After you have created a database as a global resource, its connection details are stored and can be used across all Altova products installed on your machine.

Create a database as a global resource

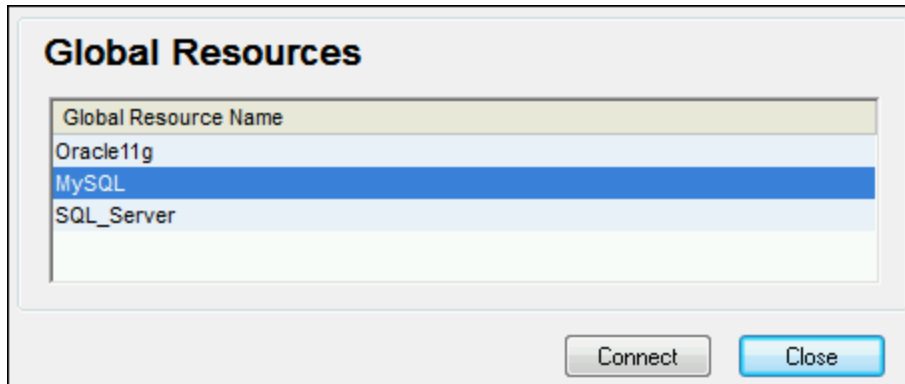
To create a database as a global resource, do the following

1. On the **Tools** menu of StyleVision, click **Global Resources**.
2. Click **Add**, and then click Database.
3. Type in a name for the global resource in the *Resource Alias* field.
4. Click **Choose Database**. The [Connection Wizard](#)⁴⁹³ appears.
5. Use the Connection Wizard to add a database connection as described above.

Use a global-resource database

To use a database that has been created as a global resource (*see above*), do the following:

1. Start the Connection Wizard as described above.
2. Select Global Resources. All the databases that have been created as global resources will be listed by their names in the Global Resources pane (see *screenshot below*).



3. Select the global resource that you want. Tip: Move the mouse cursor over a global resource in the list to see information about the database.

10.2.10 Database Connection Examples

This section includes examples for connecting to a database from StyleVision through ADO, ODBC, or JDBC. The ADO.NET connection examples are listed separately, see [Sample ADO.NET Connection Strings](#)⁵⁰⁸. For instructions about establishing a native connection to PostgreSQL and SQLite, see [Setting up a PostgreSQL Connection](#)⁵¹⁹ and [Setting up a SQLite Connection](#)⁵¹⁷, respectively.

Note the following:

- The instructions may differ if your Windows configuration, network environment and the database client or server software are not the same as the ones described in each example.
- For most database types, it is possible to connect using more than one data access technology (ADO, ADO.NET, ODBC, JDBC) or driver. The performance of the database connection, as well as its features and limitations will depend on the selected driver, database client software (if applicable), and any additional connectivity parameters that you may have configured outside StyleVision.

10.2.10.1 Firebird (JDBC)

This example illustrates how to connect to a Firebird database server through JDBC.

Prerequisites:

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. StyleVision will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.
- Make sure that the platform of StyleVision (32-bit, 64-bit) matches that of the JRE/JDK.

- The Firebird JDBC driver must be available on your operating system (it takes the form of a .jar file which provides connectivity to the database). The driver can be downloaded from the Firebird website (<https://www.firebirdsql.org/>). This example uses *Jaybird 2.2.8*.
- You have the following database connection details: host, database path or alias, username, and password.

To connect to Firebird through JDBC:

1. [Start the database connection wizard](#) ⁴⁹³.
2. Click **JDBC Connections**.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the required .jar file is located at the following path: **C:\jdbc\firebird\jaybird-full-2.2.8.jar**. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also [Configuring the CLASSPATH](#) ⁵¹⁶).
4. In the "Driver" box, select **org.firebirdsql.jdbc.FBDriver**. Note that this entry is available if a valid .jar file path is found either in the "Classpath" text box, or in the operating system's CLASSPATH environment variable (see the previous step).

Classpaths: C:\jdbc\firebird\jaybird-full-2.2.8.jar

Driver: org.firebirdsql.jdbc.FBDriver

Username: prod_admin

Password: ●●●●●●

Database URL: jdbc:firebirdsql://firebirdserv/COMPANY

Connect Close

5. Enter the username and password to the database in the corresponding text boxes.
6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

```
jdbc:firebirdsql://<host>[:<port>]/<database path or alias>
```

7. Click **Connect**.

10.2.10.2 Firebird (ODBC)

This example illustrates how to connect to a Firebird 2.5.4 database running on a Linux server.


Prerequisites:

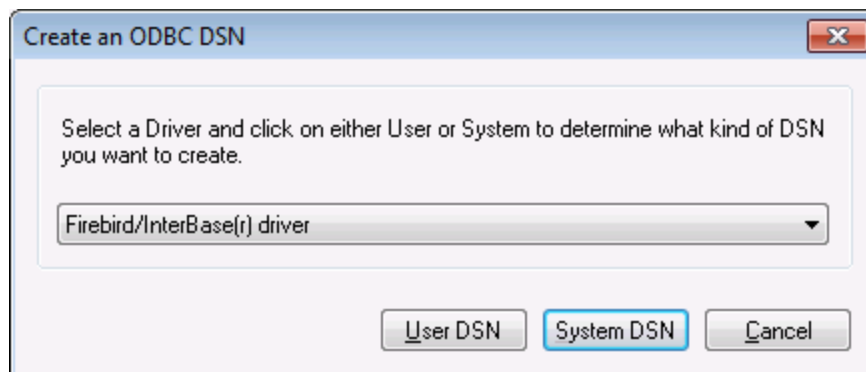
- The Firebird database server is configured to accept TCP/IP connections from clients.
- The Firebird ODBC driver must be installed on your operating system. This example uses the Firebird ODBC driver version 2.0.3.154 downloaded from the Firebird website (<https://www.firebirdsql.org/>).
- The Firebird client must be installed on your operating system. Note that there is no standalone installer available for the Firebird 2.5.4 client; the client is part of the Firebird server installation package. You can download the Firebird server installation package from the Firebird website (<https://www.firebirdsql.org/>), look for "Windows executable installer for full Superclassic/Classic or Superserver". To install only the client files, choose "**Minimum client install - no server, no tools**" when going through the wizard steps.

Important:

- The platform of both the Firebird ODBC driver and client (32-bit or 64-bit) must correspond to that of StyleVision.
 - The version of the Firebird client must correspond to the version of Firebird server to which you are connecting.
- You have the following database connection details: server host name or IP address, database path (or alias) on the server, user name, and password.

To connect to Firebird via ODBC:

1. [Start the database connection wizard](#) ⁴⁹³.
2. Click **ODBC Connections**.
3. Select **User DSN** (or **System DSN**, if you have administrative privileges), and then click **Add** .



4. Select the Firebird driver, and then click **User DSN** (or **System DSN**, depending on what you selected in the previous step). If the Firebird driver is not available in the list, make sure that it is installed on your operating system (see also [Viewing the Available ODBC Drivers](#)⁵¹²).

5. Enter the database connection details as follows:

<i>Data Source Name (DSN)</i>	Enter a descriptive name for the data source you are creating.
<i>Database</i>	<p>Enter the server host name or IP address, followed by a colon, followed by the database alias (or path). In this example, the host name is <code>firebirdserv</code>, and the database alias is <code>products</code>, as follows:</p> <pre>firebirdserv:products</pre> <p>Using a database alias assumes that, on the server side, the database administrator has configured the alias <i>products</i> to point to the actual Firebird (.fdb) database file on the server (see the Firebird documentation for more details).</p> <p>You can also use the server IP address instead of the host name, and a path instead of an alias; therefore, any of the following sample connection strings are valid:</p>

	<pre>firebirdserver:/var/Firebird/databases/butterflies.fdb 127.0.0.1:D:\Misc\Lenders.fdb</pre> <p>If the database is on the local Windows machine, click Browse and select the Firebird (.fdb) database file directly.</p>
<i>Client</i>	Enter the path to the fbclient.dll file. By default, this is the <code>bin</code> subdirectory of the Firebird installation directory.
<i>Database Account</i>	Enter the database user name supplied by the database administrator (in this example, <code>PROD_ADMIN</code>).
<i>Password</i>	Enter the database password supplied by the database administrator.

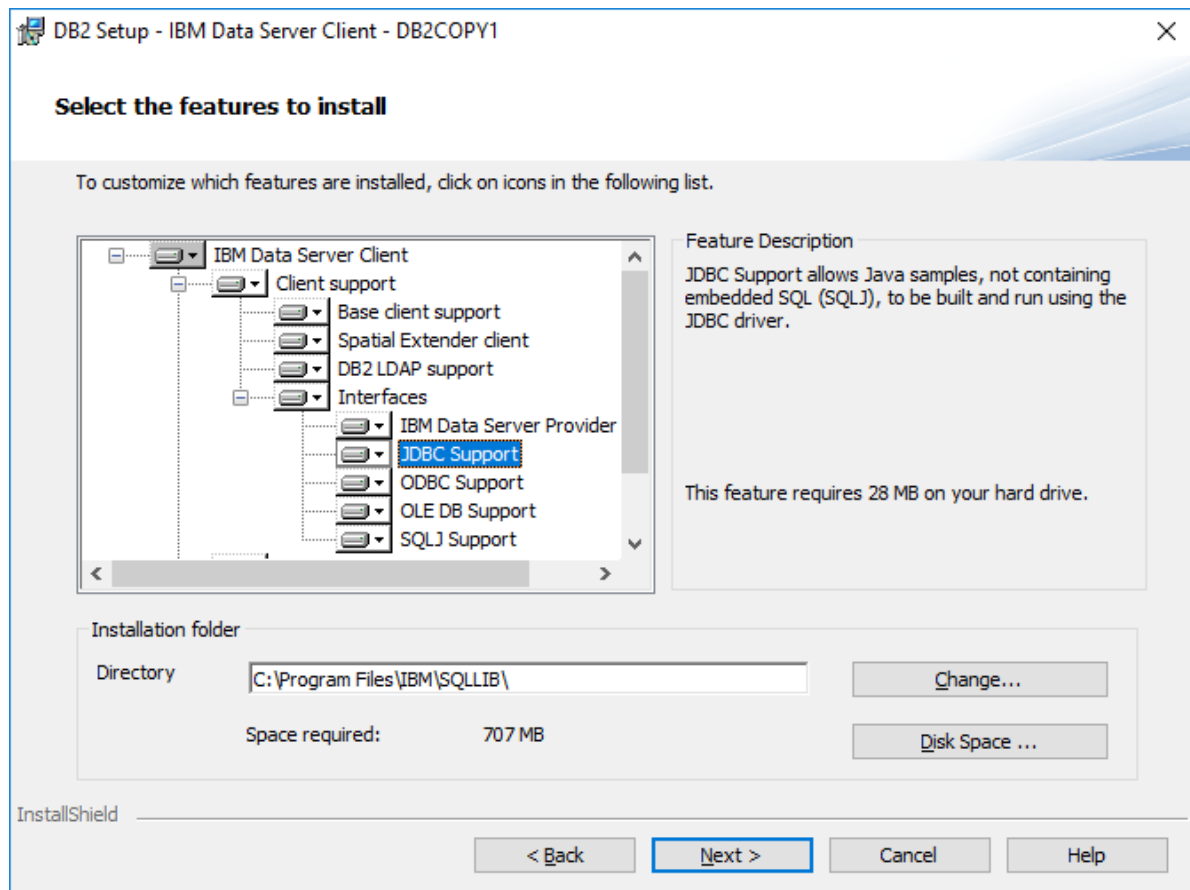
6. Click **OK**.

10.2.10.3 IBM DB2 (JDBC)

This example illustrates how to connect to an **IBM DB2** database server through JDBC.

Prerequisites:

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. StyleVision will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.
- Make sure that the platform of StyleVision (32-bit, 64-bit) matches that of the JRE/JDK. This example uses Oracle's OpenJDK 11.0 64-bit, and, consequently, the 64-bit version of StyleVision.
- The JDBC driver (one or several .jar files that provide connectivity to the database) must be available on your operating system. This example uses the JDBC driver available after installing the **IBM Data Server Client** version 10.1 (64-bit). For the JDBC drivers to be installed, choose a **Typical** installation, or select this option explicitly on the installation wizard.



If you did not change the default installation path, the required .jar files will be in the **C:\Program Files\IBM\SQLLIB\java** directory after installation.

- You need the following database connection details: host, port, database name, username, and password.

To connect to IBM DB2 through JDBC:

1. [Start the database connection wizard](#) ⁴⁹³.
2. Click **JDBC Connections**.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. This examples refers to **C:\Program Files\IBM\SQLLIB\java\db2jcc.jar**. You may need to refer to the **db2jcc4.jar** driver, depending on the database server version. For driver compatibility, refer to IBM documentation (<http://www-01.ibm.com/support/docview.wss?uid=swg21363866>). Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also [Configuring the CLASSPATH](#) ⁵¹⁶).
4. In the "Driver" box, select **com.ibm.db2.jcc.DB2Driver**. This entry becomes available only if a valid .jar file path was found either in the "Classpaths" text box, or in the operating system's CLASSPATH environment variable (see the previous step).

Classpaths:	<input type="text" value="C:\Program Files\IBM\SQLLIB\java\db2jcc.jar"/>
Driver:	<input type="text" value="com.ibm.db2.jcc.DB2Driver"/>
Username:	<input type="text" value="username"/>
Password:	<input type="password" value="●●●●●●●●"/>
Database URL:	<input type="text" value="jdbc:db2://dbserver:50000/dbname"/>

5. Enter the username and password of the database user in the corresponding text boxes.
6. Enter the JDBC connection string in the **Database URL** text box. Make sure to replace the connection details with the ones applicable to your database server.

```
jdbc:db2://hostName:port/databaseName
```

7. Click **Connect**.

10.2.10.4 IBM DB2 (ODBC)

This example illustrates how to connect to an IBM DB2 database through ODBC.

Prerequisites:

- IBM Data Server Client must be installed and configured on your operating system (this example uses IBM Data Server Client 9.7). For installation instructions, check the documentation supplied with your IBM DB2 software. After installing the IBM Data Server Client, check if the ODBC drivers are available on your machine (see [Viewing the Available ODBC Drivers](#)⁵¹²).
- Create a database alias. There are several ways to do this:
 - From IBM DB2 Configuration Assistant
 - From IBM DB2 Command Line Processor
 - From the ODBC data source wizard (for this case, the instructions are shown below)
- You have the following database connection details: host, database, port, username, and password.

To connect to IBM DB2:

1. [Start the database connection wizard](#)⁴⁹³ and select **IBM DB2 (ODBC/JDBC)**.
2. Click **Next**.

JDBC vs. ODBC

JDBC

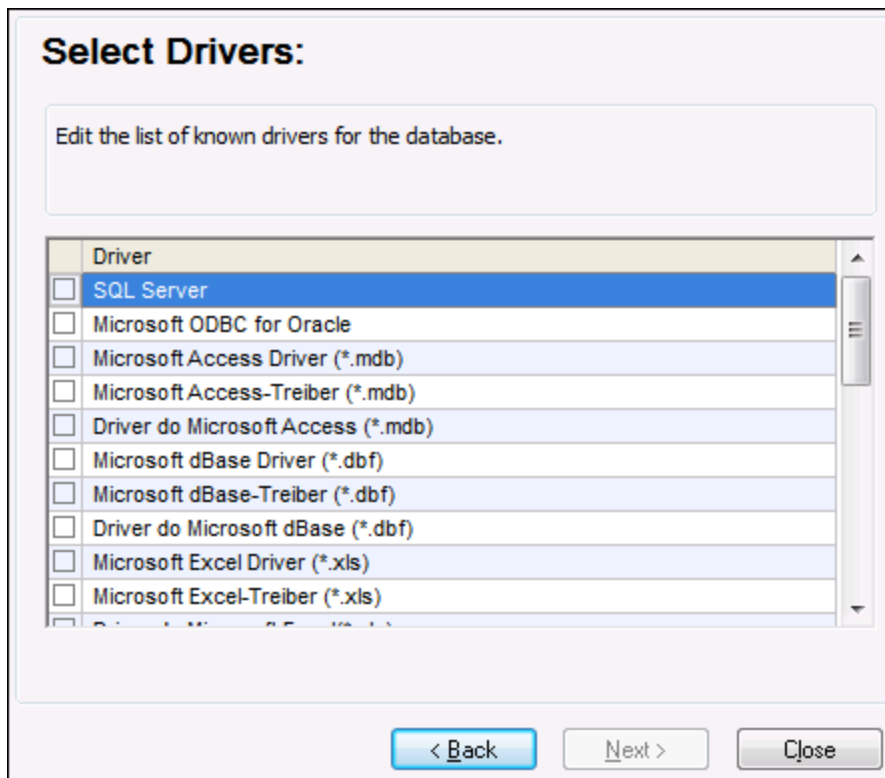
Java-based connection that may provide support for more modern features of your database that aren't available via ODBC. The tradeoff for these features is a potential sacrifice of performance.

ODBC

An ODBC connection will generally be faster and less resource-intensive than a JDBC connection, but lacks support for more modern database features (such as native XML types).

< Back Next > Close

3. Select **ODBC**, and click **Next**. If prompted to edit the list of known drivers for the database, select the database drivers applicable to IBM DB2 (see [Prerequisites](#)⁶²⁷), and click **Next**.



4. Select the IBM DB2 driver from the list, and then click **Connect**. (To edit the list of available drivers, click **Edit Drivers**, and then check or uncheck the IBM DB2 drivers you wish to add or remove, respectively.)

Connecting to IBM DB2

Where can I find IBM DB2 drivers?

Select an option how you wish to connect to the database and click Connect.

Create a new Data Source Name (DSN) with the driver:

IBM DB2 ODBC DRIVER

Use an existing Data Source Name:

User DSN System DSN Edit Drivers

Skip the configuration step for wizard

< Back Connect Close

5. Enter a data source name (in this example, **DB2DSN**), and then click **Add**.

Select the DB2 database alias you want to register for ODBC, or select Add to create a new alias. You may change the data source name and description, or accept the default.

Data source name: DB2DSN

Database alias: Add

Description:

OK Cancel

6. On the **Data Source** tab, enter the user name and password to the database.

The image shows a dialog box titled 'Data Source' with four tabs: 'Data Source', 'TCP/IP', 'Security options', and 'Advanced Settings'. The 'Security options' tab is selected. The dialog contains the following fields and controls:

- 'Data source name' text box containing 'DB2DSN'
- 'Description' text box (empty)
- 'User ID' text box containing 'john_doe'
- 'Password' text box containing ten black dots
- 'Save password' checkbox (unchecked)
- 'OK', 'Cancel', 'Apply', and 'Help' buttons at the bottom.

7. On the **TCP/IP** tab, enter the database name, a name for the alias, the host name and the port number, and then click OK.

Data Source TCP/IP Security options Advanced Settings

Database name

Database alias

Host name

Port number

The database physically resides on a host or QS/400 system.

Connect directly to the server

Connect to the server via the gateway

DCS Parameters

Optimize for application

OK Cancel Apply Help

8. Enter again the username and password, and then click **OK**.

Database alias

User ID

Password

Change password

New password

Verify new password

Connection mode

Share Exclusive

OK Cancel

10.2.10.5 IBM DB2 for i (JDBC)

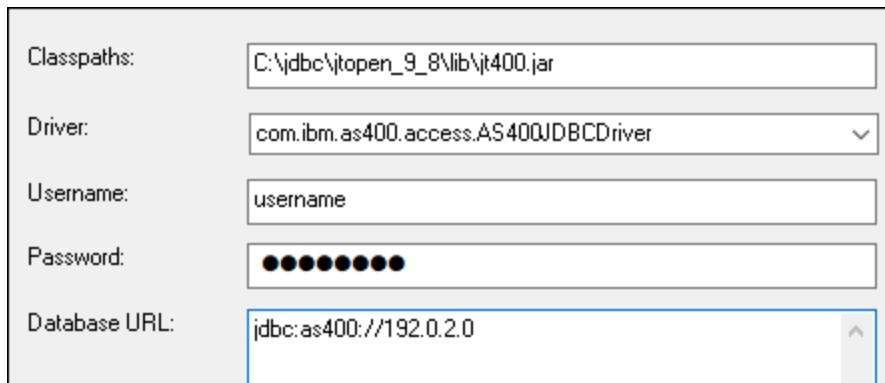
This example illustrates how to connect to an **IBM DB2 for i** database server through JDBC.

Prerequisites:

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. StyleVision will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.
- Make sure that the platform of StyleVision (32-bit, 64-bit) matches that of the JRE/JDK. This example uses Oracle's OpenJDK 11.0 64-bit, and, consequently, the 64-bit version of StyleVision.
- The JDBC driver (one or several .jar files that provide connectivity to the database) must be available on your operating system. This example uses the open source **Toolbox for Java/JTOpen** version 9.8 (<http://jt400.sourceforge.net/>). After you download the package and unpack to a local directory, the required .jar files will be available in the **lib** subdirectory.
- You need the following database connection details: host, username, and password.

To connect to IBM DB2 for i through JDBC:

1. [Start the database connection wizard](#) ⁴⁹³.
2. Click **JDBC Connections**.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. In this example, the required .jar file is at the following path: **C:\jdbc\jtopen_9_8\lib\jt400.jar**. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also [Configuring the CLASSPATH](#) ⁵¹⁶).
4. In the "Driver" box, select **com.ibm.as400.access.AS400JDBCdriver**. This entry becomes available only if a valid .jar file path was found either in the "Classpaths" text box, or in the operating system's CLASSPATH environment variable (see the previous step).



Classpaths:	C:\jdbc\jtopen_9_8\lib\jt400.jar
Driver:	com.ibm.as400.access.AS400JDBCdriver
Username:	username
Password:	●●●●●●●●●●
Database URL:	jdbc:as400://192.0.2.0

5. Enter the username and password of the database user in the corresponding text boxes.
6. Enter the JDBC connection string in the **Database URL** text box. Make sure to replace **host** with the host name or IP address of your database server.

```
jdbc:as400://host
```

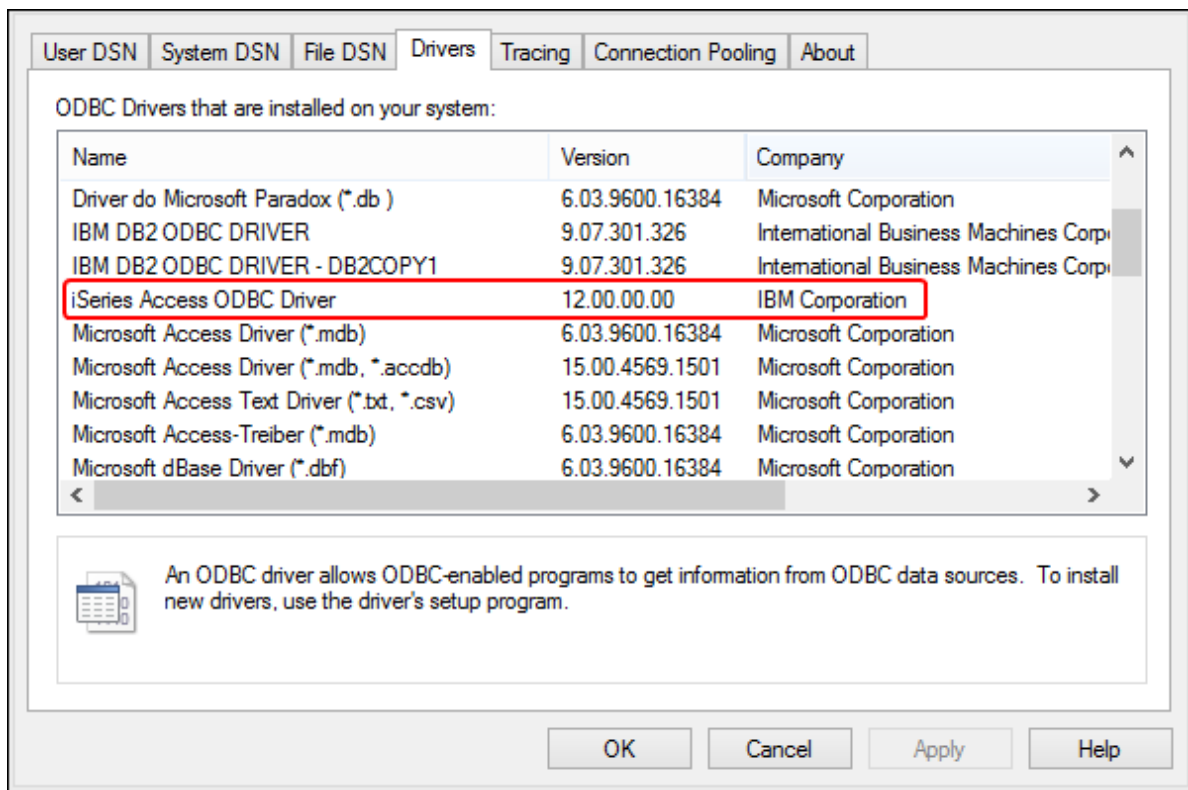
- Click **Connect**.

10.2.10.6 IBM DB2 for i (ODBC)

This example illustrates how to connect to an *IBM DB2 for i* database through ODBC.

Prerequisites:


- IBM System i Access for Windows* must be installed on your operating system (this example uses *IBM System i Access for Windows V6R1M0*). For installation instructions, check the documentation supplied with your *IBM DB2 for i* software. After installation, check if the ODBC driver is available on your machine (see [Viewing the Available ODBC Drivers](#)⁵¹²).

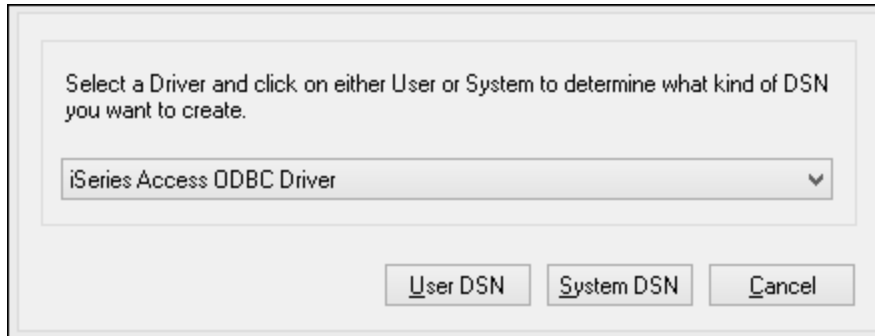


- You have the following database connection details: the I.P. address of the database server, database user name, and password.
- Run *System i Navigator* and follow the wizard to create a new connection. When prompted to specify a system, enter the I.P. address of the database server. After creating the connection, it is recommended to verify it (click on the connection, and select **File > Diagnostics > Verify Connection**). If you get connectivity errors, contact the database server administrator.

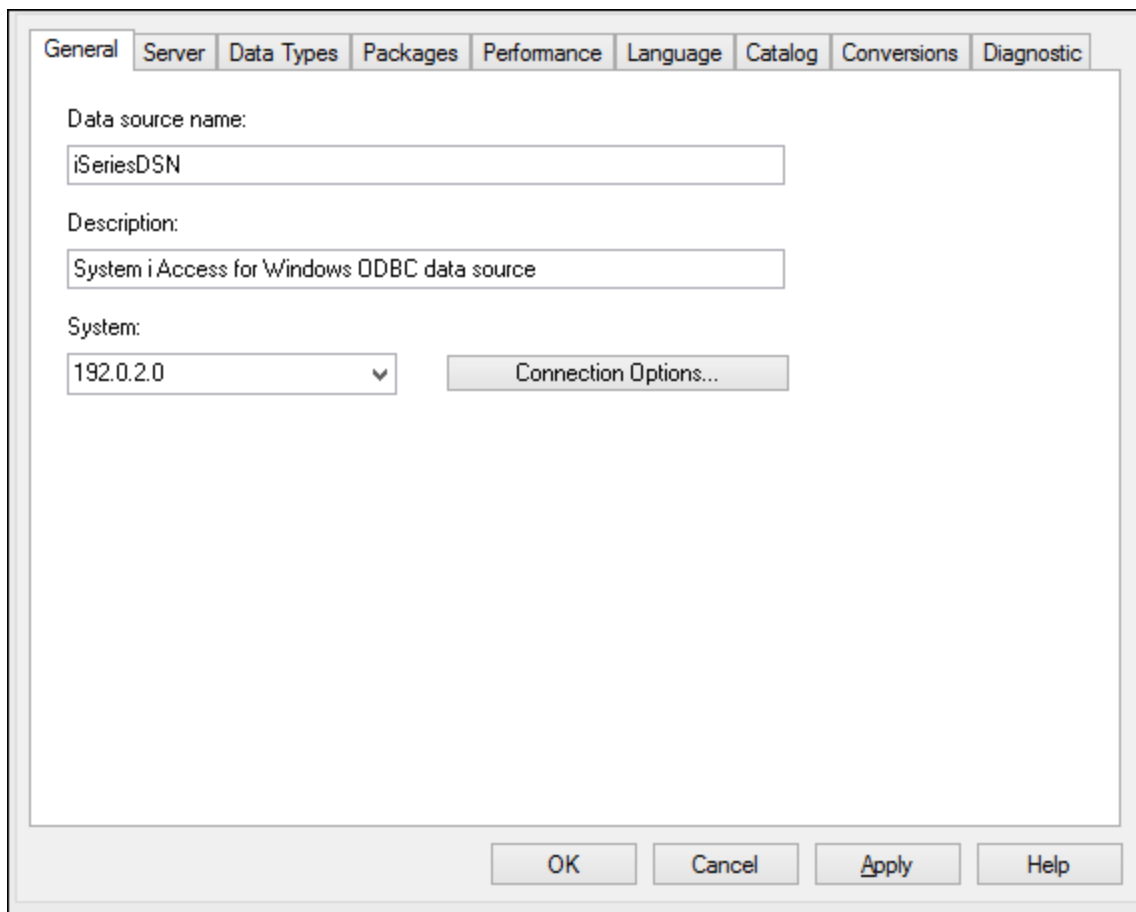
To connect to IBM DB2 for i:

- [Start the database connection wizard](#)⁴⁹³.
- Click **ODBC connections**.

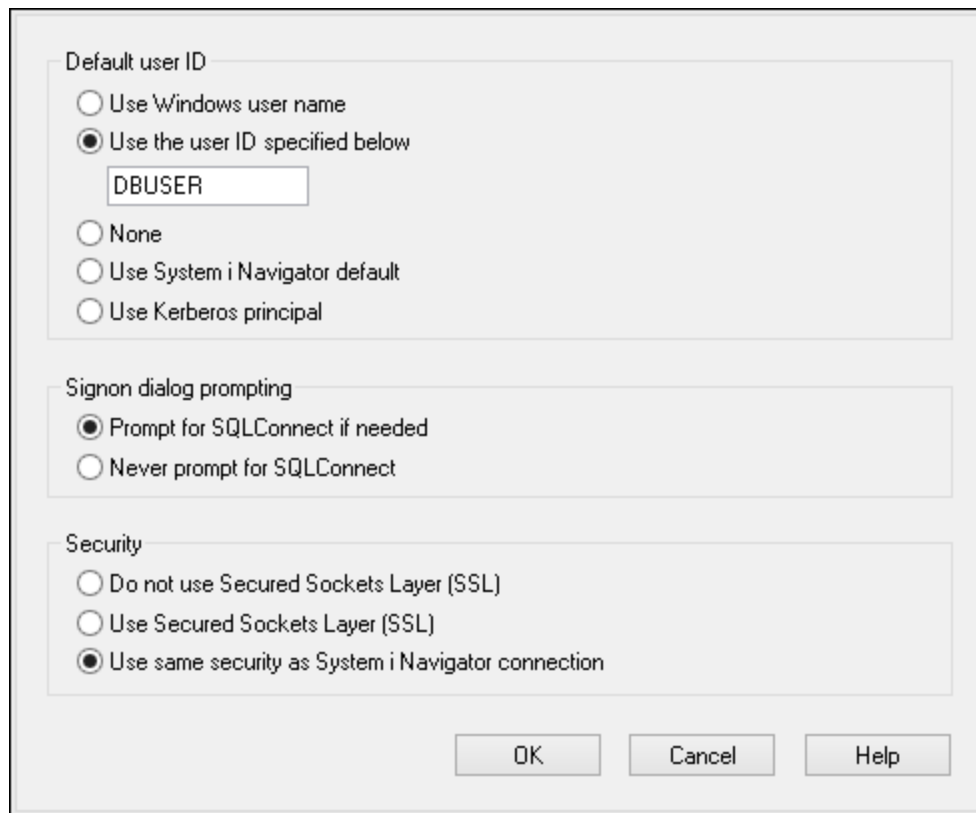
3. Click **User DSN** (alternatively, click **System DSN**, or **File DSN**, in which case the subsequent instructions will be similar).
4. Click **Add** .
5. Select the **iSeries Access ODBC Driver** from the list, and click **User DSN** (or **System DSN**, if applicable).



6. Enter a data source name and select the connection from the System combo box. In this example, the data source name is **iSeriesDSN** and the System is **192.0.2.0**.



7. Click **Connection Options**, select **Use the User ID specified below** and enter the name of the database user (in this example, **DBUSER**).



8. Click **OK**. The new data source becomes available in the list of DSNs.
9. Click **Connect**.
10. Enter the user name and password to the database when prompted, and then click **OK**.

10.2.10.7 IBM Informix (JDBC)

This example illustrates how to connect to an IBM Informix database server through JDBC.

Prerequisites:

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. StyleVision will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.
- Make sure that the platform of StyleVision (32-bit, 64-bit) matches that of the JRE/JDK.
- The JDBC driver (one or several .jar files that provide connectivity to the database) must be available on your operating system. In this example, IBM Informix JDBC driver version 3.70 is used. For the driver's installation instructions, see the documentation accompanying the driver or the "IBM Informix JDBC Driver Programmer's Guide".
- You have the following database connection details: host, name of the Informix server, database, port, username, and password.

To connect to IBM Informix through JDBC:

1. [Start the database connection wizard](#) ⁴⁹³.
2. Click **JDBC Connections**.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the required .jar file is located at the following path: **C:\Informix_JDBC_Driver\lib\ifxjdbc.jar**. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also [Configuring the CLASSPATH](#) ⁵¹⁶).
4. In the "Driver" box, select **com.informix.jdbc.IfxDriver**. Note that this entry is available if a valid .jar file path is found either in the "Classpaths" text box, or in the operating system's CLASSPATH environment variable (see the previous step).

Classpaths: C:\jdbc\Informix_JDBC_Driver\lib\ifxjdbc.jar;

Driver: com.informix.jdbc.IfxDriver

Username: dbuser

Password: ●●●●●●

Database URL: jdbc:informix-sqli://host:port/MyDatabase:INFORMIXSERVER=MyServerName

Connect Close

5. Enter the username and password to the database in the corresponding text boxes.
6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

```
jdbc:informix-sqli://hostName:port/databaseName:INFORMIXSERVER=myserver;
```

7. Click **Connect**.

10.2.10.8 MariaDB (ODBC)

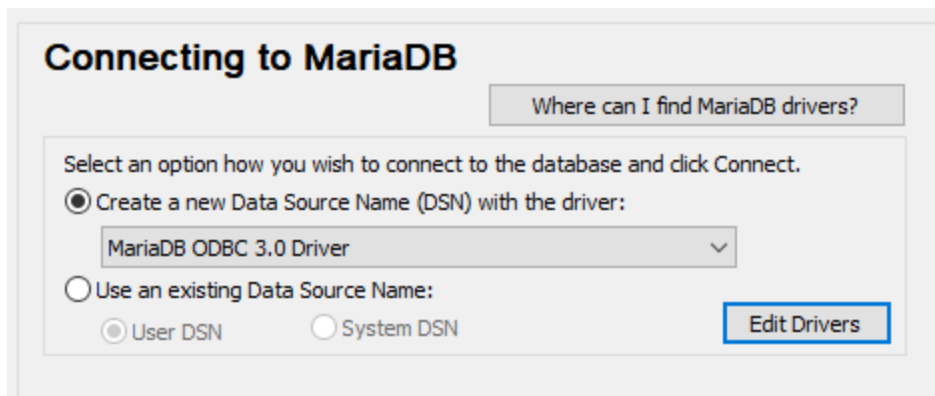
This example illustrates how to connect to a MariaDB database server through ODBC.

Prerequisites:

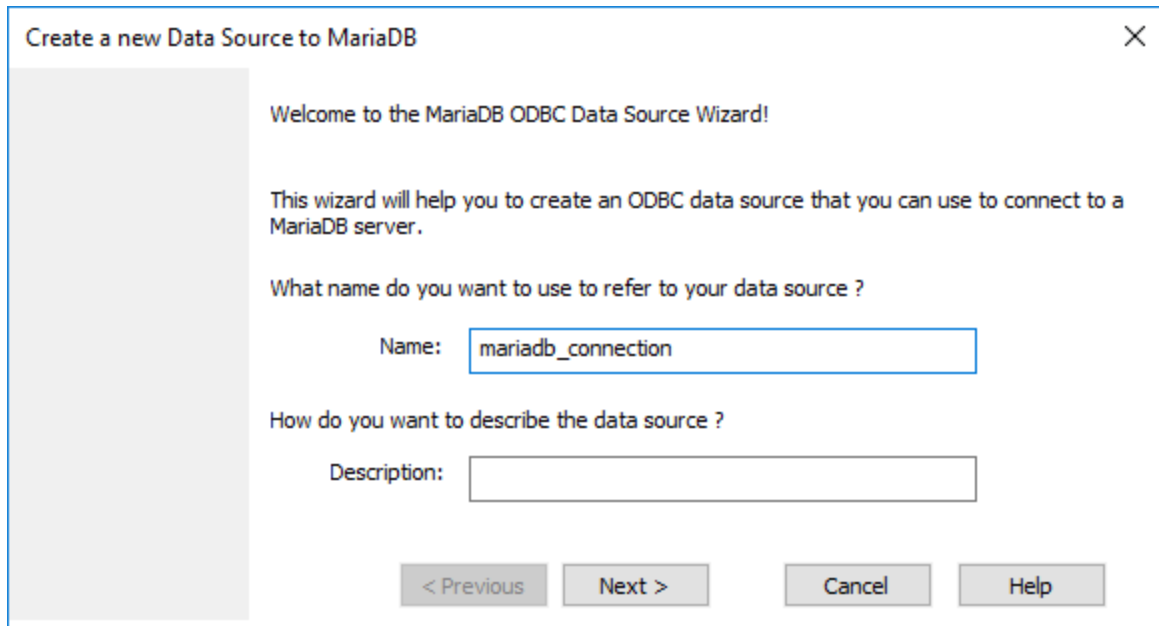
- The MariaDB Connector/ODBC (<https://downloads.mariadb.org/connector-odbc/>) must be installed.
- You have the following database connection details: host, database, port, username, and password.

To connect to MariaDB through ODBC:

1. [Start the database connection wizard](#)⁴⁹³.
2. Select **MariaDB (ODBC)**, and then click **Next**.



3. Select **Create a new Data Source Name (DSN) with the driver**, and choose **MariaDB ODBC 3.0 Driver**. If no such driver is available in the list, click **Edit Drivers**, and select any available MariaDB drivers (the list contains all ODBC drivers installed on your operating system).
4. Click **Connect**.



Create a new Data Source to MariaDB

Welcome to the MariaDB ODBC Data Source Wizard!

This wizard will help you to create an ODBC data source that you can use to connect to a MariaDB server.

What name do you want to use to refer to your data source ?

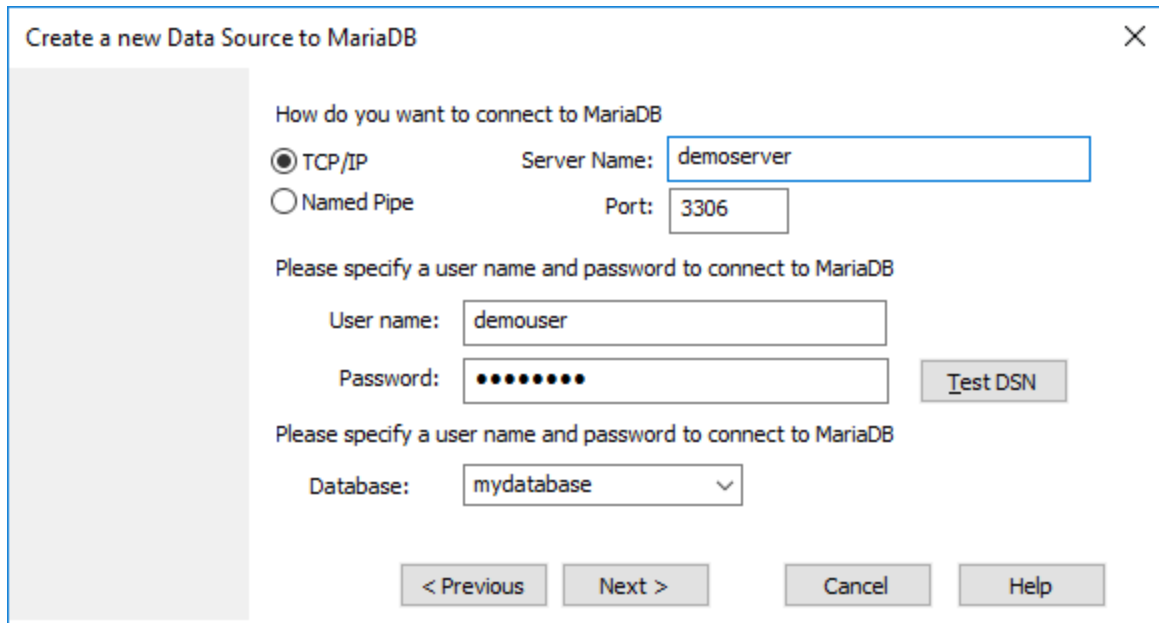
Name:

How do you want to describe the data source ?

Description:

< Previous Next > Cancel Help

5. Enter name and, optionally, a description that will help you identify this ODBC data source in future.



Create a new Data Source to MariaDB

How do you want to connect to MariaDB

TCP/IP Server Name:
 Named Pipe Port:

Please specify a user name and password to connect to MariaDB

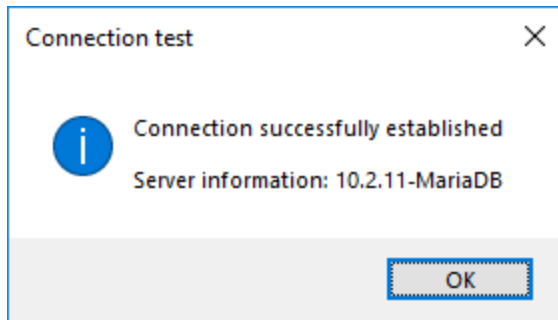
User name:
Password: Test DSN

Please specify a user name and password to connect to MariaDB

Database:

< Previous Next > Cancel Help

6. Fill in the database connection credentials (TCP/IP Server, User, Password), select a database, and then click **Test DSN**. Upon successful connection, a message box appears:



7. Click **Next** and complete the wizard. Other parameters may be required, depending on the case (for example, SSL certificates if you are connecting to MariaDB through a secure connection).

Note: If the database server is remote, it must be configured by the server administrator to accept remote connections from your machine's IP address.

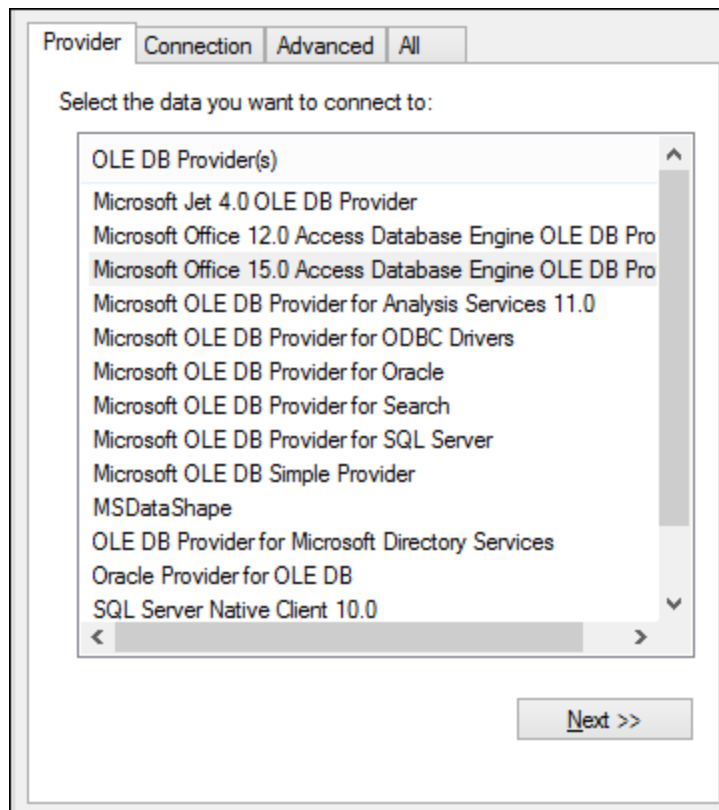
10.2.10.9 Microsoft Access (ADO)

A simple way to connect to a Microsoft Access database is to follow the wizard and browse for the database file, as shown in [Connecting to an Existing Microsoft Access Database](#)⁵⁰¹. An alternative approach is to set up an ADO connection explicitly, as shown in this topic. This approach is useful if your database is password-protected.

It is also possible to connect to Microsoft Access through an ODBC connection, but it has limitations, so it is best to avoid it.

To connect to a password-protected Microsoft Access database:

1. [Start the database connection wizard](#)⁴⁹³.
2. Click **ADO Connections**.
3. Click **Build**.



4. Select the **Microsoft Office 15.0 Access Database Engine OLE DB Provider**, and then click **Next**.

5. In the Data Source box, enter the path to the Microsoft Access file in UNC format, for example, \\myserver\mynetworkshare\Reports\Revenue.accdb, where **myserver** is the name of the server and **mynetworkshare** is the name of the network share.
6. On the **All** tab, double click the **Jet OLEDB:Database Password** property and enter the database password as property value.

Note: If you are still unable to connect, locate the workgroup information file (**System.MDW**) applicable to your user profile, and set the value of the **Jet OLEDB: System database** property to the path of the **System.MDW** file.

10.2.10.10 Microsoft Azure SQL (ODBC)

In order to connect properly to an Azure SQL database, you must install the latest [SQL Server Native Client](#).

For information about connecting to an Azure SQL database in the cloud, see this [Altova blog entry](#).

10.2.10.11 Microsoft SQL Server (ADO)

This example illustrates how to connect to a SQL Server database through ADO. These instructions are applicable when you use the recommended **Microsoft OLE DB Driver for SQL Server (MSOLEDBSQL)**, which is available for download at <https://docs.microsoft.com/en-us/sql/connect/oledb/download-oledb-driver-for-sql-server?view=sql-server-ver15>.

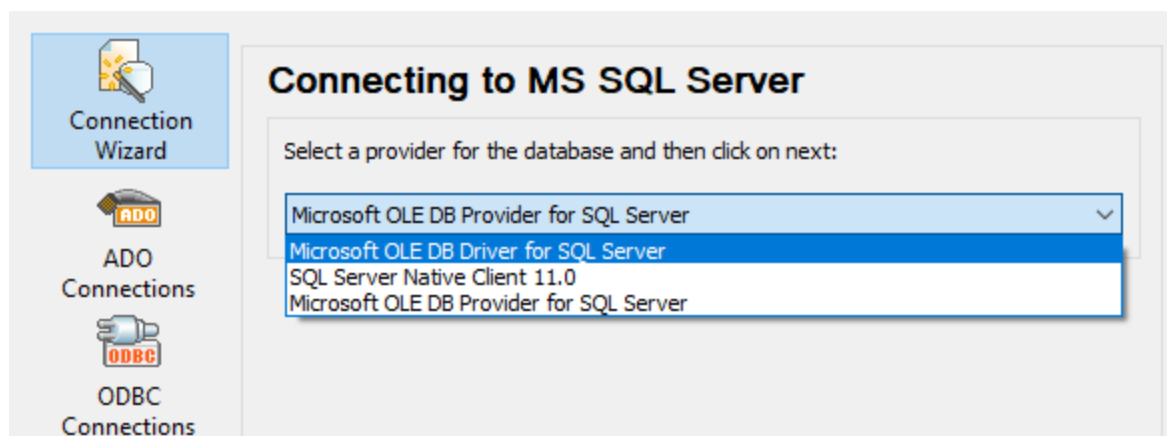
Before following these instructions, make sure that you have downloaded and installed the provider above on your workstation. The ADO provider must match the platform of StyleVision (32-bit or 64-bit).

If you would like to use other ADO providers such as **SQL Server Native Client (SQLNCLI)** or **Microsoft OLE DB Provider for SQL Server (SQLOLEDB)**, the instructions are similar, but these providers are deprecated and thus not recommended. Also, for the connection to be successful with a deprecated provider, you may need to set additional connection properties as described in [Setting up the SQL Server Data Link Properties](#) ⁵⁰¹.

The **Microsoft OLE DB Provider for SQL Server (SQLOLEDB)** is known to have issues with parameter binding of complex queries like Common Table Expressions (CTE) and nested SELECT statements.

To connect to SQL Server:

1. [Start the database connection wizard](#) ⁴⁹³.
2. Select **Microsoft SQL Server (ADO)**, and then click **Next**. The list of available ADO providers is displayed. In this example, the **Microsoft OLE DB Driver for SQL Server** is used. If it's not in the list, make sure that it is installed on your computer, as mentioned above.



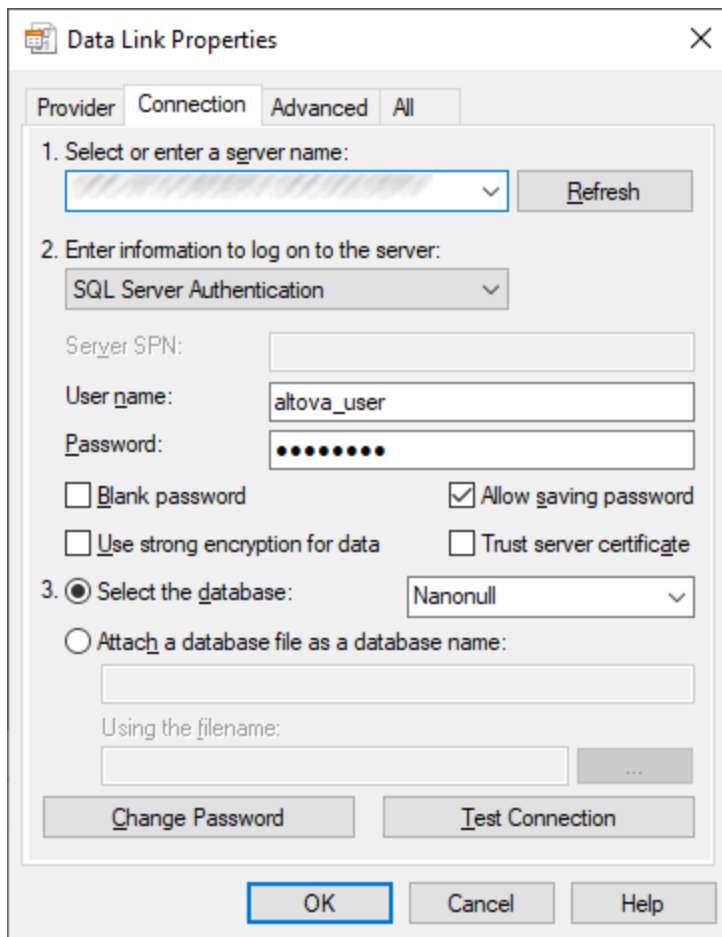
3. Click **Next**. The Data Link Properties dialog box appears.

The screenshot shows the 'Data Link Properties' dialog box with the 'Connection' tab selected. The dialog is divided into three sections:

- 1. Select or enter a server name:** A dropdown menu and a 'Refresh' button.
- 2. Enter information to log on to the server:** A dropdown menu for authentication (currently 'SQL Server Authentication'), and text boxes for 'Server SPN', 'User name', and 'Password'. There are also checkboxes for 'Blank password', 'Allow saving password', 'Use strong encryption for data', and 'Trust server certificate'.
- 3. Select the database:** A radio button selected for 'Select the database:' with a dropdown menu. An unselected radio button is for 'Attach a database file as a database name:', which includes a text box and a 'Using the filename:' section with another text box and a browse button ('...').

At the bottom of the dialog are buttons for 'Change Password', 'Test Connection', 'OK', 'Cancel', and 'Help'.

4. Select or enter the name of the database server, for example, **SQLSERV01**. If you are connecting to a named SQL Server instance, the server name looks like **SQLSERV01\SOMEINSTANCE**.
5. If the database server was configured to allow connections from users authenticated on the Windows domain, select **Windows Authentication**. Otherwise, select **SQL Server Authentication**, clear the **Blank password** check box, and enter the database credentials in the relevant boxes.
6. Select the **Allow saving password** check box and the database to which you are connecting (in this example, "Nanonull").



7. To test the connection at this time, click **Test Connection**. This is an optional, recommended step.
8. Click **OK**.


10.2.10.12 Microsoft SQL Server (ODBC)

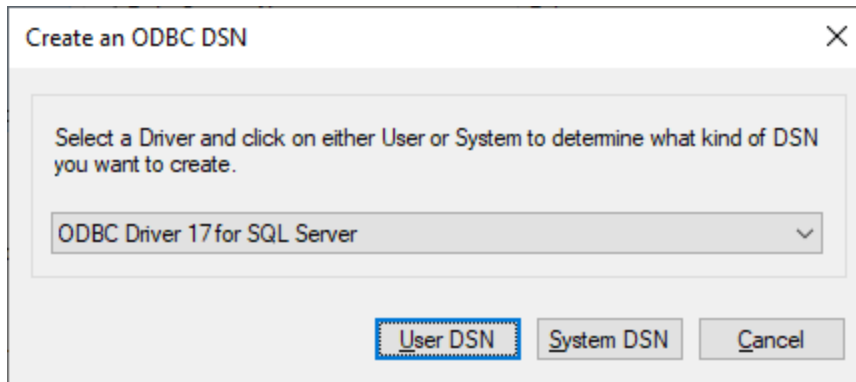
This example illustrates how to connect to a SQL Server database through ODBC.

Prerequisites:

- Download and install the **Microsoft ODBC Driver for SQL Server** from the Microsoft website, see <https://docs.microsoft.com/en-us/SQL/connect/odbc/download-odbc-driver-for-sql-server>. This example uses **Microsoft ODBC Driver 17 for SQL Server** to connect to a **SQL Server 2016** database. You might want to download a different ODBC driver version, depending on the version of SQL Server where you want to connect. For information about ODBC driver versions supported by your SQL Server database, refer to the driver's system requirements.

To connect to SQL Server using ODBC:

1. [Start the database connection wizard](#) ⁴⁹³.
2. Click **ODBC Connections**.
3. Select **User DSN** (or **System DSN**, if you have administrative privileges), and then click **Add** .
4. Select the driver from the list. Note that the driver appears in the list only after it has been installed.



5. Click **User DSN** (or **System DSN** if you are creating a System DSN).

Creating a **System DSN** requires that StyleVision be run as an administrator. Therefore, in order to create a **System DSN**, cancel the wizard, make sure that you run StyleVision as an administrator, and perform the steps above again.

6. Enter a name and, optionally, a description to identify this connection, and then select from the list the SQL Server to which you are connecting (**SQLSERV01** in this example).

Microsoft SQL Server DSN Configuration

This wizard will help you create an ODBC data source that you can use to connect to SQL Server.

What name do you want to use to refer to the data source?

Name:

How do you want to describe the data source?

Description:

Which SQL Server do you want to connect to?

Server:

Finish Next > Cancel Help

7. If the database server was configured to allow connections from users authenticated on the Windows domain, select **With Integrated Windows authentication**. Otherwise, select one of the other options, as applicable. This example uses **With SQL Server authentication...**, which requires that the user name and password be entered in the relevant boxes.

Microsoft SQL Server

How should SQL Server verify the authenticity of the login ID?

With Integrated Windows authentication.
SPN (Optional):

With Azure Active Directory Integrated authentication.

With SQL Server authentication using a login ID and password entered by the user.

With Azure Active Directory Password authentication using a login ID and password entered by the user.

With Azure Active Directory Interactive authentication using a login ID entered by the user.

Login ID:

Password:

< Back Next > Cancel Help

8. Optionally, select the **Change the default database to** check box and enter the name of the database to which you are connecting (in this example, **Sandbox**).

Microsoft SQL Server

Change the default database to:
Sandbox

Mirror server:
SPN for mirror server (Optional):

Attach database filename:

Use ANSI quoted identifiers.
 Use ANSI nulls, paddings and warnings.

Application intent:
READWRITE

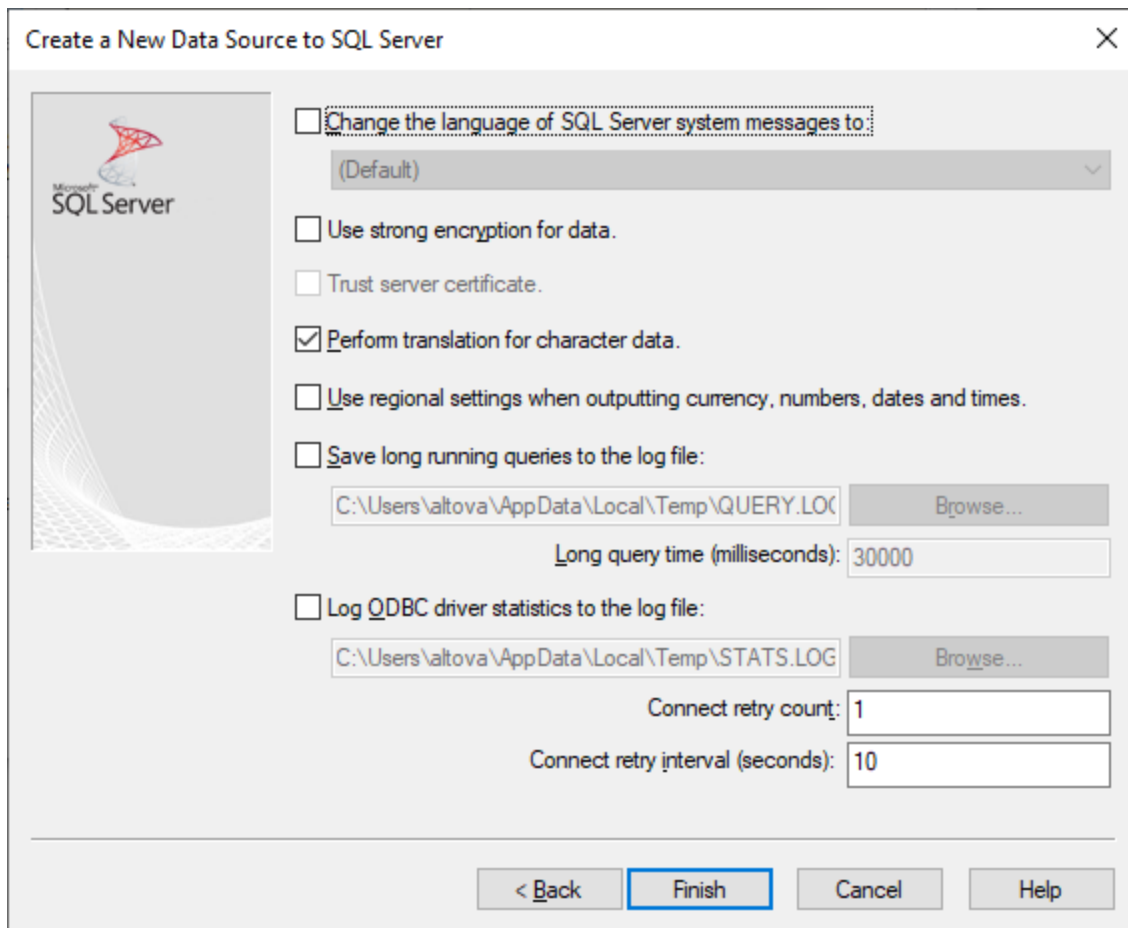
Multi-subnet failover.
 Transparent Network IP Resolution.
 Column Encryption.

Enclave Attestation Info:

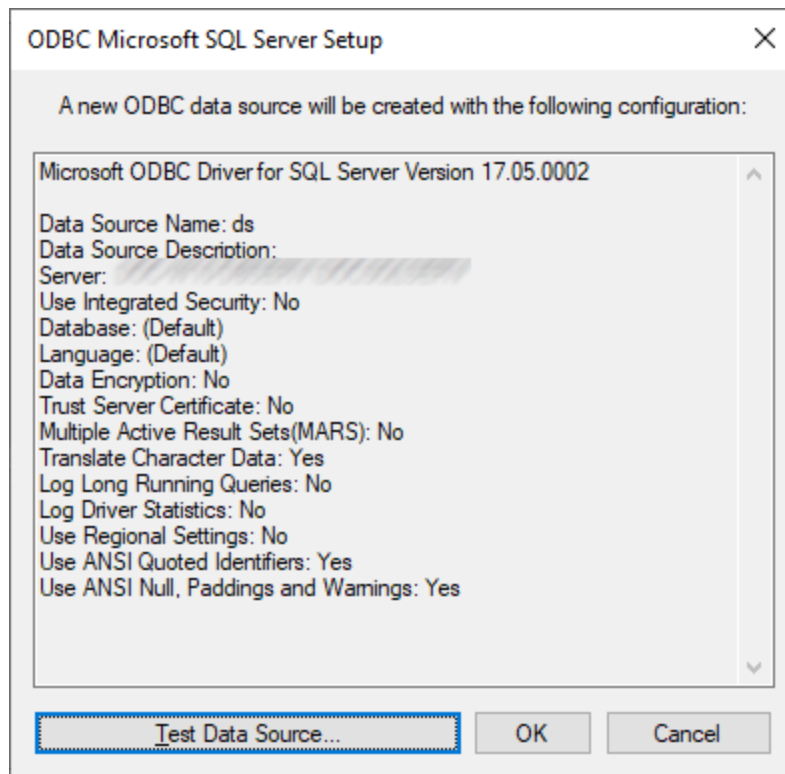
Use FMTONLY metadata discovery.

< Back Next > Cancel Help

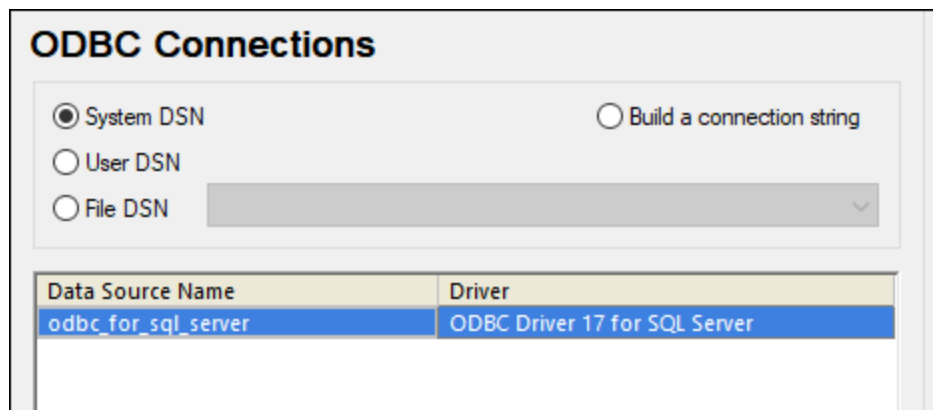
9. Click **Next** and, optionally, configure additional parameters for this connection.



10. Click **Finish**. A confirmation dialog box listing the connection details opens.



11. Click **OK**. The data source now appears in the list of **User** or **System** data sources, as configured, for example:



10.2.10.13 MySQL (ODBC)

This example illustrates how to connect to a MySQL database server from a Windows machine through the ODBC driver. The MySQL ODBC driver is not available on Windows, so it must be downloaded and installed separately. This example uses MySQL Connector/ODBC 8.0.

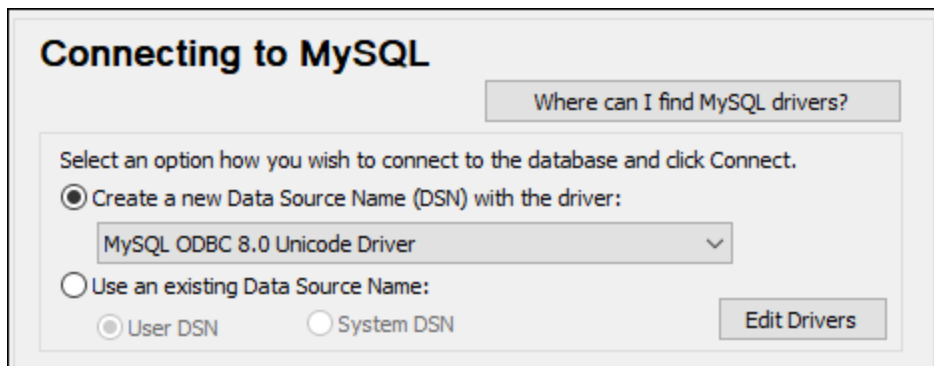
Prerequisites:

- MySQL ODBC driver must be installed on your operating system. Check the MySQL documentation for the driver version recommended for your database server version (see <https://dev.mysql.com/downloads/connector/odbc/>).
- You have the following database connection details: host, database, port, username, and password.

If you installed MySQL Connector/ODBC for 64-bit platform, make sure to install StyleVision for 64-bit platform as well.

To connect to MySQL via ODBC:

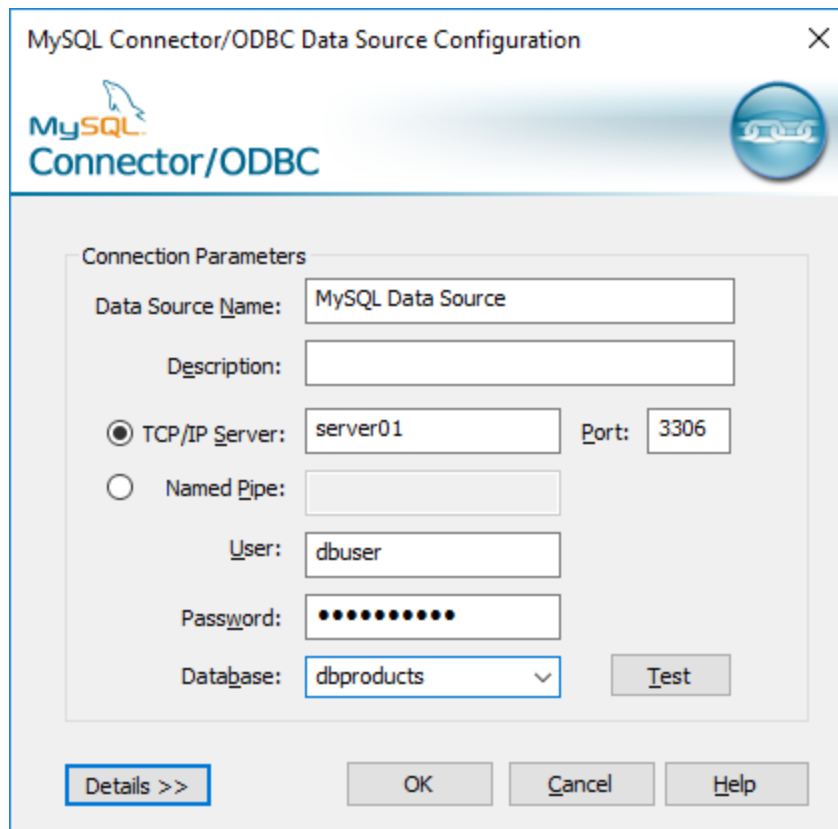
1. [Start the database connection wizard](#) ⁴⁹³.
2. Select **MySQL (ODBC)**, and then click **Next**.



3. Select **Create a new Data Source Name (DSN) with the driver**, and select a MySQL driver. If no MySQL driver is available in the list, click **Edit Drivers**, and select any available MySQL drivers (the list contains all ODBC drivers installed on your operating system).

If you installed StyleVision 64-bit, then the 64-bit ODBC drivers are shown in the list. Otherwise, the 32-bit ODBC drivers are shown. See also [Viewing the Available ODBC Drivers](#) ⁵¹².

4. Click **Connect**.



5. In the Data Source Name box, enter a descriptive name that will help you identify this ODBC data source in future.
6. Fill in the database connection credentials (TCP/IP Server, User, Password), select a database, and then click **OK**.

Note: If the database server is remote, it must be configured by the server administrator to accept remote connections from your machine's IP address. Also, if you click **Details>>**, there are several additional parameters available for configuration. Check the driver's documentation before changing their default values.

10.2.10.14 Oracle (JDBC)

This example shows you how to connect to an Oracle database server from a client machine, using the JDBC interface. The connection is created as a pure Java connection, using the **Oracle Instant Client Package (Basic)** available from the Oracle website. The advantage of this connection type is that it requires only the Java environment and the .jar libraries supplied by the Oracle Instant Client Package, saving you the effort to install and configure a more complex database client.

Prerequisites:

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. StyleVision will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you

may have set in application **Options**; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.

- Make sure that the platform of StyleVision (32-bit, 64-bit) matches that of the JRE/JDK.
- The **Oracle Instant Client Package (Basic)** must be available on your operating system. The package can be downloaded from the official Oracle website. This example uses Oracle Instant Client Package version 12.1.0.2.0, for Windows 32-bit and, consequently, Oracle JDK 32-bit.
- You have the following database connection details: host, port, service name, username, and password.

To connect to Oracle through the Instant Client Package:

1. [Start the database connection wizard](#)⁴⁹³.
2. Click **JDBC Connections**.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the required .jar file is located at the following path: **C:\jdbc\instantclient_12_1\ojdbc7.jar**. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also [Configuring the CLASSPATH](#)⁵¹⁶).
4. In the "Driver" box, select either **oracle.jdbc.OracleDriver** or **oracle.jdbc.driver.OracleDriver**. Note that these entries are available if a valid .jar file path is found either in the "Classpaths" text box, or in the operating system's CLASSPATH environment variable (see the previous step).
5. Enter the username and password to the database in the corresponding text boxes.

Classpaths: C:\jdbc\instantclient_12_1\ojdbc7.jar

Driver: oracle.jdbc.driver.OracleDriver

Username: johndoe

Password: ●●●●●●

Database URL: jdbc:oracle:thin:@//ora12c:1521/orcl12c

Connect Close

6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

```
jdbc:oracle:thin:@//host:port:service
```

7. Click **Connect**.

10.2.10.15 Oracle (ODBC)

This example illustrates a common scenario where you connect from StyleVision to an Oracle database server on a network machine, through an Oracle database client installed on the local operating system.

The example includes instructions for setting up an ODBC data source (DSN) using the database connection wizard in StyleVision. If you have already created a DSN, or if you prefer to create it directly from the **ODBC Data Source administrator** in Windows, you can do so, and then select it when prompted by the wizard. For more information about ODBC data sources, see [Setting up an ODBC Connection](#)⁵¹⁰.

Prerequisites:

- The Oracle database client (which includes the ODBC Oracle driver) must be installed and configured on your operating system. For instructions on how to install and configure an Oracle database client, refer to the documentation supplied with your Oracle software.
- The **tnsnames.ora** file located in Oracle home directory contains an entry that describes the database connection parameters, in a format similar to this:

```
ORCL =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP) (HOST = server01) (PORT = 1521))
    )
    (CONNECT_DATA =
      (SID = orcl)
      (SERVER = DEDICATED)
    )
  )
```

The path to the **tnsnames.ora** file depends on the location where Oracle home directory was installed. For Oracle database client 11.2.0, the default Oracle home directory path could be as follows:

```
C:\app\username\product\11.2.0\client_1\network\admin\tnsnames.ora
```

You can add new entries to the **tnsnames.ora** file either by pasting the connection details and saving the file, or by running the Oracle *Net Configuration Assistant* wizard (if available). If you want these values to appear in dropdown lists during the configuration process, then you may need to add the path to the admin folder as a **TNS_ADMIN** environment variable.

To connect to Oracle using ODBC:

1. [Start the database connection wizard](#)⁴⁹³.
2. Select **Oracle (ODBC / JDBC)**, and then click **Next**.

JDBC vs. ODBC

JDBC

Java-based connection that may provide support for more modern features of your database that aren't available via ODBC. The tradeoff for these features is a potential sacrifice of performance.

ODBC

An ODBC connection will generally be faster and less resource-intensive than a JDBC connection, but lacks support for more modern database features (such as native XML types).

< Back Next > Close

3. Select **ODBC**.

Connecting to Oracle

Where can I find Oracle drivers?

Select an option how you wish to connect to the database and click Connect.

Create a new Data Source Name (DSN) with the driver:
Microsoft ODBC for Oracle

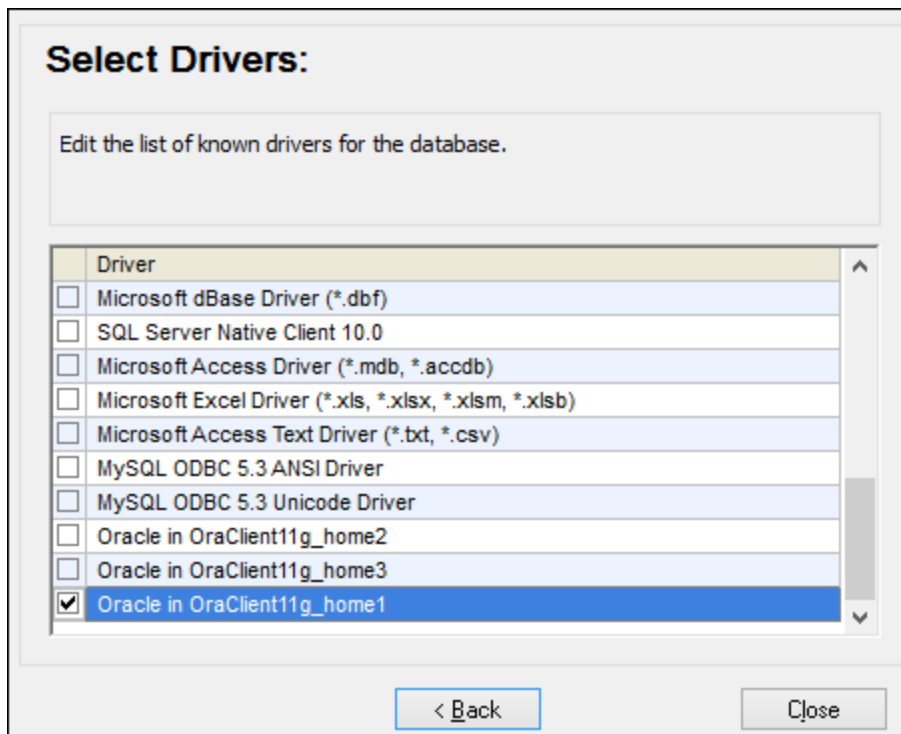
Use an existing Data Source Name:
 User DSN System DSN Edit Drivers

Data Source Name

Skip the configuration step for wizard

< Back Connect Close

4. Click **Edit Drivers**.



5. Select the Oracle drivers you wish to use (in this example, **Oracle in OraClient11g_home1**). The list displays the Oracle drivers available on your system after installation of Oracle client.
6. Click **Back**.
7. Select **Create a new data source name (DSN) with the driver**, and then select the Oracle driver chosen in step 4.

Connecting to Oracle Where can I find Oracle drivers?

Select an option how you wish to connect to the database and click Connect.

Create a new Data Source Name (DSN) with the driver:
Oracle in OraClient11g_home1

Use an existing Data Source Name:
 User DSN System DSN Edit Drivers

Skip the configuration step for wizard

< Back Connect Close

Avoid using the Microsoft-supplied driver called **Microsoft ODBC for Oracle** driver. Microsoft recommends using the ODBC driver provided by Oracle (see <http://msdn.microsoft.com/en-us/library/ms714756%28v=vs.85%29.aspx>)

8. Click **Connect**.

9. In the Data Source Name text box, enter a name to identify the data source (in this example, **Oracle DSN 1**).
10. In the TNS Service Name box, enter the connection name as it is defined in the **tnsnames.ora** file (see [prerequisites](#)⁵⁵⁵). In this example, the connection name is **ORCL**. *Note:* If you wish to have the dropdown list of the combo box populated with the values of the **tnsnames.ora** file, then you may need to add the path to the admin folder as a **TNS_ADMIN** environment variable.
11. Click **OK**.

12. Enter the username and password to the database, and then click OK.

10.2.10.16 PostgreSQL (ODBC)

This example illustrates how to connect to a PostgreSQL database server from a Windows machine through the ODBC driver. The PostgreSQL ODBC driver is not available on Windows, so it must be downloaded and


installed separately. This example uses the psqLODBC driver (version 11.0) downloaded from the official website (see also [Database Drivers Overview](#)⁴⁹⁵).

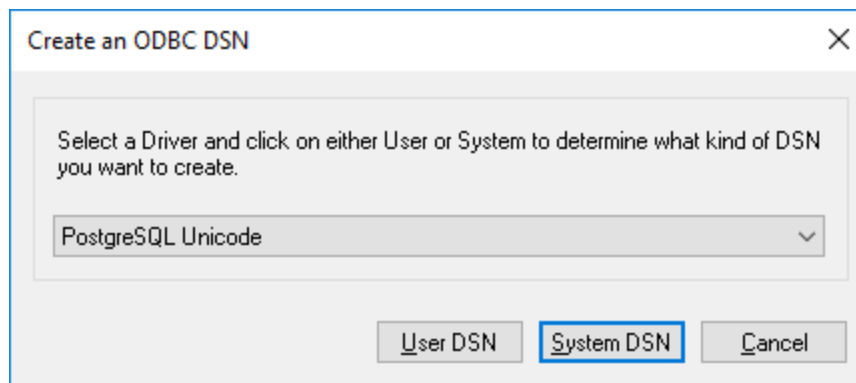
Note: You can also connect to a PostgreSQL database server directly (without the ODBC driver), see [Setting up a PostgreSQL Connection](#)⁵¹⁹.

Prerequisites:

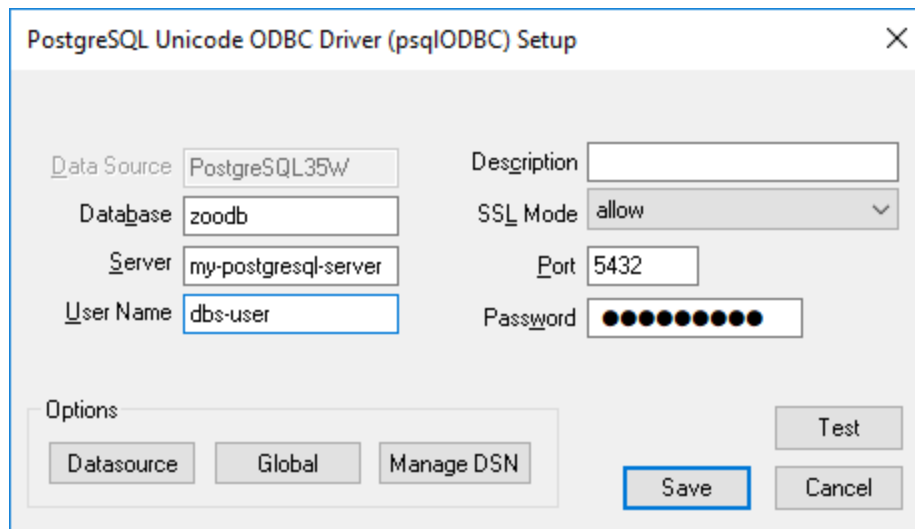
- *psqLODBC* driver must be installed on your operating system.
- You have the following database connection details: server, port, database, user name, and password.

To set up a connection to PostgreSQL using ODBC:

1. [Start the database connection wizard](#)⁴⁹³.
2. Click **ODBC Connections**.
3. Select the **User DSN** option.
4. Click **Create a new DSN**  and select the driver from the drop-down list. If no PostgreSQL driver is available in the list, make sure that the PostgreSQL ODBC driver is installed on your operating system, as mentioned in the prerequisites above.



5. Click **User DSN**.

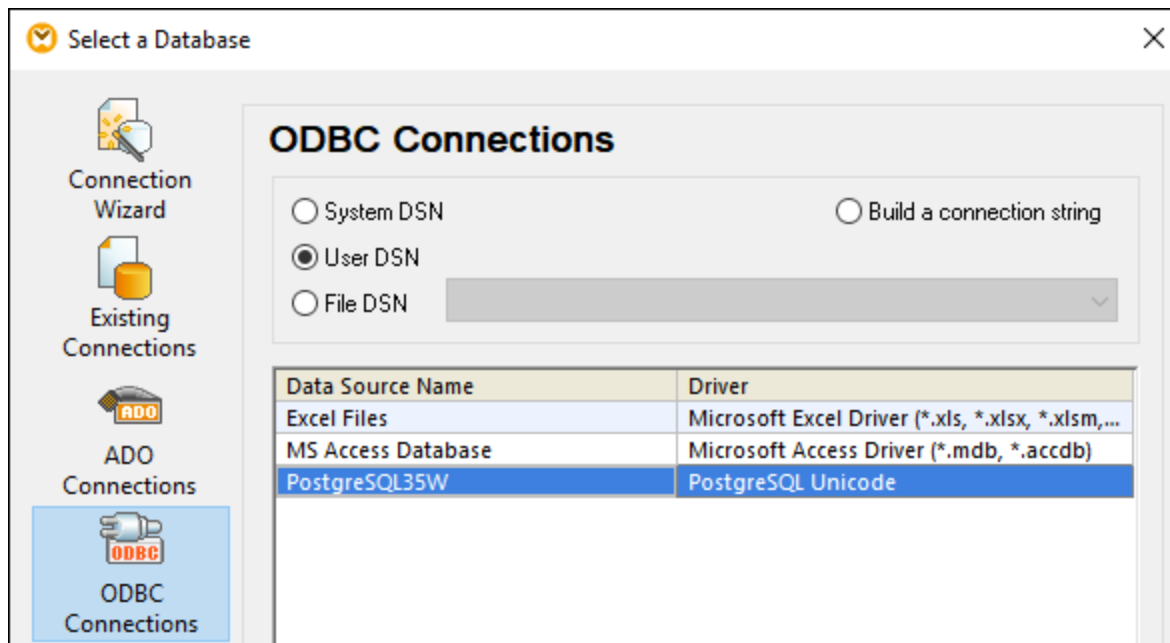


The screenshot shows the 'PostgreSQL Unicode ODBC Driver (psqlODBC) Setup' dialog box. It contains the following fields and controls:

- Data Source:** PostgreSQL35W
- Description:** (empty text box)
- Database:** zoodb
- SSL Mode:** allow (dropdown menu)
- Server:** my-postgresql-server
- Port:** 5432
- User Name:** dbs-user
- Password:** (password field with 10 dots)
- Options:** Datasource, Global, Manage DSN (checkboxes)
- Buttons:** Test, Save, Cancel

6. Fill in the database connection credentials (these must be supplied by the database owner), and then click **Save**.

The connection is now available in the list of ODBC connections. To connect to the database, you can either double-click the connection or select it, and then click **Connect**.



10.2.10.17 Progress OpenEdge (JDBC)

This example illustrates how to connect to a Progress OpenEdge 11.6 database server through JDBC.

Prerequisites

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. StyleVision will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.
- Make sure that the platform of StyleVision (32-bit, 64-bit) matches that of the JRE/JDK.
- The operating system's `PATH` environment variable must include the path to the `bin` directory of the JRE or JDK installation directory, for example `C:\Program Files (x86)\Java\jre1.8.0_51\bin`.
- The Progress OpenEdge JDBC driver must be available on your operating system. In this example, JDBC connectivity is provided by the **openedge.jar** and **pool.jar** driver component files available in **C:\Progress\OpenEdge\java** as part of the OpenEdge SDK installation.
- You have the following database connection details: host, port, database name, username, and password.

Connecting to OpenEdge through JDBC

1. [Start the database connection wizard](#) ⁴⁹³.
2. Click **JDBC Connections**.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the required .jar file paths are: `C:\Progress\OpenEdge\java\openedge.jar;C:\Progress\OpenEdge\java\pool.jar`. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also [Configuring the CLASSPATH](#) ⁵¹⁶).
4. In the "Driver" box, select **com.ddtek.jdbc.openedge.OpenEdgeDriver**. Note that this entry is available if a valid .jar file path is found either in the "Classpaths" text box, or in the operating system's CLASSPATH environment variable (see the previous step).

The screenshot shows a configuration window for the Progress OpenEdge ODBC driver. It contains the following fields:

- Classpaths:** C:\Progress\OpenEdge\java\openedge.jar;C:\Progress\OpenEd;
- Driver:** com.ddtek.jdbc.openedge.OpenEdgeDriver
- Username:** dbuser
- Password:** (masked with six dots)
- Database URL:** jdbc:datadirect:openedge://localhost:8910;databaseName=obpsdev

Buttons at the bottom: **Connect** and **Close**.

5. Enter the username and password to the database in the corresponding text boxes.
6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

```
jdbc:datadirect:openedge://host:port;databaseName=db_name
```

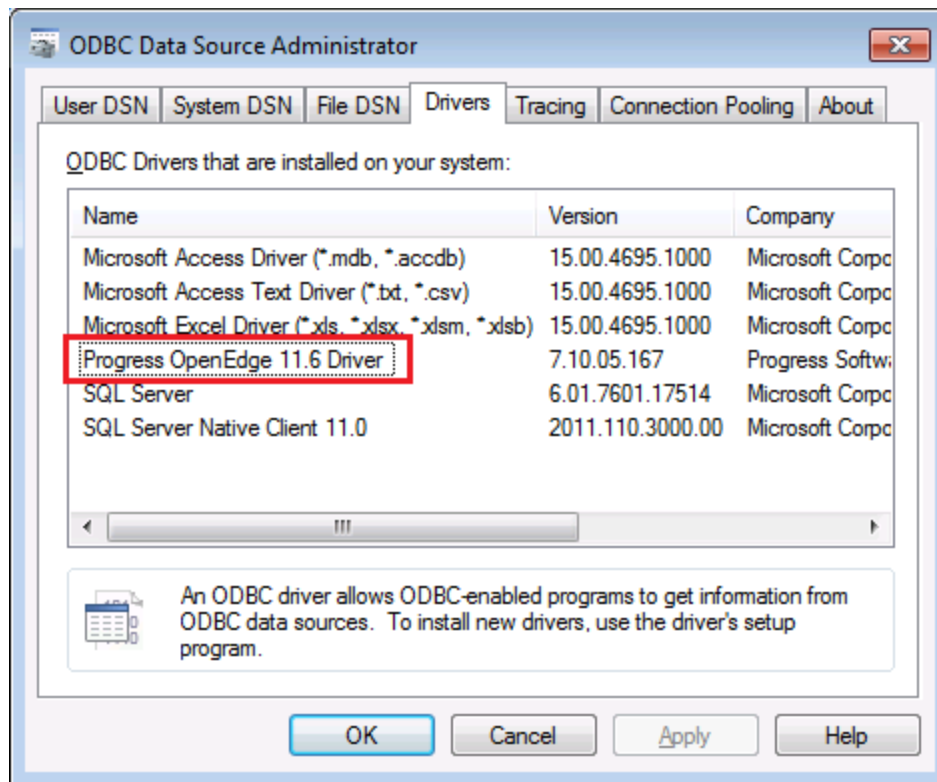
7. Click **Connect**.

10.2.10.18 Progress OpenEdge (ODBC)

This example illustrates how to connect to a Progress OpenEdge database server through the Progress OpenEdge 11.6 ODBC driver.


Prerequisites:

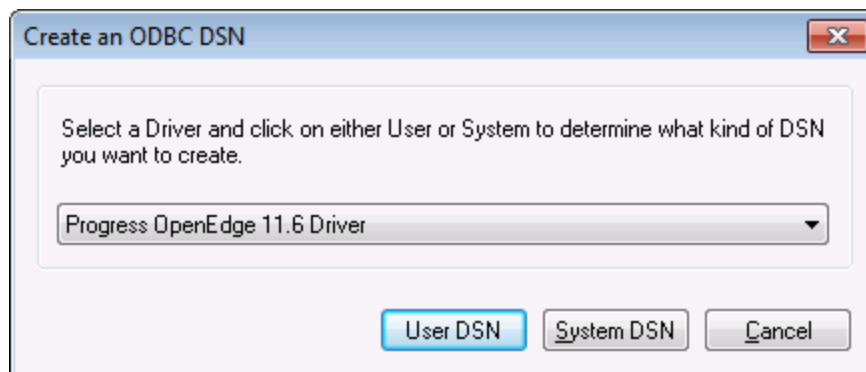
- The *ODBC Connector for Progress OpenEdge* driver must be installed on your operating system. The Progress OpenEdge ODBC driver can be downloaded from the vendor's website (see also [Database Drivers Overview](#)⁴⁹⁵). Make sure to download the 32-bit driver when running the 32-bit version of StyleVision, and the 64-bit driver when running the 64-bit version. After installation, check if the ODBC driver is available on your machine (see also [Viewing the Available ODBC Drivers](#)⁶¹²).



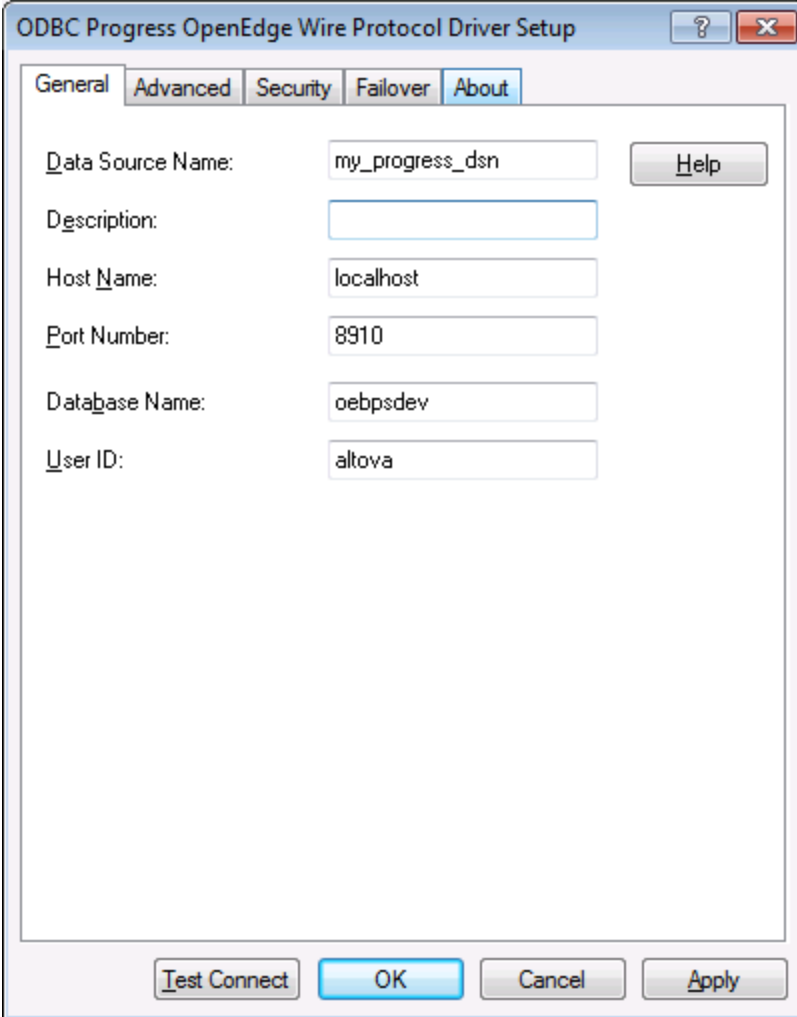
- You have the following database connection details: host name, port number, database name, user ID, and password.

Connecting to Progress OpenEdge through ODBC

1. [Start the database connection wizard](#) ⁴⁹³.
2. Click **ODBC Connections**.
3. Click **User DSN** (alternatively, click **System DSN**, or **File DSN**, in which case the subsequent instructions will be similar).
4. Click **Add** .
5. Select the **Progress OpenEdge Driver** from the list, and click **User DSN** (or **System DSN**, if applicable).



6. Fill in the database connection credentials (Database, Server, Port, User Name, Password), and then click **OK**. To verify connectivity before saving the entered data, click **Test Connect**.

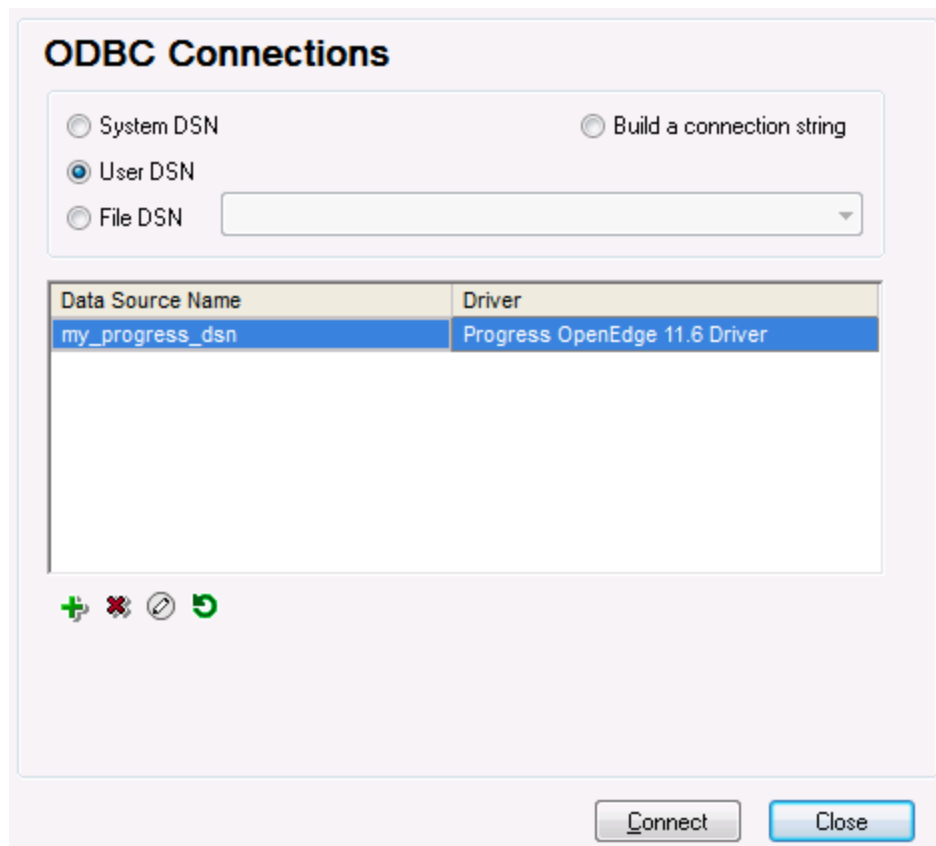


The screenshot shows the 'ODBC Progress OpenEdge Wire Protocol Driver Setup' dialog box with the 'General' tab selected. The fields are filled with the following values:

Field	Value
Data Source Name:	my_progress_dsn
Description:	
Host Name:	localhost
Port Number:	8910
Database Name:	oebpsdev
User ID:	altova

Buttons visible: Help, Test Connect, OK, Cancel, Apply.

7. Click OK. The new data source now appears in the list of ODBC data sources.



8. Click **Connect**.

10.2.10.19 Sybase (JDBC)

This example illustrates how to connect to a Sybase database server through JDBC.

Prerequisites:

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. StyleVision will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.
- Make sure that the platform of StyleVision (32-bit, 64-bit) matches that of the JRE/JDK.
- Sybase *jConnect* component must be installed on your operating system (in this example, *jConnect 7.0* is used, installed as part of the *Sybase Adaptive Server Enterprise PC Client* installation). For the installation instructions of the database client, refer to Sybase documentation.
- You have the following database connection details: host, port, database name, username, and password.

To connect to Sybase through JDBC:

1. [Start the database connection wizard](#) ⁴⁹³.
2. Click **JDBC Connections**.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the required .jar file path is: **C:\sybase\jConnect-7_0\classes\jconn4.jar**. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also [Configuring the CLASSPATH](#) ⁵¹⁶).
4. In the "Driver" box, select **com.sybase.jdbc4.jdbc.SybDriver**. Note that this entry is available if a valid .jar file path is found either in the "Classpaths" text box, or in the operating system's CLASSPATH environment variable (see the previous step).

Classpaths: C:\sybase\jConnect-7_0\classes\jconn4.jar;

Driver: com.sybase.jdbc4.jdbc.SybDriver

Username: dbuser

Password: ●●●●●●

Database URL: jdbc:sybase:Tds:SYBASE12:2048/PRODUCTSDB

Connect Close

5. Enter the username and password to the database in the corresponding text boxes.
6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

```
jdbc:sybase:Tds:hostName:port/databaseName
```

7. Click **Connect**.

10.2.10.20 Teradata (JDBC)

This example illustrates how to connect to a Teradata database server through JDBC.

Prerequisites:

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. StyleVision will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.
- Make sure that the platform of StyleVision (32-bit, 64-bit) matches that of the JRE/JDK.
- The JDBC driver (one or more .jar files that provide connectivity to the database) must be available on your operating system. In this example, Teradata JDBC Driver 16.20.00.02 is used. For more information, see <https://downloads.teradata.com/download/connectivity/jdbc-driver>.
- You have the following database connection details: host, database, port, username, and password.

To connect to Teradata through JDBC:

1. [Start the database connection wizard](#)⁴⁹³.
2. Click **JDBC Connections**.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the .jar files are located at the following path: **C:\jdbc\teradata**. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also [Configuring the CLASSPATH](#)⁵¹⁶).
4. In the "Driver" box, select **com.teradata.jdbc.TeraDriver**. Note that this entry is available if a valid .jar file path is found either in the "Classpath" text box, or in the operating system's CLASSPATH environment variable (see the previous step).

JDBC Connections

Enter a connection string and select (or enter manually) a valid JDBC driver. Click on 'Connect' to proceed.

Classpaths:

Driver:

Username:

Password:

Database URL:

5. Enter the username and password to the database in the corresponding text boxes.
6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted value with the one applicable to your database server.

```
jdbc:teradata://databaseServerName
```

7. Click **Connect**.

10.2.10.21 Teradata (ODBC)

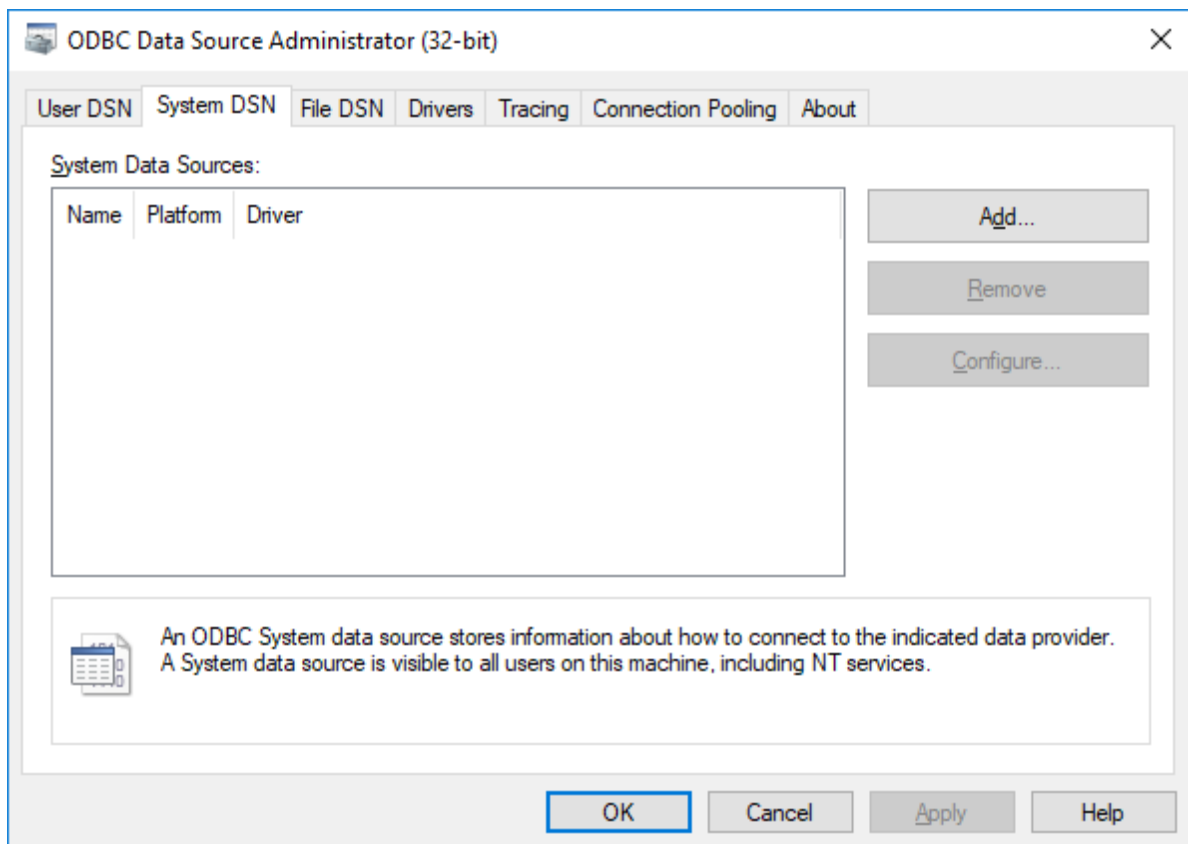
This example illustrates how to connect to a Teradata database server through ODBC.

Prerequisites:

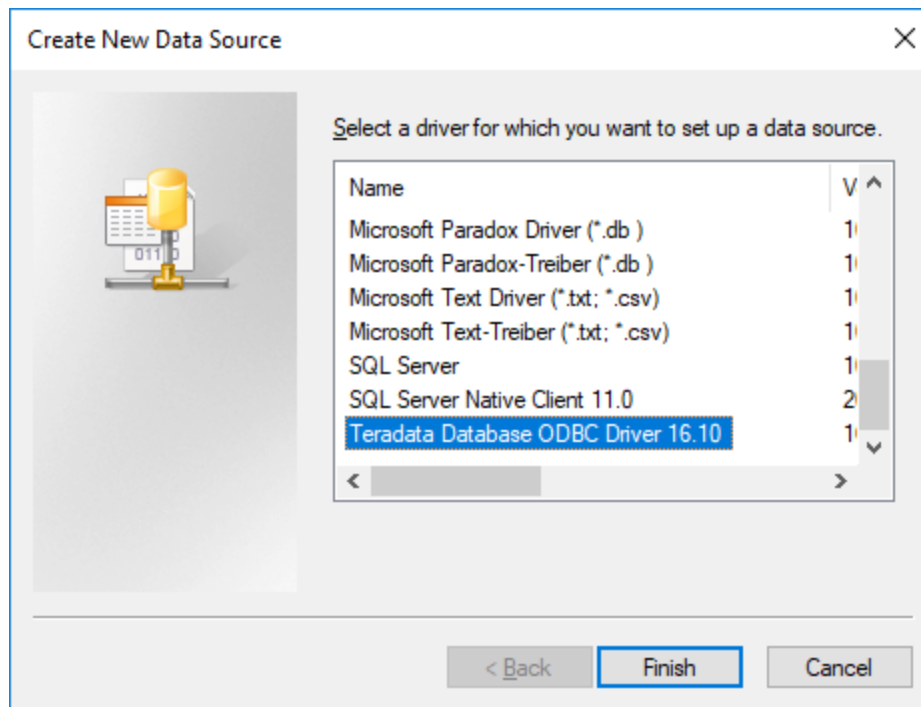
- The Teradata ODBC driver must be installed (see <https://downloads.teradata.com/download/connectivity/odbc-driver/windows>. This example uses Teradata ODBC Driver for Windows version 16.20.00.
- You have the following database connection details: host, username, and password.

To connect to Teradata through ODBC:

1. Press the **Windows** key, start typing "ODBC", and select **Set up ODBC data sources (32-bit)** from the list of suggestions. If you have a 64-bit ODBC driver, select **Set up ODBC data sources (64-bit)** and use 64-bit StyleVision in the subsequent steps.



2. Click the **System DSN** tab, and then click **Add**.

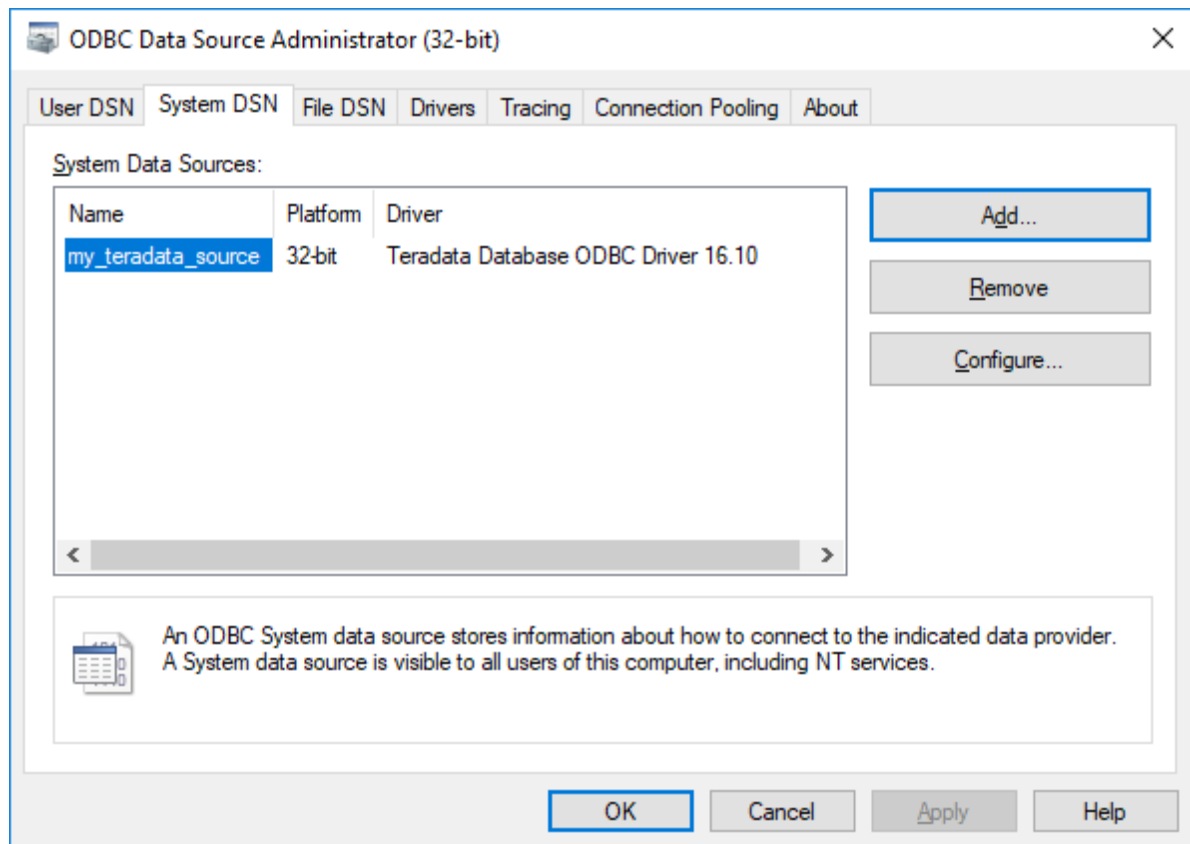


3. Select **Teradata Database ODBC Driver** and click **Finish**.

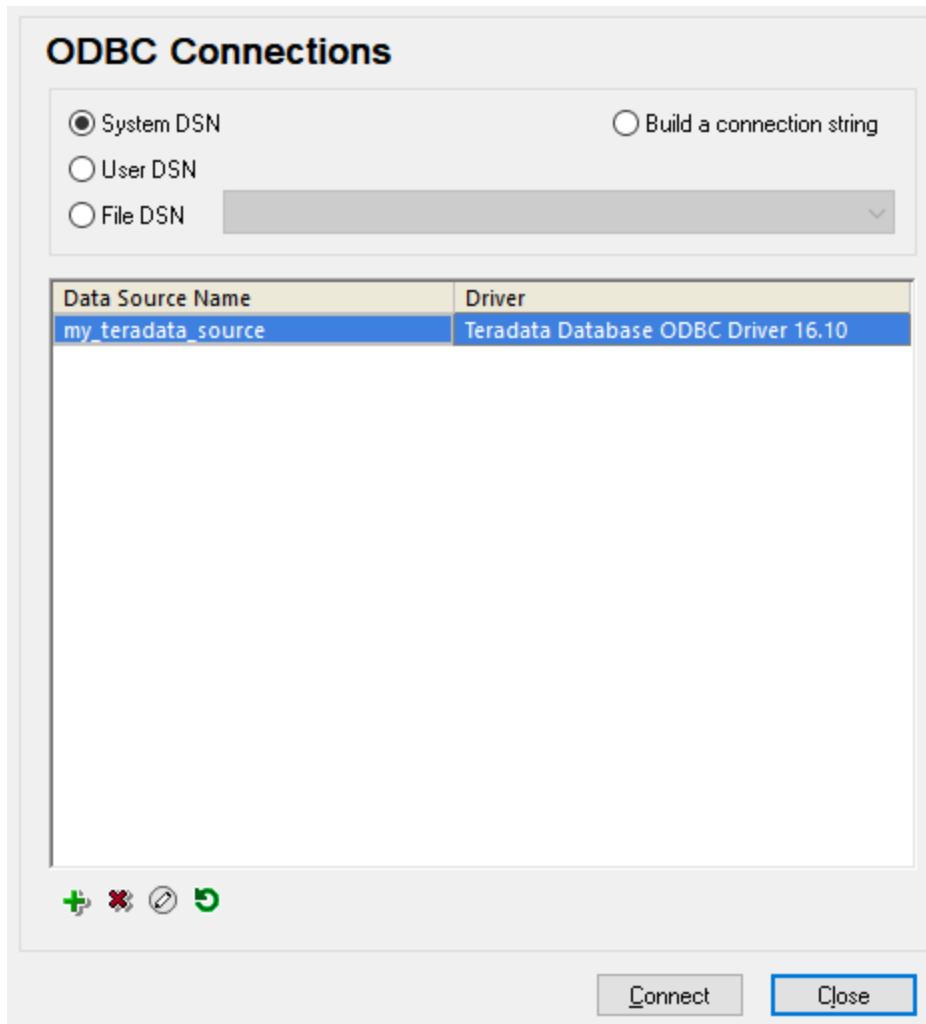
The screenshot shows the 'ODBC Driver Setup for Teradata Database' dialog box. It is divided into several sections:

- Data Source:** Contains a 'Name' field with the value 'my_teradata_source' and an empty 'Description' field. To the right are 'OK', 'Cancel', and 'Help' buttons.
- Teradata Server Info:** Contains a 'Name or IP address' dropdown menu with 'demoserver' selected.
- Authentication:** Features a 'Use Integrated Security' checkbox (unchecked), a 'Mechanism' dropdown, a 'Parameter' field with a 'Change...' button, a 'Username' field with 'demouser', and two radio buttons: 'Password' (selected) with a masked password field, and 'Teradata Wallet String' with an empty field.
- Optional:** Contains a 'Default Database' field and an 'Account String' field with an 'Options >>' button.
- Session Character Set:** A dropdown menu with 'UTF8' selected.

4. Enter name and, optionally, a description that will help you identify this ODBC data source in future. Also, enter the database connection credentials (Database server, User, Password), and, optionally, select a database.
5. Click **OK**. The data source now appears in the list.



6. Run StyleVision and [start the database connection wizard](#) ⁴⁹³.
7. Click **ODBC Connections**.



8. Click **System DSN**, select the data source created previously, and then click **Connect**.

Note: If you get the following error: "The driver returned invalid (or failed to return) SQL_DRIVER_ODBC_VER: 03.80", make sure that the path to the ODBC client (for example, **C:\Program Files\Teradata\Client\16.10\bin**, if you installed it to this location) exists in your system's PATH environment variable. If this path is missing, add it manually.

10.3 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:

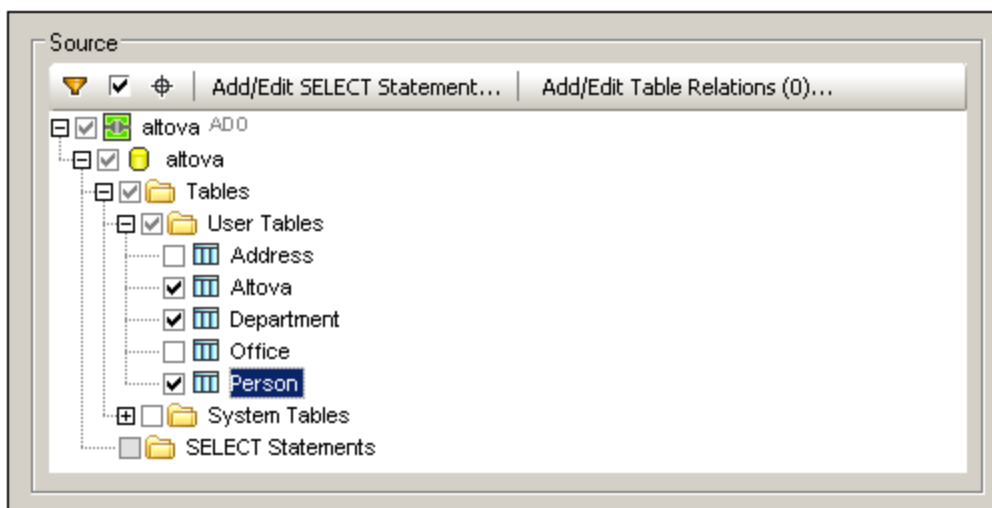
- In the case of [non-XML databases](#)⁵⁷⁵, you select the table/s for which the SPS is being created. StyleVision automatically generates: (i) a schema based on the structure of the table/s, and (ii) temporary XML files based on this schema and containing the data in the selected table/s. How to select the tables is described in the section [DB Data Selection | Non-XML Databases](#)⁵⁷⁵.
- In the case of [XML databases](#)⁵⁸², you must do two things. First, [select the XML cell](#)⁵⁸² of the DB in which the required XML data is stored. This XML data is loaded as the [Working XML File](#)²⁸ of the SPS. Second, [select the schema](#)⁵⁸² 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](#)⁵⁸².

10.3.1 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.



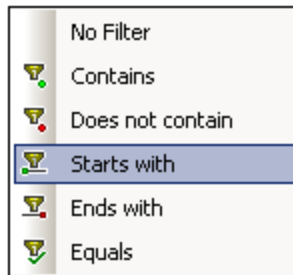
The toolbar of the Source pane (*screenshot below*) contains three icons, respectively, from left to right: **Filter Folder Contents**, **Checked Objects Only**, and **Object Locator**. The **Checked Objects Only** icon toggles the display between all tables and checked tables.



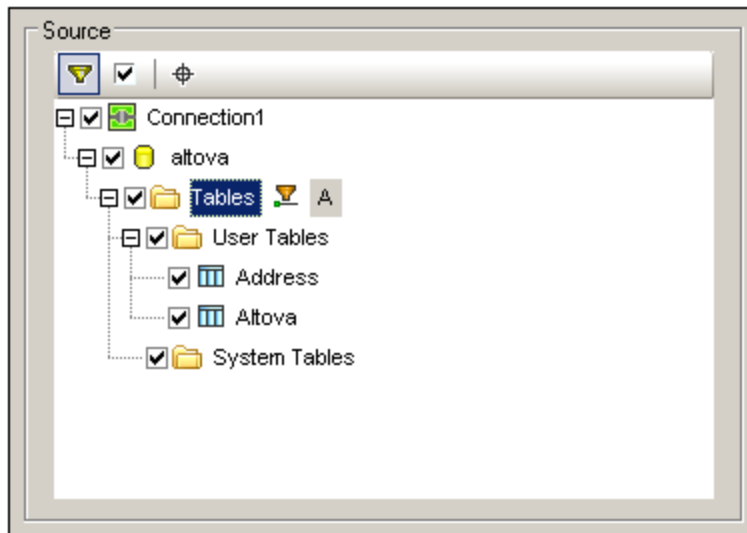
Filtering folder contents

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.



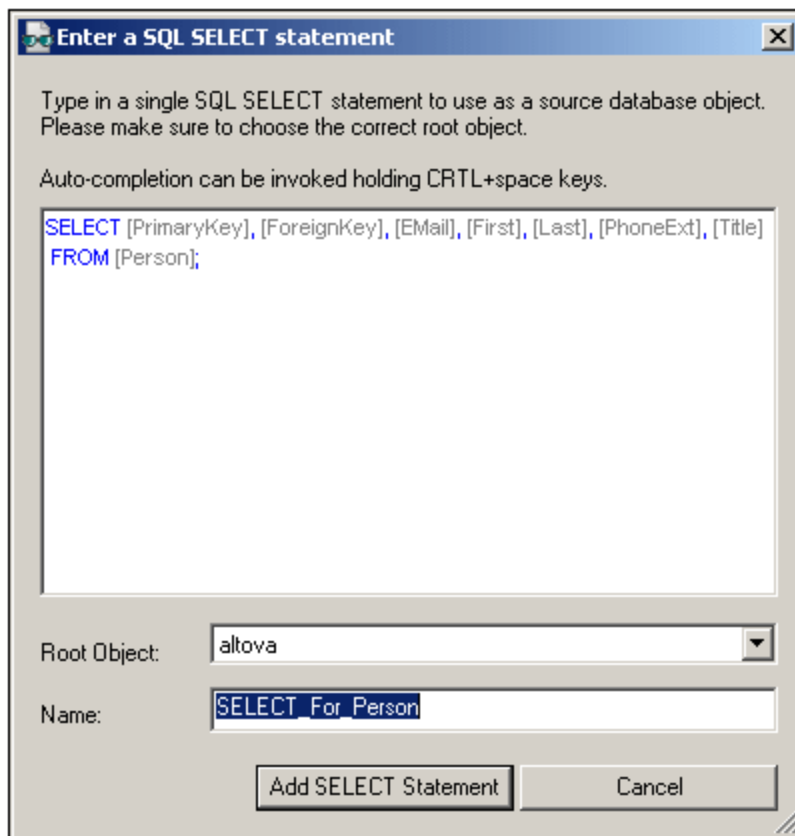
3. Click the object in the list to see it in the Source pane.

Adding and editing SELECT statements for local views

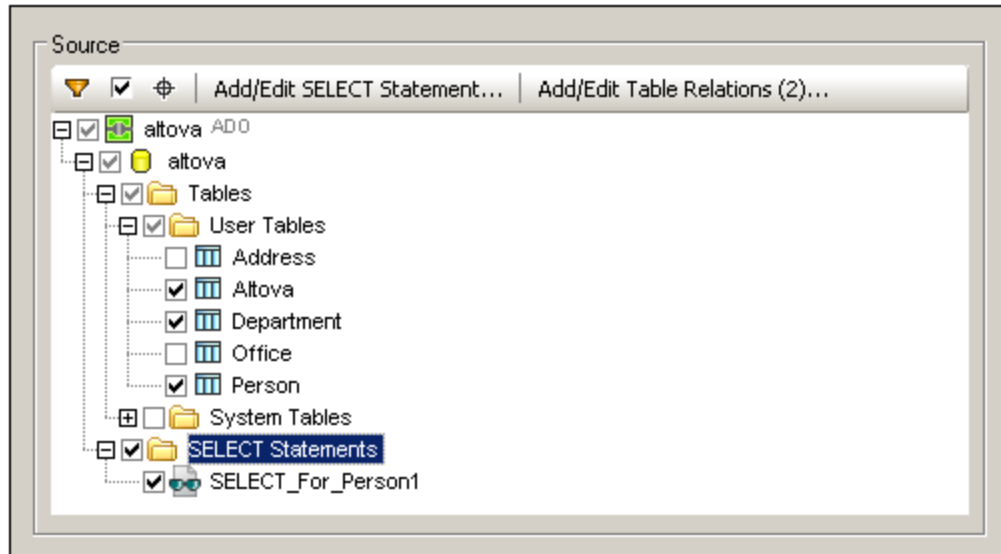
You can create `SELECT` statements in SQL to create local views. When the schema is generated from a DB connection which has local views (or `SELECT` statements) defined for it, the schema that is generated for the DB will contain a table for each `SELECT` statement.

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*).

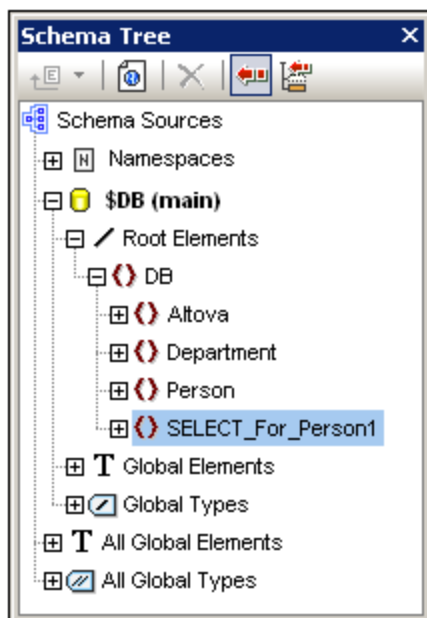


2. Enter the `SELECT` statement. If you connect to an Oracle or IBM DB2 database using JDBC, then the `SELECT` statement must have no final semicolon. If you wish to create a `SELECT` statement for an entire table, right-click the table in the Insert Database Objects dialog and select the context menu 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*).

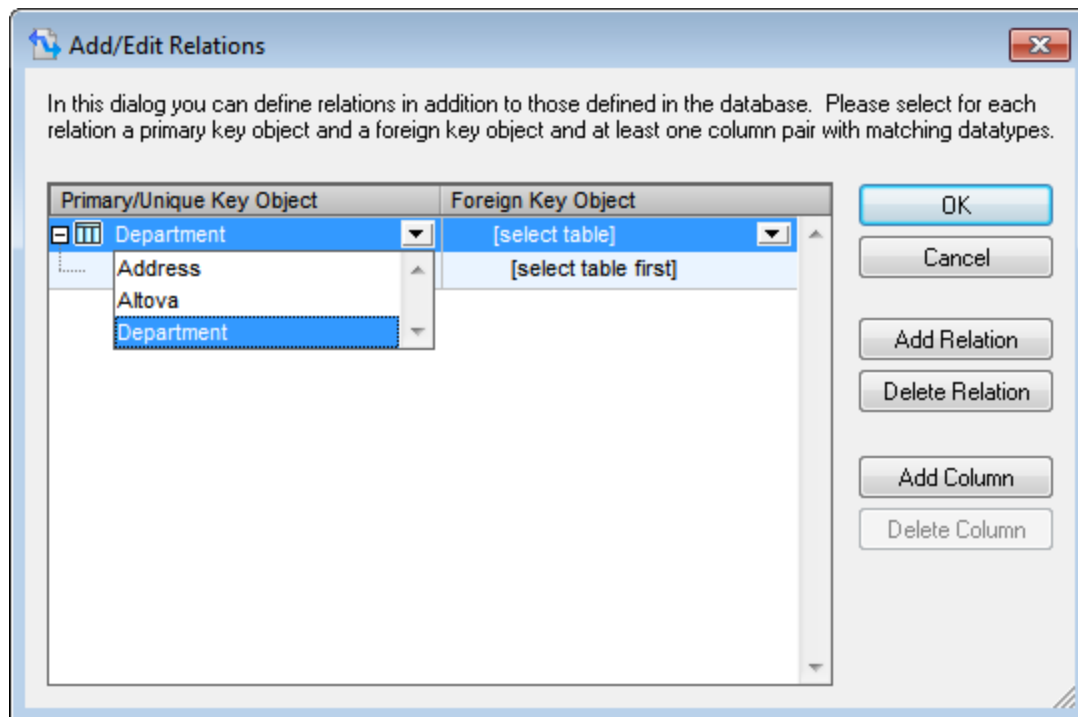


Local relations between tables

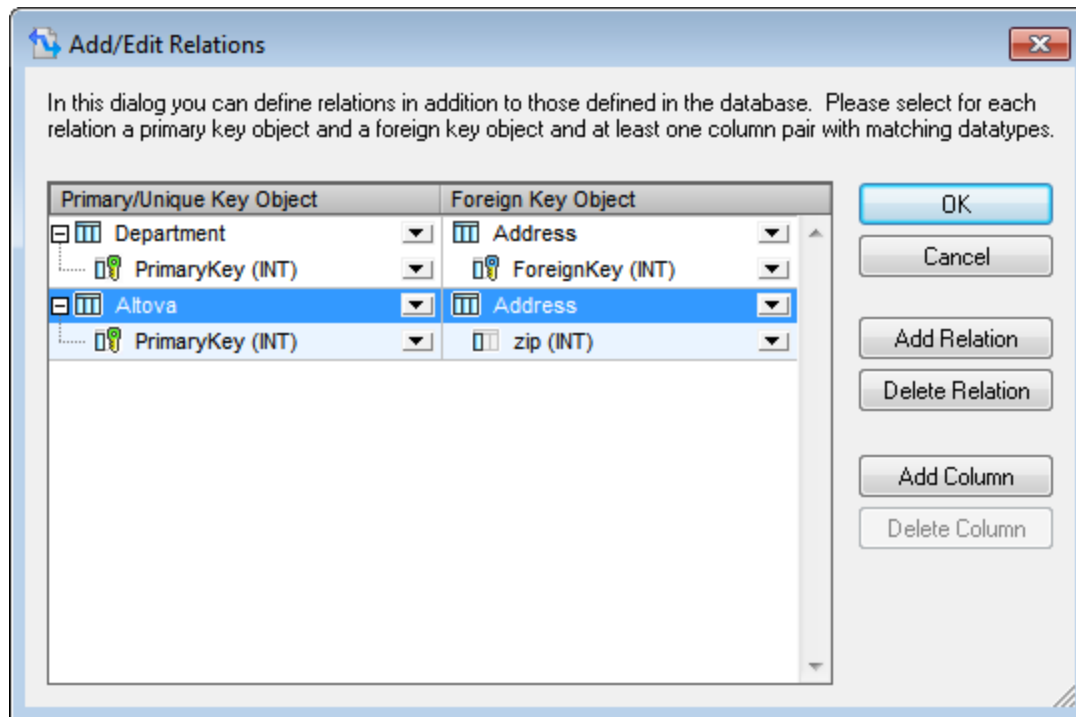
You can create local relations between two tables, similar to the primary-key/foreign-key kind of relationship. The relation is local, in StyleVision, which means that the database itself does not have to be modified. The local relationship will be represented in the generated schema.

To create a local relation, do the following:

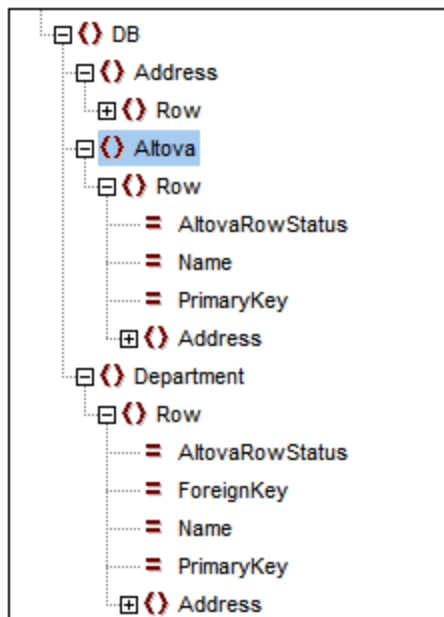
1. In the Insert Database Objects dialog, click the **Add/Edit Relations** tab. This pops up the Add/Edit Table Relations dialog (see *screenshot below*).
2. Click the **Add Relation** button, and in the Primary/Unique Key column, click the dropdown button of the Select Table combo box (*screenshot below*). 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.



- Click **OK** to complete the relation. When the schema is generated, it will reflect the newly created relations.

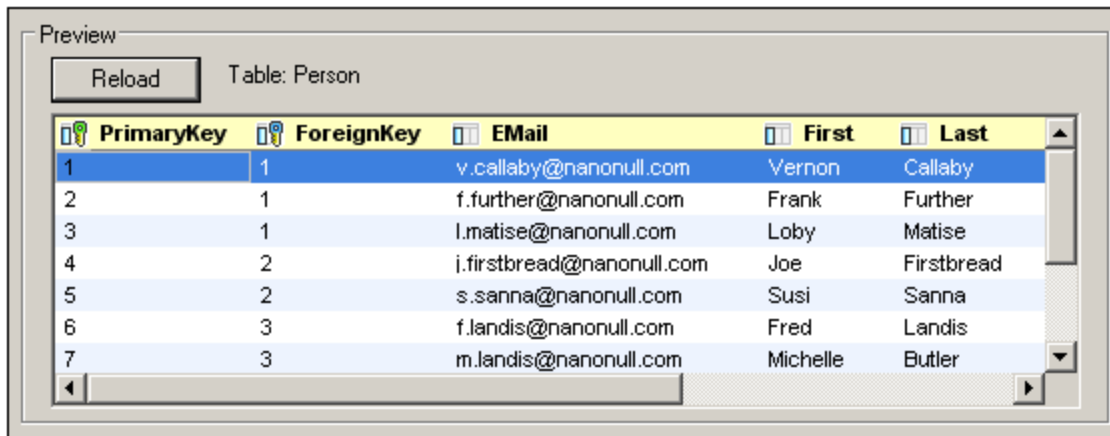


Look at the two screenshots above and see how, in the generated schema, `Altova` and `Department` each contain `Address`. Those `Department` rows with `PrimaryKey` values equal to the `ForeignKey` value of the `Address` row will be output. And those `Altova` rows with `PrimaryKey` values equal to the `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*).

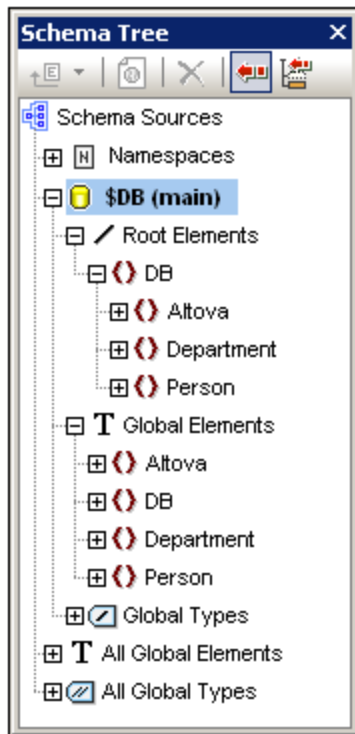


The screenshot shows a 'Preview' window with a 'Reload' button and the text 'Table: Person'. Below this is a table with the following data:

PrimaryKey	ForeignKey	EMail	First	Last
1	1	v.callaby@nanonull.com	Vernon	Callaby
2	1	f.further@nanonull.com	Frank	Further
3	1	l.matise@nanonull.com	Loby	Matise
4	2	j.firstbread@nanonull.com	Joe	Firstbread
5	2	s.sanna@nanonull.com	Susi	Sanna
6	3	f.landis@nanonull.com	Fred	Landis
7	3	m.landis@nanonull.com	Michelle	Butler

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](#)⁵⁸⁶. 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.

10.3.2 XML Databases

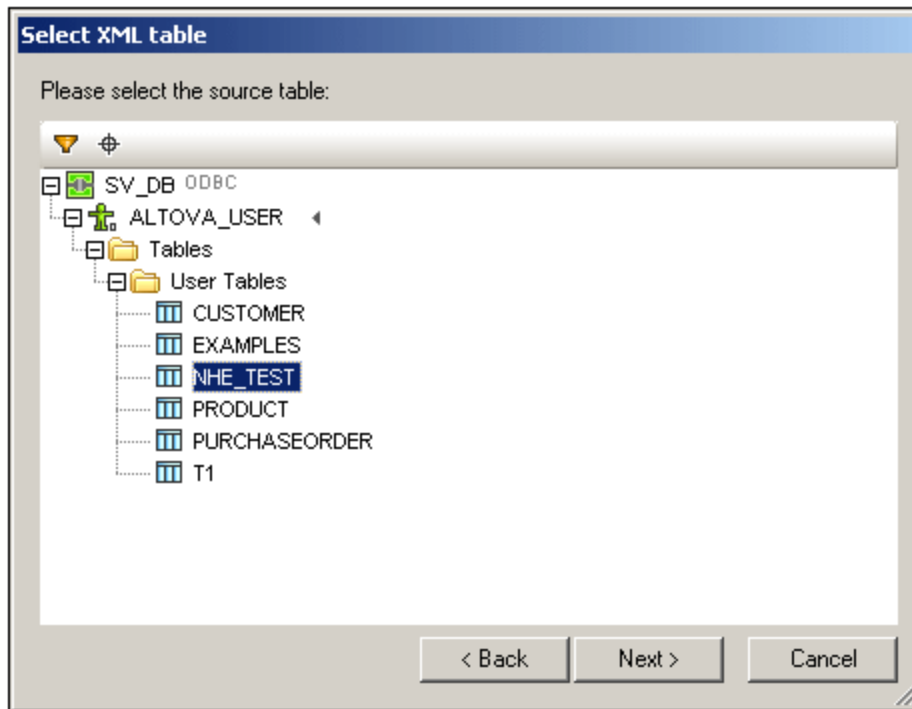
After having made the connection to the XML database (currently only IBM DB2 XML databases are supported) via the Open Database dialog (via the **File | New | New from XML Column in DB Table** command; see *previous sections*), you will need to do two things:

- 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.

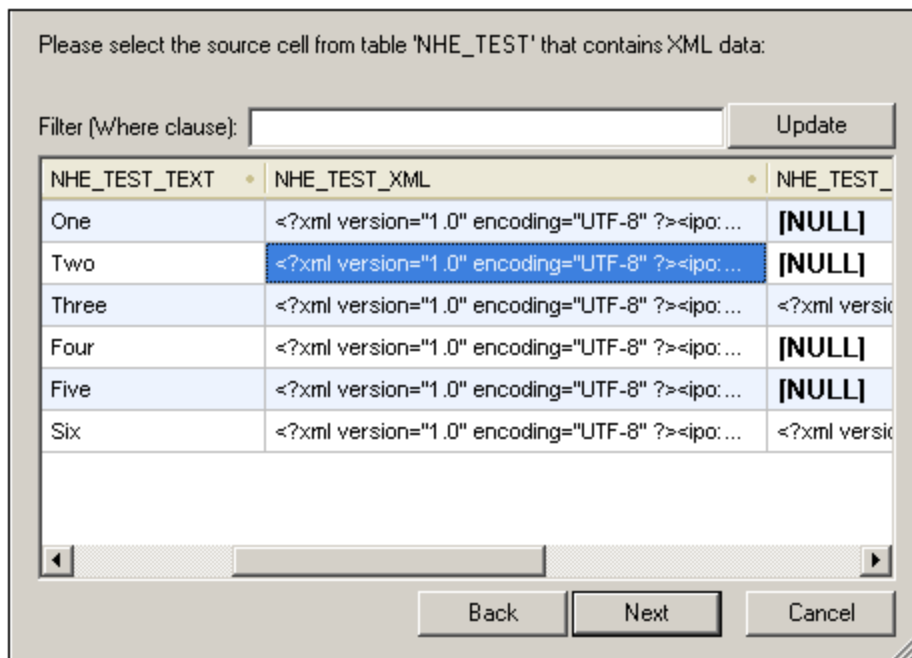
Selecting the XML Cell and Working XML File

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.



- 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'`)



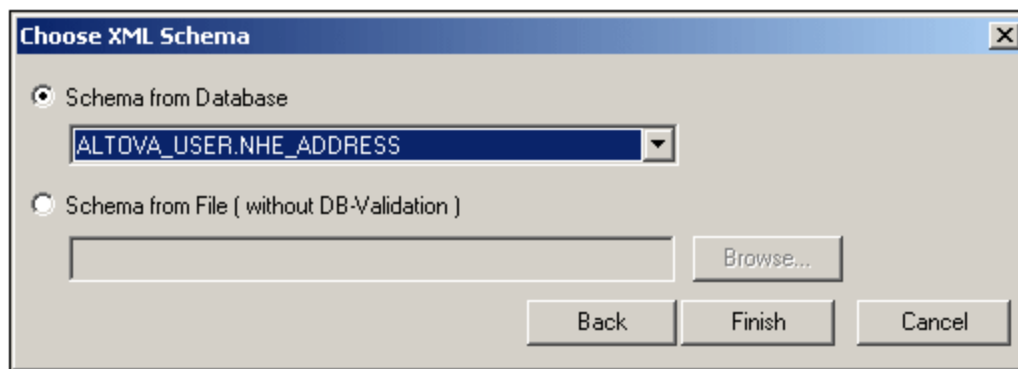
- Select the cell containing the XML data you wish to create as the Working XML File. In the screenshot above, the selected cell is highlighted in blue.
- Click **Next**. This pops up the Choose XML Schema dialog, in which you select the schema to be used for the SPS. See next section.

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:

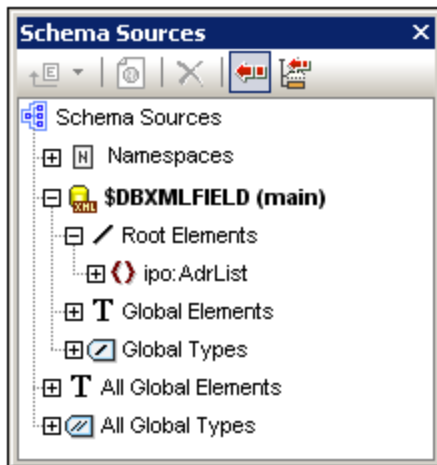
1. In the Choose XML Schema dialog (*screenshot below*), select the appropriate radio button according to whether you wish to select the schema from among those stored in the DB or from a file location. Note that if a non-DB schema is selected—that is, a schema from an external file—then no DB validation 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.

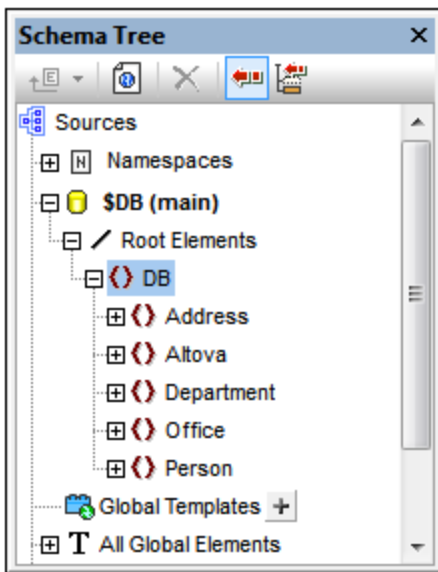
Note: The Working XML File should be valid against the schema selected for the SPS. Also ensure that the schema's [root element \(document element\)](#)²⁰² corresponds to the root element of the XML document.

10.4 The DB Schema and DB XML files

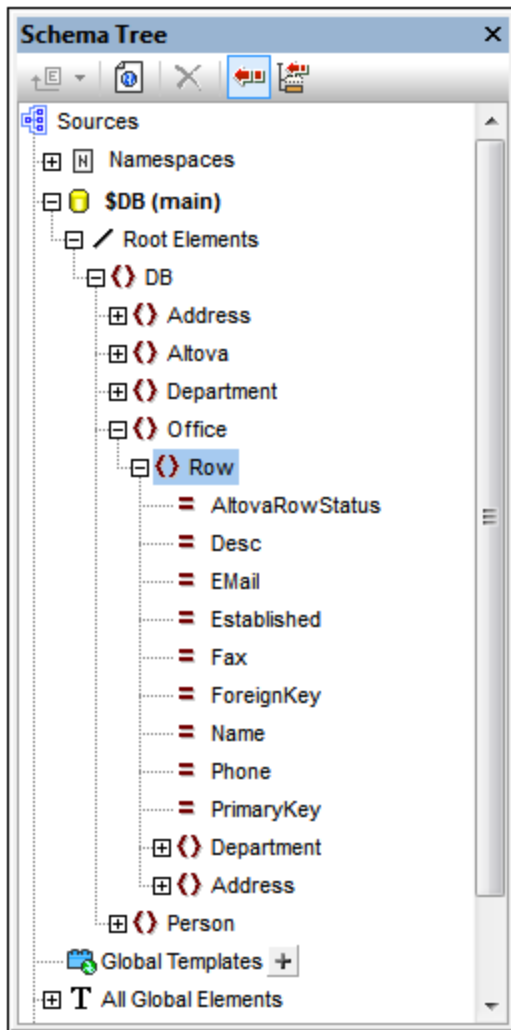
The DB XML Schema

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.



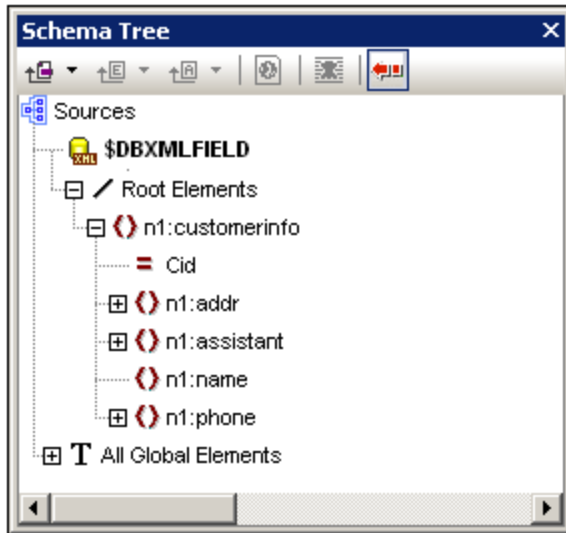
Note: The structure of the generated XML Schema is as outlined above. Whatever tables are selected during the connection step are included in the structure. The construction of a DB Filter does not affect the structure of the XML Schema.

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.



Two **temporary** XML files are generated from the DB (see [DBs and StyleVision](#)⁴⁹⁰ 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 and RTF 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](#)²⁴ 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](#)⁸¹⁰ for details.)
- You do not have to specifically assign a Working XML File in order to see HTML, Text, and RTF previews. The automatically generated (non-editable) temporary XML file is used for this purpose.

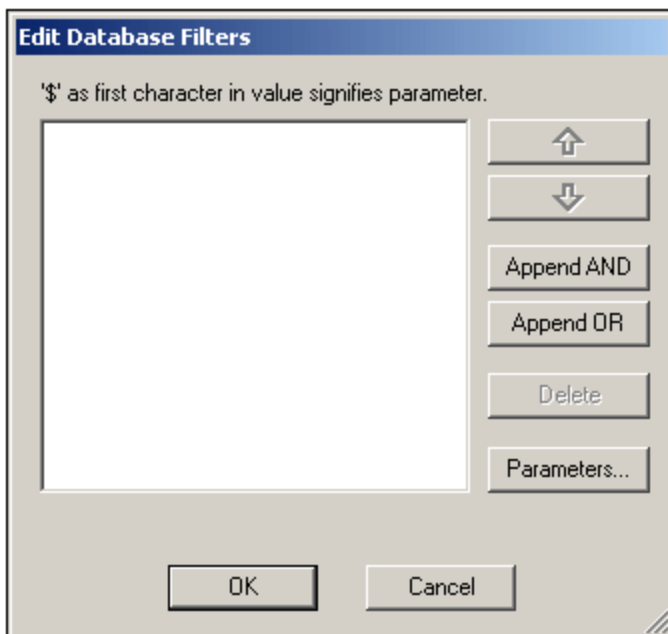
10.5 DB Filters: Filtering DB Data

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 HTMLText, and RTF 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.

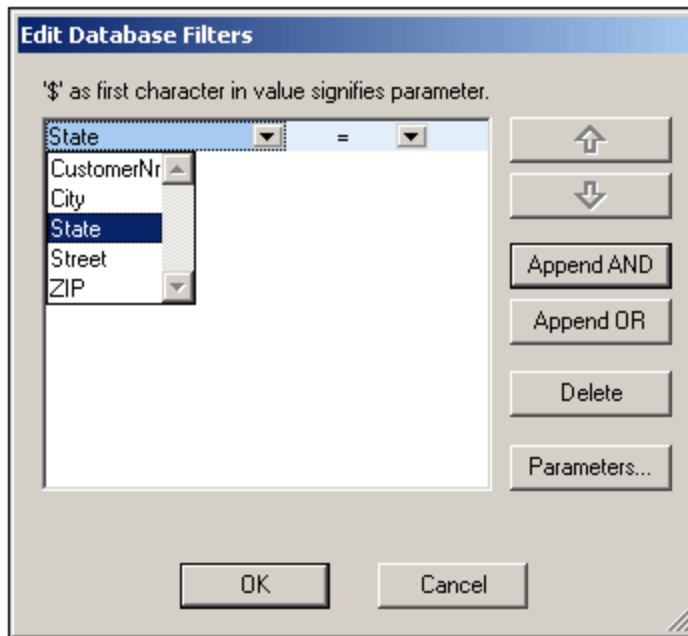
Note: Using a DB Filter modifies the data that is imported into the temporary non-editable XML File. If you save an SPS with a DB Filter and then generate an XML File from the SPS, the generated XML File will be filtered according to the criteria in the DB Filter.

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](#) ⁸¹⁰ or click the  icon in the toolbar. The following dialog is displayed:



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](#)⁵⁹⁰ 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](#)⁵⁹².

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.

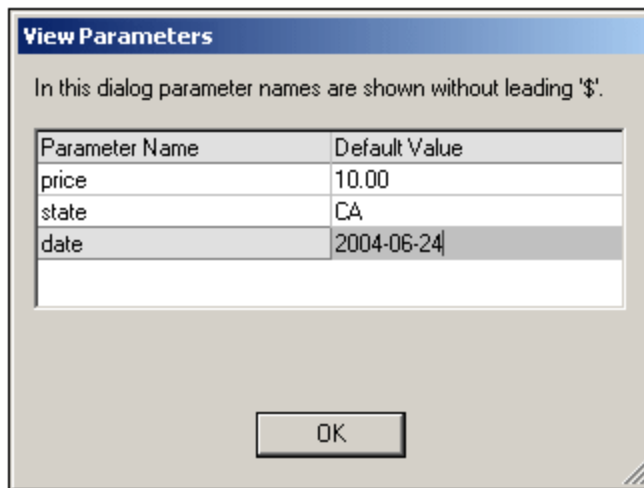
Parameters are useful if you wish to use a single value in multiple expressions, or if you wish to pass a value to a parameter from the command line (see [StyleVision Server](#)⁶⁶³ for details).

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](#)³⁰²), 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.



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 | Stylesheet Parameters**.

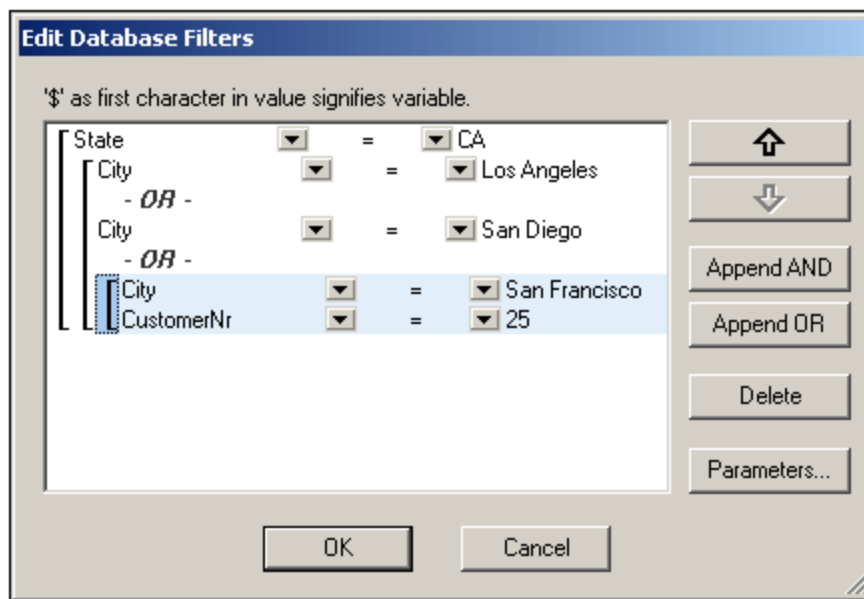
Note: The Edit Parameters dialog contains **all** the parameters that have been defined for the stylesheet. While it is an error to use an undeclared parameter in the SPS, it is not an error to declare a parameter and not use it.

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](#)⁸¹⁰. 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 reloaded 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](#)⁸¹⁰. The filter will be cleared, and the filter symbol will no longer appear alongside the name of the element in the Schema Tree.

10.6 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.

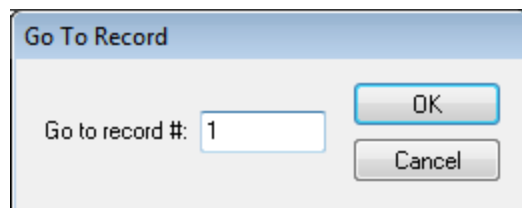
Note: You can also create a DB table in any other format, such as `(contents)`.

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

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:

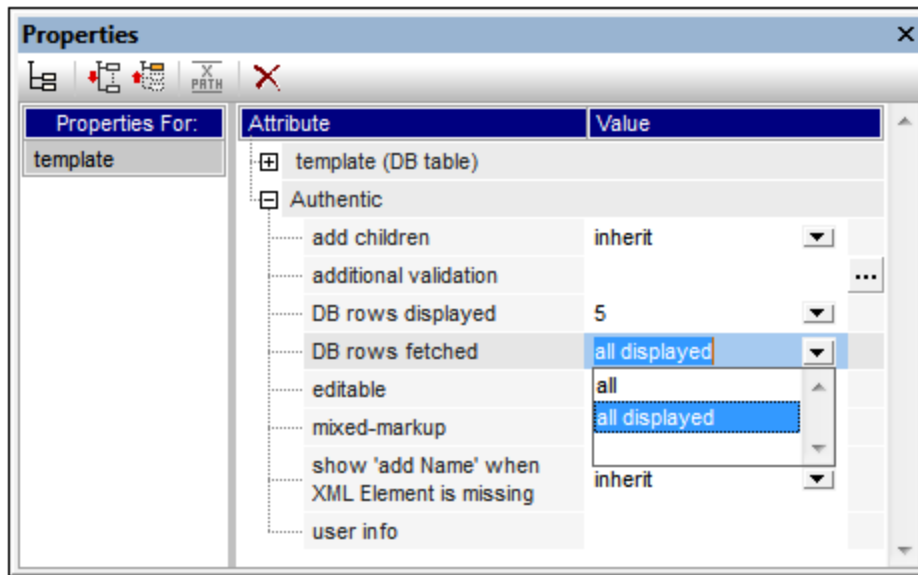
1. Place the cursor at the allowed location (*see above*) where you wish the Query button to appear.
2. Right-click, and from the context menu (or **Insert** menu), select **Insert | DB Control | Query Button**. The Query Button is inserted at that point in the Design document.

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](#) ⁶³⁹.

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 Rows Displayed*, select `all` from the dropdown list or enter the number of rows to be displayed.
4. For the property *DB Rows Fetched*, select `all` or `all displayed` from the dropdown list, or enter the number of rows to be fetched. The value `all displayed` is the default. This property determines how many rows are fetched when the DB data is loaded. If the value is less than that of the *DB Rows Displayed* property, then the value of *DB Rows Displayed* is used as the value of *DB Rows Fetched*. The *DB Rows Fetched* property enables you to reduce the number of rows initially loaded, thus speeding up the loading and display time. Additional rows (records) are loaded and displayed when the row navigation buttons in Authentic View are used.

Note:

- The number of rows 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 rows displayed (because a row 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.

A	The row has been added (but not yet saved to the DB).
U, u	The row has been updated (but not yet saved to the DB).
D, d, X	The row has been deleted (but not yet saved to the DB).

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:

Added	A	Bold, underlined
Modified (Updated)	U, u	Underlined
Deleted	D, d, X	Strikethrough
Datatype error		Red text

10.7 Generating Output Files

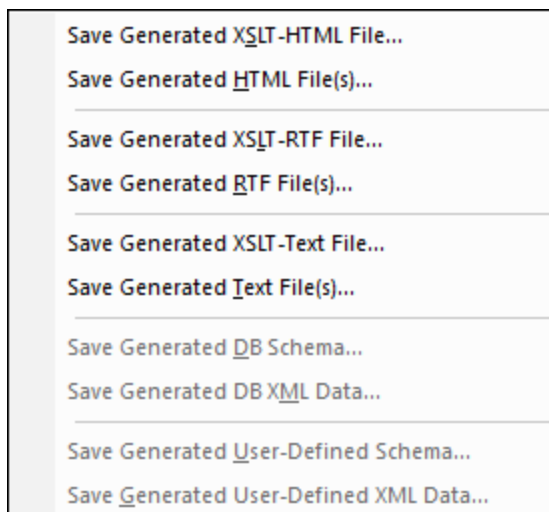
After you have created a DB-based SPS, you can generate files related to it and save them. The following files can be generated and saved:

- 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 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.




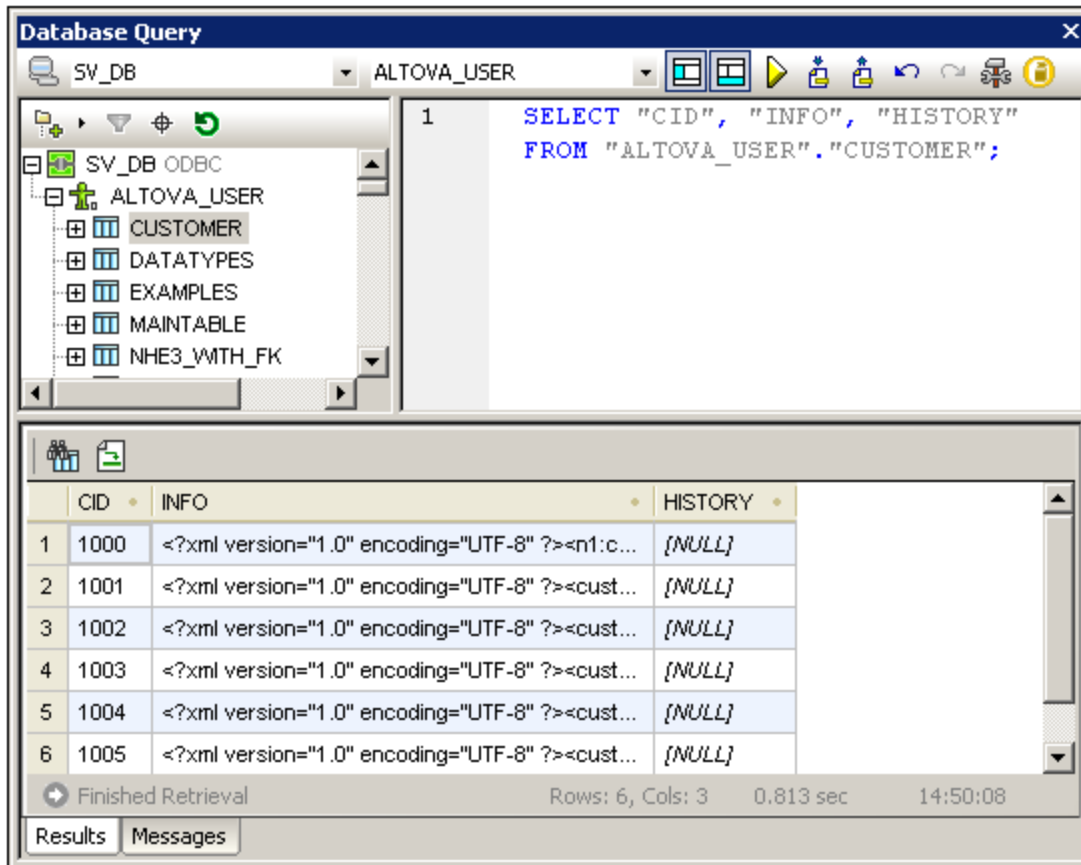
2. Select the file you wish to generate. This pops up the Save As dialog.
3. Browse for the desired folder, enter the desired filename, and click **OK**.

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 and RTF output files based on the design in the SPS.

10.8 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** ⁶⁰³ at top left, which displays connection info and database tables.
- A **Query pane** ⁶⁰⁷ at top right, in which the query is entered.
- A tabbed **Results/Messages pane** ⁶¹⁰. 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](#) ⁶⁰⁷.



Toggles the Browser pane on and off.



Toggles the Results/Messages pane on and off.

Overview of the Query Database mechanism

The Query Database mechanism is as follows. (It is described in detail in the sub-sections of this section.)


1. A [connection to the database is established](#)⁶⁰⁰ via the *Database Query | Connect to a Data Source* window.
2. The connected database or parts of it are displayed in the [Browser pane](#)⁶⁰³, which can be configured to suit viewing requirements.
3. A [query](#)⁶⁰⁹ written in a syntax appropriate to the database to be queried is entered in the [Query pane](#)⁶⁰⁹, and the query is executed.
4. The [results of the query](#)⁶¹⁰ can be viewed through various filters.

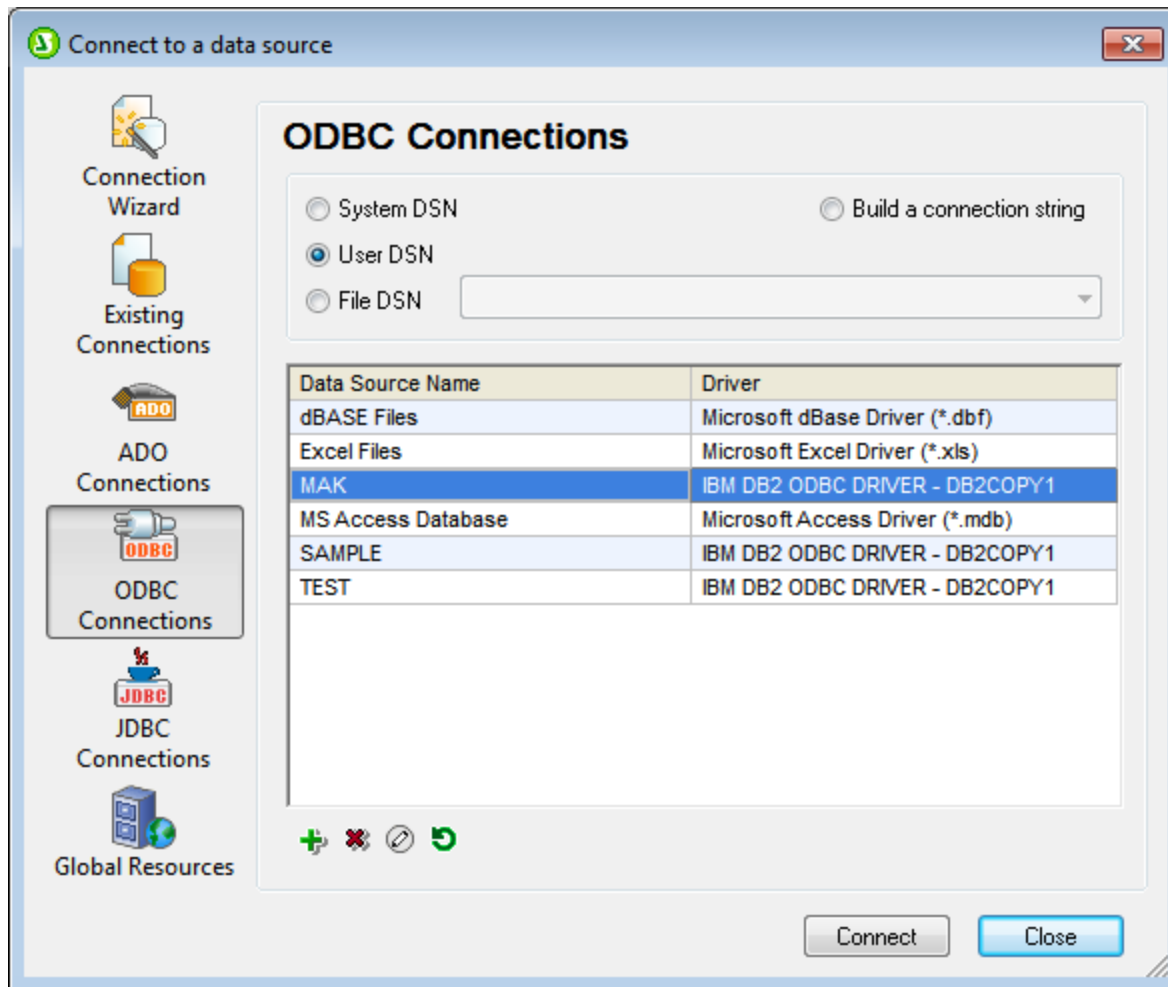
10.8.1 Data Sources

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](#)⁶⁰² 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](#)⁴⁹².

Supported databases

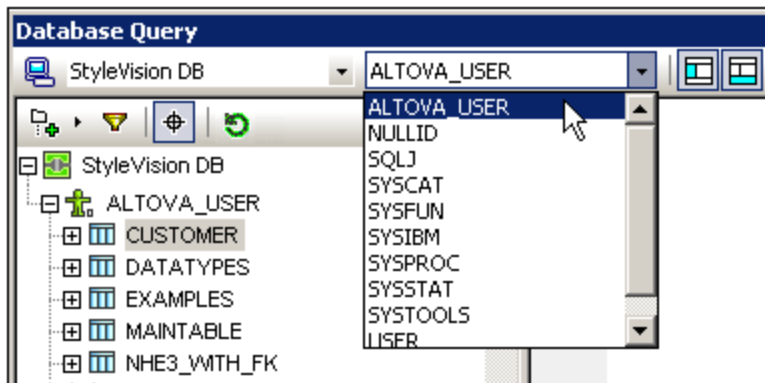
The table below lists all the supported databases. If your Altova application is a 64-bit version, ensure that you have access to the 64-bit database drivers needed for the specific database you are connecting to.

Database	Notes
Firebird 2.x, 3.x, 4.x	
IBM DB2 8.x, 9.x, 10.x, 11.x	
IBM Db2 for i 6.x, 7.4, 7.5	Logical files are supported and shown as views.
IBM Informix 11.70 and later	
MariaDB 10 and later	MariaDB supports native connections. No separate drivers are required.

Database	Notes
Microsoft Access 2003 and later	At the time of writing (early September 2019), there is no Microsoft Access Runtime available for Access 2019. You can connect to an Access 2019 database from Altova products only if Microsoft Access 2016 Runtime is installed and only if the database does not use the "Large Number" data type.
Microsoft Azure SQL Database	SQL Server 2016 codebase
Microsoft SQL Server 2005 and later Microsoft SQL Server on Linux	
MySQL 5 and later	MySQL 5.7 and later supports native connections. No separate drivers are required.
Oracle 9i and later	
PostgreSQL 8 and later	PostgreSQL connections are supported both as native connections and driver-based connections through interfaces (drivers) such as ODBC or JDBC. Native connections do not require any drivers.
Progress OpenEdge 11.6	
SQLite 3.x	SQLite connections are supported as native, direct connections to the SQLite database file. No separate drivers are required. In Authentic view, data coming from a SQLite database is not editable. When you attempt to save SQLite data from the Authentic view, a message box will inform you of this known limitation.
Sybase ASE 15, 16	
Teradata 16	

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](#) ⁶⁰³.

10.8.2 Browser Pane: Viewing the DB Objects

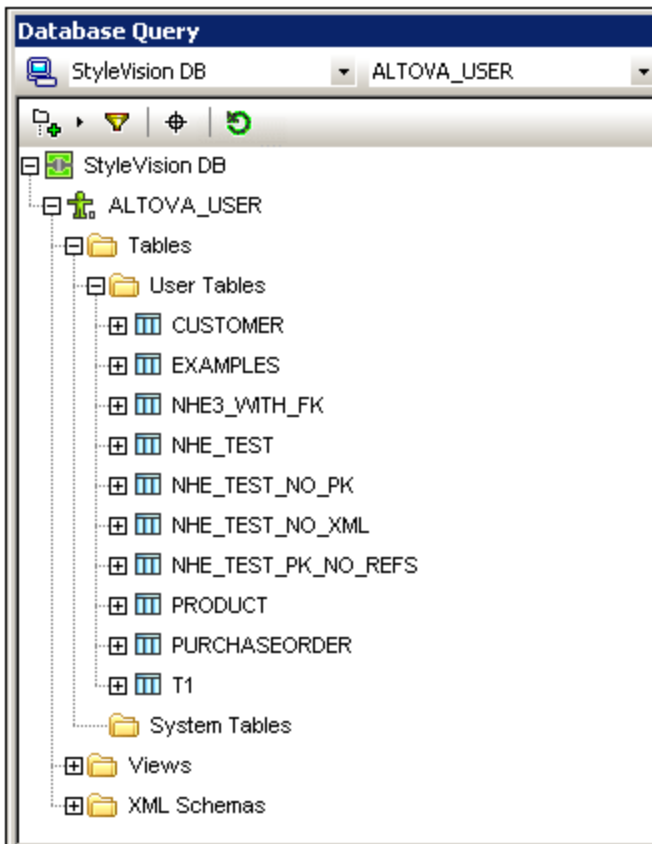
The Browser pane provides an overview of objects in the selected database. This overview includes database constraint information, such as whether a column is a primary or foreign key. In IBM DB2 version 9 databases, the Browser additionally shows registered XML schemas in a separate folder (see *screenshot below*).

This section describes the following:

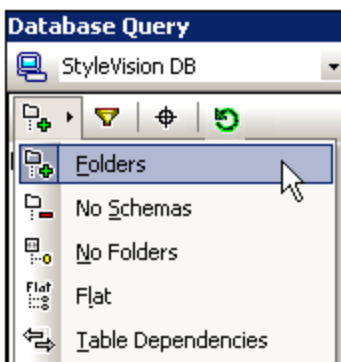
- The [layouts](#) ⁶⁰³ available in the Browser pane.
- [How to filter](#) ⁶⁰⁵ database objects.
- [How to find](#) ⁶⁰⁶ database objects.
- [How to refresh](#) ⁶⁰⁷ the root object of the active data source.

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.

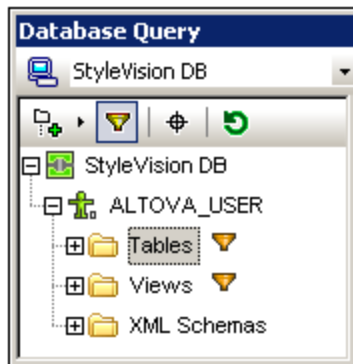
To sort tables into User and System tables, switch to Folders, No Schemas or Flat layout, then right-click the Tables folder and select **Sort into User and System Tables**. The tables are sorted alphabetically in the User Tables and System Tables folders.

Filtering database objects

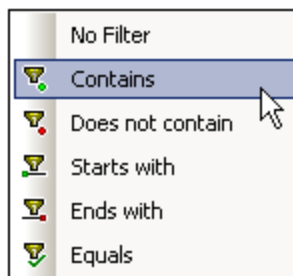
In the Browser pane (in all layouts except No Folders and Table Dependencies), schemas, tables, and views can be filtered by name or part of a name. Objects are filtered as you type in the characters, and filtering is case-insensitive by default.

To filter objects in the Browser, do the following:

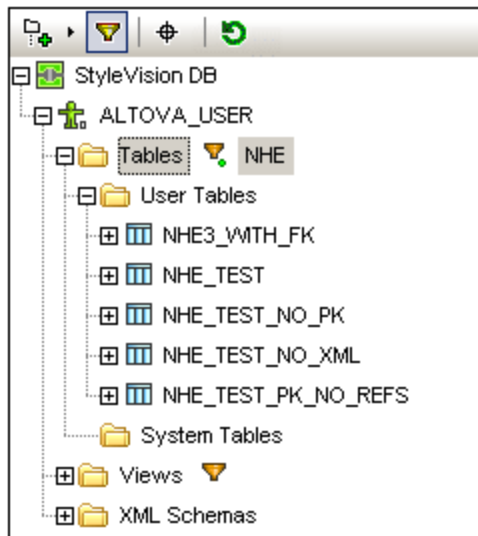
1. Click the Filter Folder Contents icon in the toolbar of the Browser pane. Filter icons appear next to the Tables and Views folders in the currently selected layout (*screenshot below*).



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*.



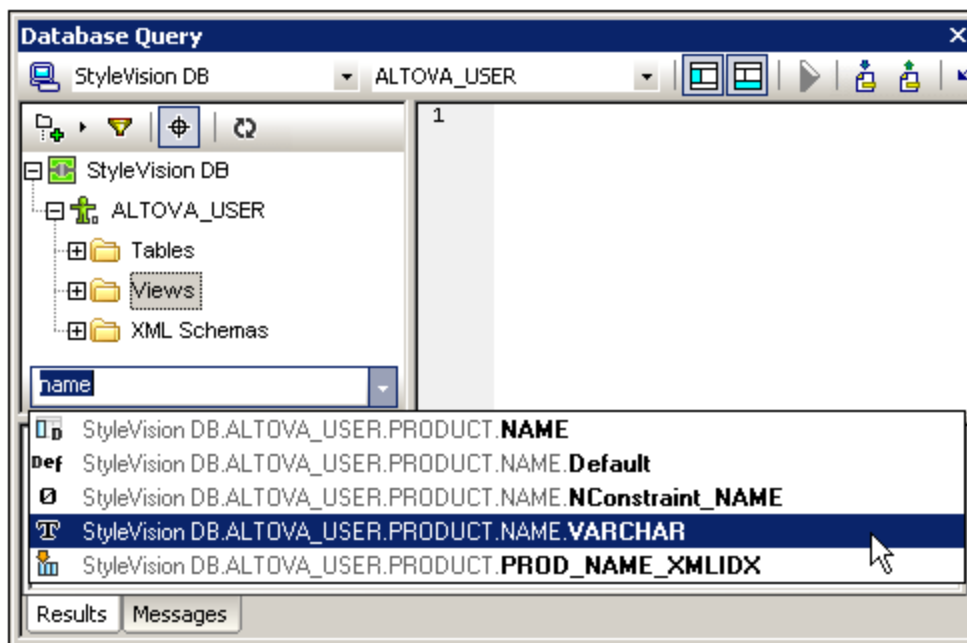
3. In the entry field that appears, enter the filter string (in the screenshot below, the filter string on the *Tables* folder is *NHE*). The filter is applied as you type.



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.








10.8.3 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](#)⁶⁰⁷ of the Database Query window executes the query and displays the result and execution messages in the [Results/Messages pane](#)⁶¹⁰. How to work with queries is described in the next section, [Query Pane: Working with Queries](#)⁶⁰⁹. In this section, we describe the main features of the Query pane:

- [SQL Editor icons](#)⁶⁰⁷ in the Database Query toolbar
- [SQL Editor options](#)⁶⁰⁷
- [Auto-completion of SQL statements](#)⁶⁰⁸
- [Definition of regions in an SQL script](#)⁶⁰⁹
- [Insertion of comments in an SQL script](#)⁶⁰⁹
- [Use of bookmarks](#)⁶⁰⁹

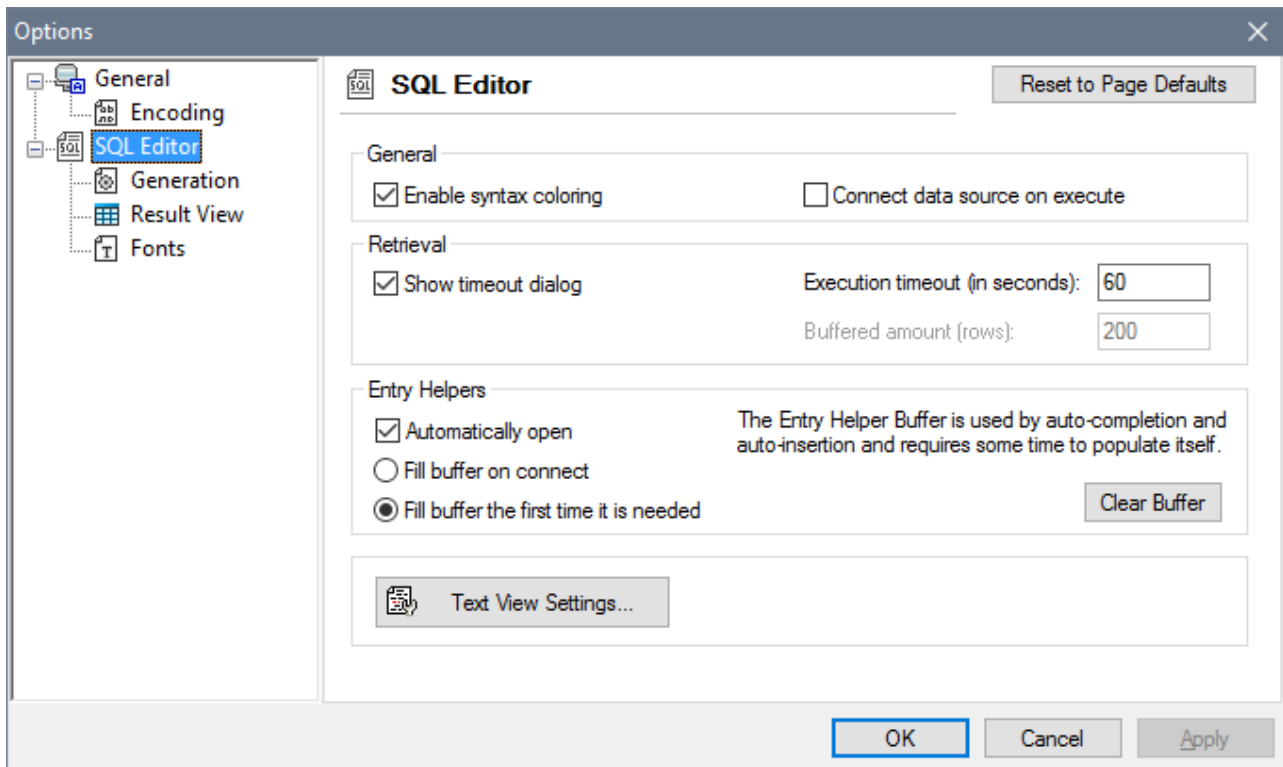
SQL Editor icons in the Database Query toolbar

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. Entry helpers refer to the entry helpers that appear as part of the auto-completion feature. When you type in an SQL statement, the editor displays a list of context-sensitive auto-completion suggestions. These suggestions can be set to appear automatically. If the automatic display is switched off, then you can ask for an auto-completion suggestion in SQL Editor at any point in the SQL query by pressing **Ctrl+Spacebar**. The buffer for the entry helper information can be filled either on connection to the data source or the first time it is needed. The Text View settings button opens a dialog in which you can specify settings such as indentation and tab size of text in the SQL Editor.
- **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 | 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*).

```
1  -- region CUSTOMER table
4
5  -- region EXAMPLES table
6  SELECT "ID", "FILENAME1", "FILENAME2",
7  "FILECONTENT1", "FILECONTENT2" FROM
8  "ALTOVA_USER"."EXAMPLES";
   -- endregion
```

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**.

10.8.4 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](#)⁶¹⁰. Messages about the execution are displayed in the [Messages tab](#)⁶¹¹.

10.8.5 Results and Messages

The Results/Messages pane has two tabs:

- The [Results tab](#)⁶¹⁰ shows the data that is retrieved by the query.
- The [Messages tab](#)⁶¹¹ 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*).

Amount: 200

	CID	INFO	HISTORY
1	1000	<?xml version="1.0" encoding="UTF-8" ?><n1:c...	[NULL]
2	1001	<?xml version="1.0" encoding="UTF-8" ?><cust...	[NULL]
3	1002	<?xml version="1.0" encoding="UTF-8" ?><cust...	[NULL]
4	1003	<?xml version="1.0" encoding="UTF-8" ?><cust...	[NULL]
5	1004	<?xml version="1.0" encoding="UTF-8" ?><cust...	[NULL]
6	1005	<?xml version="1.0" encoding="UTF-8" ?><cust...	[NULL]

Fin Rows: 6, Cols: 3 0.109 sec 10:19:59

Results Messages

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:



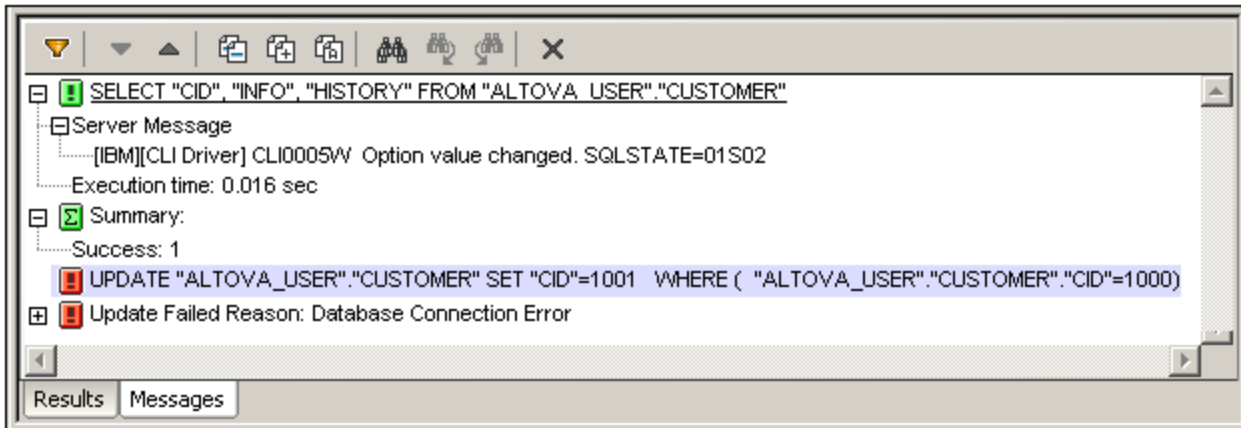
Go to Statement Highlights the statement in the SQL Editor that produced the current result.



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 lets 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.

11 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.

Nanonull, Inc.	
Location: <input type="text" value="US"/>	
Street:	119 Oakstreet, Suite 4876
City:	Vereno
State & Zip:	<input type="text" value="DC"/> <input type="text" value="29213"/>
Phone:	+1 (321) 555 5155 0
Fax:	+1 (321) 555 5155 4
E-mail:	office@nanonull.com
<u>Vereno Office Summary: 4 departments, 15 employees.</u>	
<p>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 <i>NanoSoft Development Suite</i> 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.</p>	

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](#)⁶¹⁴ and [how to edit documents in Authentic View](#)⁶²⁶.

11.1 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.

Altova website:  [XML content editing](#), [XML authoring](#)

11.1.1 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](#) ⁶¹⁵.

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](#) ⁶¹⁷.

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

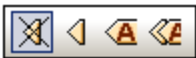
These are the menus that appear when you right-click in the Main Window. The available commands are context-sensitive editing commands, i.e. they allow you to manipulate structure and content relevant to the selected node. Such manipulations include inserting, appending, or deleting a node, adding entities, or cutting and pasting content.

11.1.2 Authentic View Toolbar Icons

Icons in the Authentic View toolbar are command shortcuts. Some icons will already be familiar to you from other Windows applications or Altova products, others might be new to you. This section describes icons unique to Authentic View. In the description below, related icons are grouped together.

Show/hide XML markup

In Authentic View, the tags for all, some, or none of the XML elements or attributes can be displayed, either with their names (large markup) or without names (small markup). The four markup icons appear in the toolbar, and the corresponding commands are available in the **Authentic** menu.



	Hide markup. All XML tags are hidden except those which have been collapsed. Double-clicking on a collapsed tag (which is the usual way to expand it) in Hide markup mode will cause the node's content to be displayed and the tags to be hidden.
	Show small markup. XML element/attribute tags are shown without names.
	Show large markup. XML element/attribute tags are shown with names.
	Show mixed markup. In the StyleVision Power Stylesheet, each XML element or attribute can be specified to display (as either large or small markup), or not to display at all. This is called mixed markup mode since some elements can be specified to be displayed with markup and some without markup. In mixed markup mode, therefore, the Authentic View user sees a customized markup. Note, however, that this customization is created by the person who has designed the StyleVision Power Stylesheet. It cannot be defined by the Authentic View user.







Editing dynamic table structures

Rows in a **dynamic SPS table** are repetitions of a data structure. Each row represents an occurrence of a single element. Each row, therefore, has the same XML substructure as the next.

The dynamic table editing commands manipulate the rows of a dynamic SPS table. That is, you can modify the number and order of the element occurrences. You cannot, however, edit the columns of a dynamic SPS table, since this would entail changing the substructure of individual element occurrences.

The icons for dynamic table editing commands appear in the toolbar, and are also available in the **Authentic** menu.




	Append row to table
	Insert row in table
	Duplicate current table row (i.e. cell contents are duplicated)
	Move current row up by one row
	Move current row down by one row
	Delete the current row

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; Go to Previous Record; Open the *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.

11.1.3 Authentic View Main Window

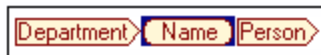
There are four viewing modes in Authentic View: Large Markup; Small Markup; Mixed Markup; and Hide All Markup. These modes enable you to view the document with varying levels of markup information. To switch between modes, use the commands in the **Authentic** menu or the icons in the toolbar (see the previous section, [Authentic View toolbar icons](#)⁶¹⁵).

Large markup

This shows the start and end tags of elements and attributes with the element/attribute names in the tags:



The element `Name` in the figure above is **expanded**, i.e. the start and end tags, as well as the content of the element, are shown. An element/attribute can be **contracted** by double-clicking either its start or end tag. To expand the contracted element/attribute, double-click the contracted tag.



In large markup, attributes are recognized by the equals-to symbol in the start and end tags of the attribute:



Small markup

This shows the start and end tags of elements/attributes without names:

ⓧ Nanonull, Inc. ⓧ

Location: ⓧ US ⓧ

<p>ⓧ</p> <p>Street: ⓧ 119 Oakstreet, Suite 4876 ⓧ</p> <p>City: ⓧ Vereno ⓧ</p> <p>State & Zip: ⓧ DC ⓧ <input style="width: 150px;" type="text" value="29213"/></p>	<p>Phone: ⓧ +1 (321) 555 5155 0 ⓧ</p> <p>Fax: ⓧ +1 (321) 555 5155 4 ⓧ</p> <p>E-mail: ⓧ office@nanonull.com ⓧ</p>
---	--

ⓧ

ⓧ ⓧ [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.

ⓧ ⓧ

Notice that start tags have a symbol inside them while end tags are empty. Also, element tags have an angular-brackets symbol while attribute tags have an equals sign as their symbol (see screenshot below).

ⓧ ⓧ 2006-04-01 ⓧ ⓧ Boston ⓧ, ⓧ USA ⓧ ⓧ ⓧ

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.

ⓧ ⓧ ⓧ **Office Summary: 4 departments, 15 employees.** ⓧ ⓧ

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.



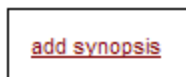
- Data-entry devices. The display contains either an input field (text box), a multiline input field, combo box, check box, or radio button. In the case of input fields and multiline input fields, the text you enter in the field becomes the XML content of the element or the value of the attribute.



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

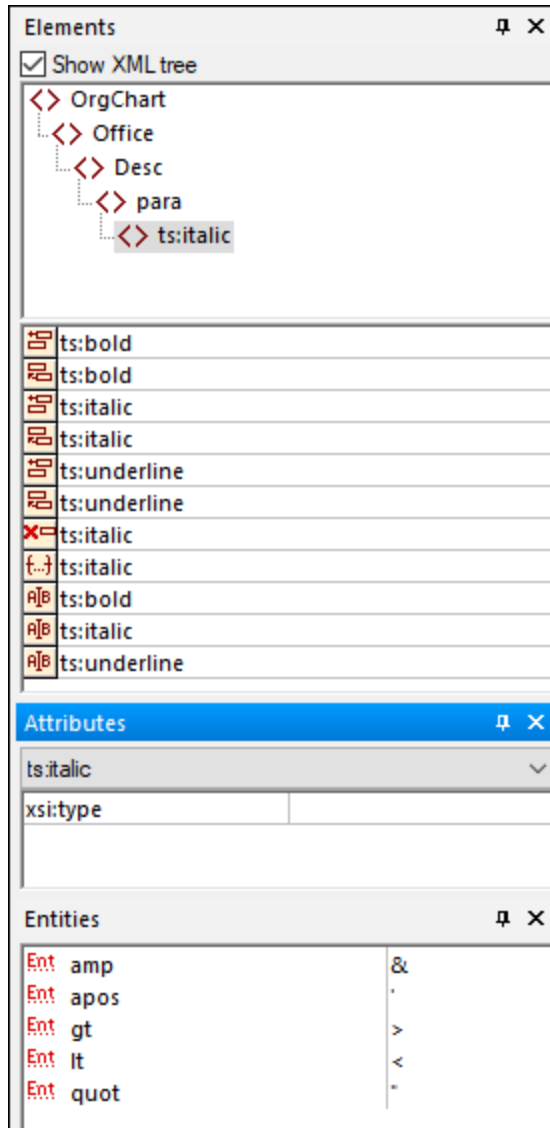
When an element or attribute is **optional** (according to the referenced schema), a prompt of type `add [element/attribute]` is displayed:



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.

11.1.4 Authentic View Entry Helpers

There are three entry helpers in Authentic View: for Elements, Attributes, and Entities. They are displayed as windows down the right side of the Authentic View interface (see *screenshot below*).



The Elements and Attributes Entry Helpers are context-sensitive, i.e. what appears in the entry helper depends on where the cursor is in the document. The entities displayed in the Entities Entry Helper are not context-sensitive; all entities allowed for the document are displayed no matter where the cursor is.

Each of the entry helpers is described separately below.

Elements Entry Helper

The Elements Entry Helper consists of two parts:

- The upper part, containing an XML tree that can be toggled on and off using the **Show XML tree** check box. The XML tree shows the ancestors up to the document's root element for the current element. When you click on an element in the XML tree, elements corresponding to that element (as described in the next item in this list) appear in the lower part of the Elements Entry Helper.

- The lower part, containing a list of the nodes that can be inserted within, before, and after; removed; applied to or cleared from the selected element or text range in Authentic View. What you can do with an element listed in the Entry Helper is indicated by the icon to the left of the element name in the Entry Helper. The icons that occur in the Elements Entry Helper are listed below, together with an explanation of what they mean.

To use a 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

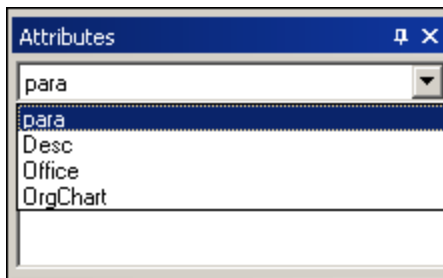
This icon appears when text within an element of mixed content is selected. Clicking the icon clears the element from around the selected text range.

Clear Element (when insertion point selected)

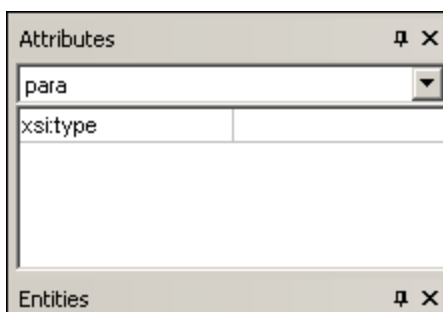
This icon appears when the cursor is placed within an element that is a child of a mixed-content element. Clicking the icon clears the inline element.

Attributes Entry Helper

The Attributes Entry Helper consists of a drop-down combo box and a list of attributes. The element that you have selected (you can click the start or end tag, or place the cursor anywhere in the element content to select it) appears in the combo box. The Attributes Entry Helper shown in the figures below has a `para` element in the combo box. Clicking the arrow in the combo box drops down a list of all the `para` element's **ancestors up to the document's root element**, which in this case is `OrgChart`.



Below the combo box, a list of valid attributes for that element is displayed, in this case for `para`. If an attribute is mandatory on a given element, then it appears in bold. (In the example below, there are no mandatory attributes except the built-in attribute `xsi:type`.)



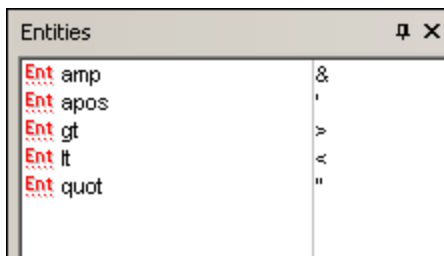
To enter a value for an attribute, click in the value field of the attribute and enter the value. This creates the attribute and its value in the XML document.

Note the following:

- 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 either (i) selecting a value from the dropdown list that appears, or (ii) entering a value. Values displayed in the dropdown list 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](#)⁶⁴⁶ in the Editing in Authentic View section.

11.1.5 Authentic View Context Menus

Right-clicking on some selected document content or node pops up a context menu with commands relevant to the selection or cursor location.

Inserting elements

The figure below shows the **Insert** submenu, which is a list of all elements that can be inserted at that current cursor location. The **Insert Before** submenu lists all elements that can be inserted before the current element. The **Insert After** submenu lists all elements that can be inserted after the current element. In the figure below, the current element is the `para` element. The `bold` and `italic` elements can be inserted within the current `para` element.



As can be seen below, the `para` and `Office` elements can be inserted before the current `para` element.



The node insertion, replacement (**Apply**), and markup removal (**Clear**) commands that are available in the context menu are also available in the [Authentic View entry helpers](#)⁶¹⁹ 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](#)⁶⁴⁶ 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](#)⁶¹⁹.

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).

Note: 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 node or by clicking either the start or end tag of the node.

11.2 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](#)⁶⁴⁵.
- 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](#)⁶⁴⁶ for details.
- What [image formats](#)⁶⁴⁸ 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.

Altova website: [XML content editing](#), [XML authoring](#)

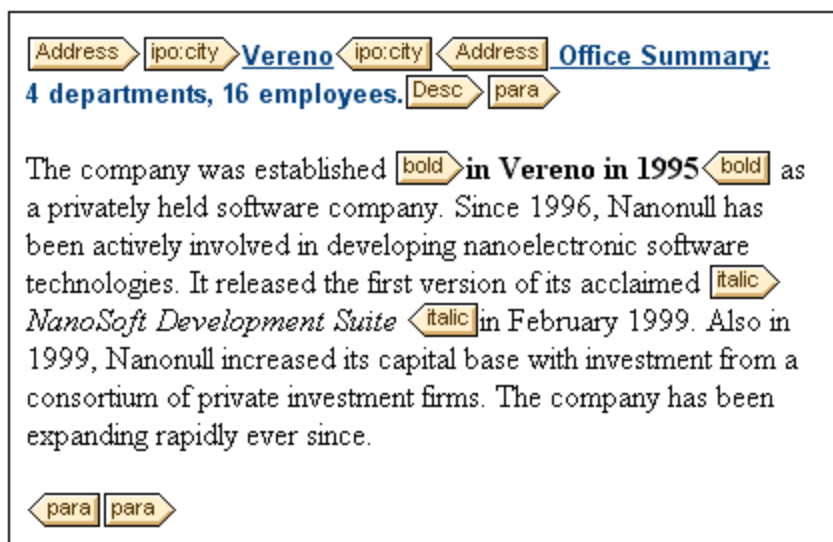
11.2.1 Basic Editing

When you edit in Authentic View, you are editing an XML document. Authentic View, however, can hide the structural XML markup of the document, thus displaying only the content of the document (*first screenshot below*). You are therefore not exposed to the technicalities of XML, and can edit the document as you would a normal text document. If you wish, you could switch on the markup at any time while editing (*second screenshot below*).

Vereno Office Summary: 4 departments, 16 employees.

The company was established **in Vereno in 1995** as a privately held software company. Since 1996, Nanonull has been actively involved in developing nanoelectronic software technologies. It released the first version of its acclaimed *NanoSoft Development Suite* in February 1999. Also in 1999, Nanonull increased its capital base with investment from a consortium of private investment firms. The company has been expanding rapidly ever since.

An editable Authentic View document with no XML markup.

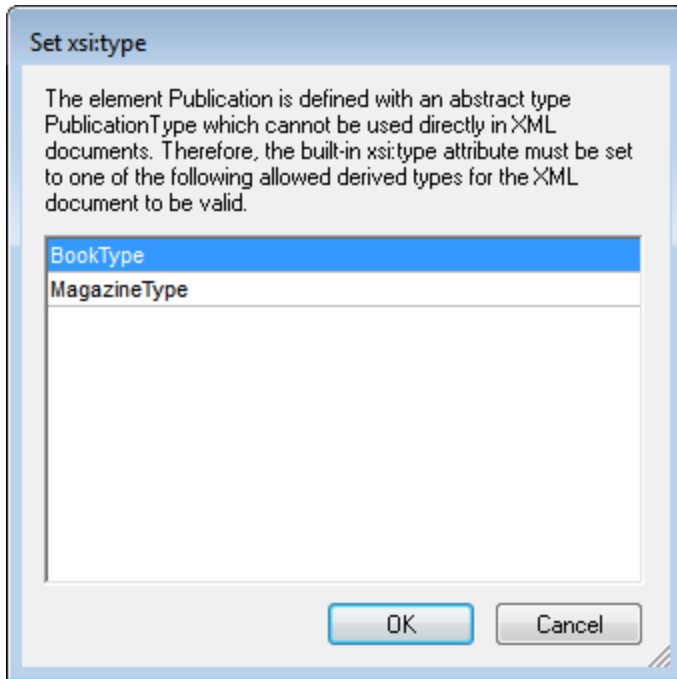


An editable Authentic View document with XML markup tags.

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](#)⁶²³.

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](#)⁶²⁰ section of the documentation.

Using RichEdit in Authentic View

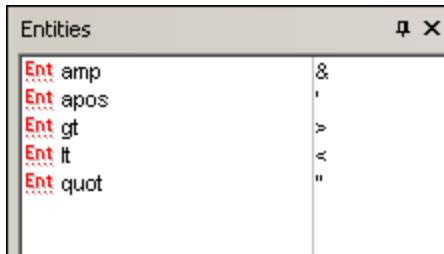
In Authentic View, when the cursor is placed inside an element that has been created as a RichEdit component, the buttons and controls in the RichEdit toolbar (*screenshot below*) become enabled. Otherwise they are grayed out.



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](#)⁶²³ (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](#)⁶⁴⁶ within the Authentic View application. Once defined, these entities appear in the [Entities Entry Helper](#)⁶²³ 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 (`>`). 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](#)⁶¹⁵. 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.)

11.2.2 Tables in Authentic View

The three table types fall into two categories: SPS tables (static and dynamic) and CALS/HTML Tables.

SPS tables are of two types: static and dynamic. SPS tables are designed by the designer of the StyleVision Power Stylesheet to which your XML document is linked. You yourself cannot insert an SPS table into the XML document, but you can enter data into SPS table fields and add and delete the rows of dynamic SPS tables. The section on [SPS tables](#)⁶³¹ 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](#)⁶³² and the [CALS/HTML Table editing icons](#)⁶³⁶ are described below.

11.2.2.1 SPS Tables

Two types of SPS tables are used in Authentic View: static tables and dynamic tables.

Static 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.

Nanonull, Inc.	
Street:	119 Oakstreet, Suite 4876
City:	Vereno
State & Zip:	DC 29213
Phone:	+1 (321) 555 5155
Fax:	+1 (321) 555 5155 - 9
E-mail:	office@nanonull.com

Note: The icons or commands for editing dynamic tables **must not** be used to edit static tables.

Dynamic tables

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.

Administration								
First	Last	Title	Ext	EMail	Shares	Leave		
						Total	Used	Left
Vernon	Callaby	Office Manager	581	v.callaby@nanonull.com	1500	25	4	21
Frank	Further	Accounts Receivable	471	f.further@nanonull.com	0	22	2	20
Loby	Matise	Accounting Manager	963	l.matise@nanonull.com	add Shares	25	7	18
Employees: 3 (20% of Office, 9% of Company)					Shares: 1500 (13% of Office, 6% of Company)			
Non-Shareholders: Frank Further, Loby Matise.								


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.

11.2.2.2 CALS/HTML Tables

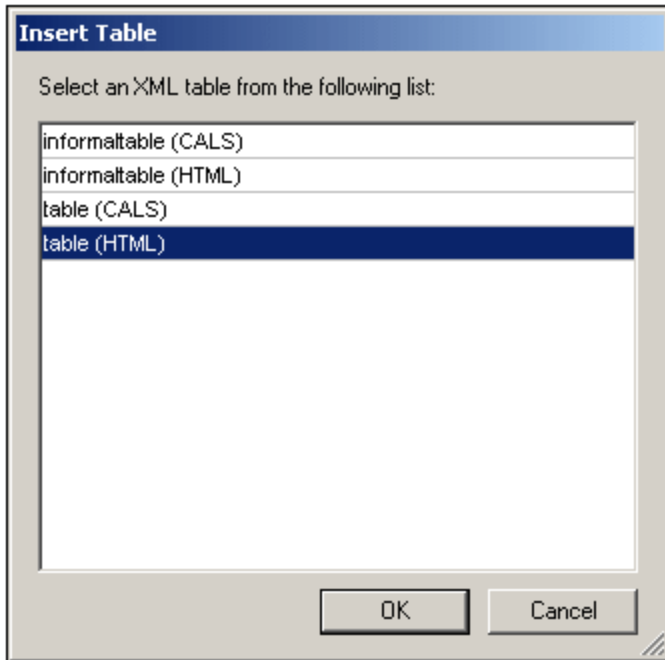
CALS/HTML tables can be inserted by you, the user of Authentic View, for certain XML data structures that have been specified to show a table format. There are three steps involved when working with CALS/HTML tables: inserting the table; formatting it; and entering data. The commands for working with CALS/HTML tables are available as icons in the toolbar (see [CALS/HTML table editing icons](#)⁶³⁶).

Inserting tables

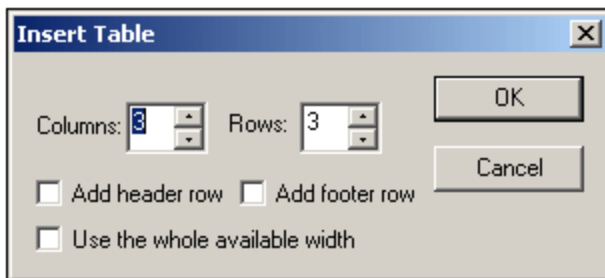
To insert a CALS/HTML table do the following:

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.


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.

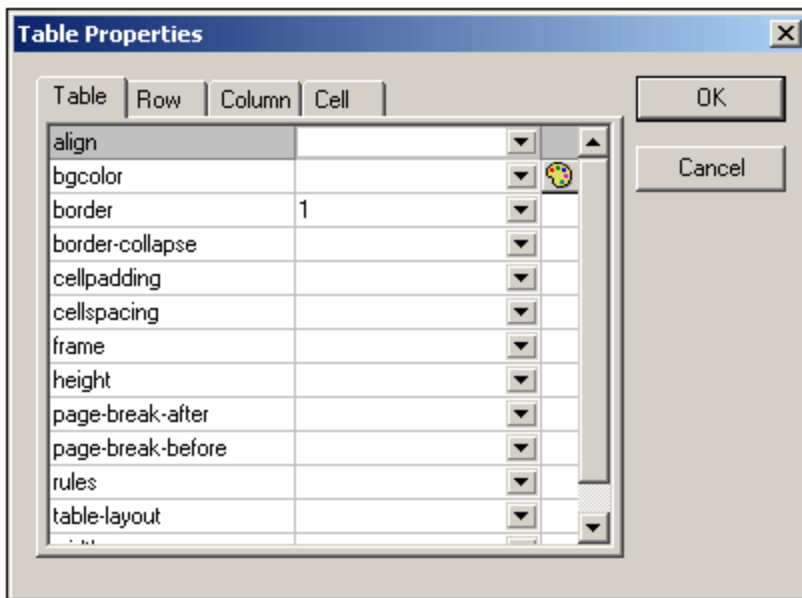
Formatting tables and entering data

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.

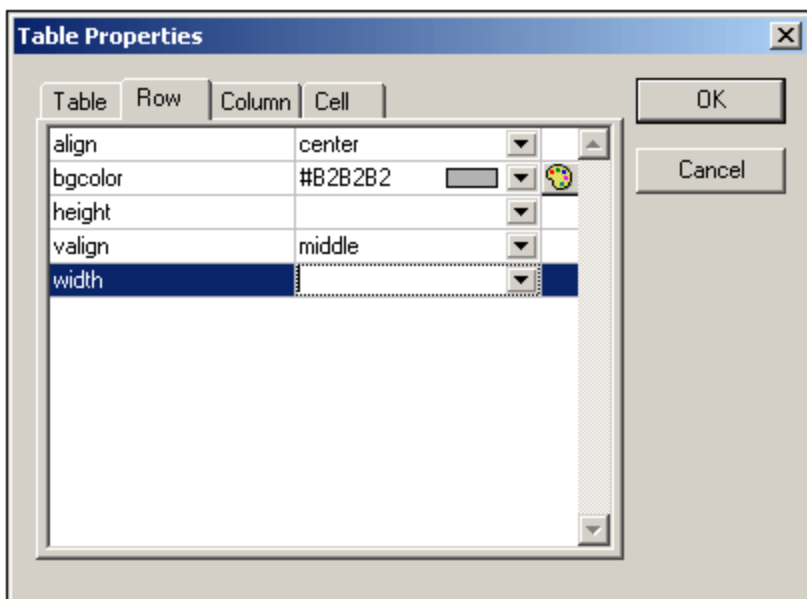
To format a table, row, column, or cell, do the following:

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.



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  (Table Properties) icon. Click the **Row** tab.




Since the first row will be the header row, set a background color to differentiate this row from the other rows. Note the Row properties that have been set in the figure above. Then enter the column header text. Your table will now look like this:

Name	Telephone	Email

Notice that the alignment is centered as specified.

- Now, say you want to divide the "Telephone" column into the sub-columns "Office" and "Home", in which case you would need to split the horizontal width of the Telephone column into two columns. First, however, we will split the vertical extent of the header cell to make a sub-header row. Place the cursor in the "Telephone" cell, and click the  (Split vertically) icon. Your table will look like this:

Name	Telephone		Email

- Now place the cursor in the cell below the cell containing "Telephone", and click the  (Split horizontally) icon. Then type in the column headers "Office" and "Home". Your table will now look like this:

Name	Telephone		Email
	Office	Home	

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](#)⁶³⁶.

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.

Name	Telephone		Email
	Office	Home	

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.

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.

11.2.2.3 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](#)⁶³².

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

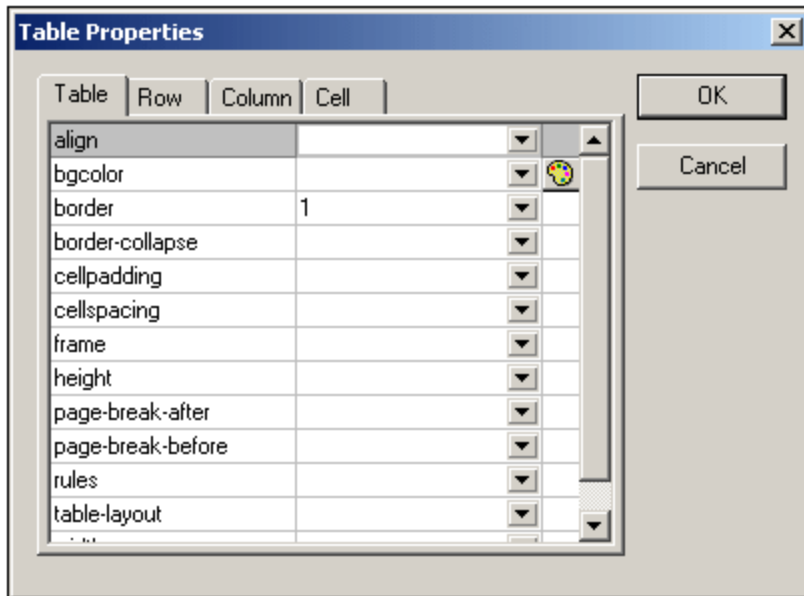


This command aligns the cell contents to the bottom of the cell.

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.



11.2.3 Editing a DB

In Authentic View, you can edit database (DB) tables and save data back to a DB. This section contains a full description of interface features available to you when editing a DB table. The following general points need to be noted:

- The number of records in a DB table that are displayed in Authentic View may have been deliberately restricted by the designer of the StyleVision Power Stylesheet in order to make the design more compact. In such cases, only that limited number of records is initially loaded into Authentic View. Using the DB table row navigation icons (see [Navigating a DB Table](#)⁶³⁹), you can load and display the other records in the DB table.
- You can [query the DB](#)⁶³⁹ 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](#)⁶⁴³.

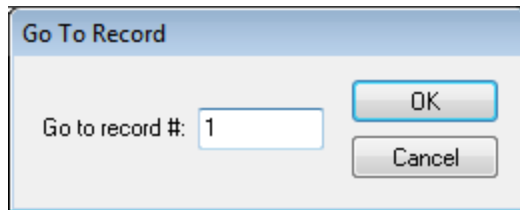
Note: In Authentic View, data coming from a SQLite database is not editable. When you attempt to save SQLite data in Authentic View, a message box will inform you of this known limitation.

11.2.3.1 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](#) ⁸⁰⁵ menu command.

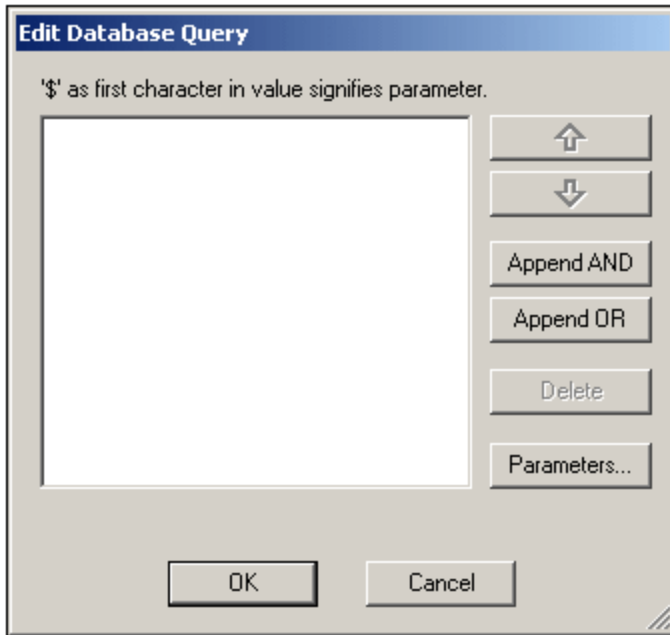
11.2.3.2 DB Queries

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.

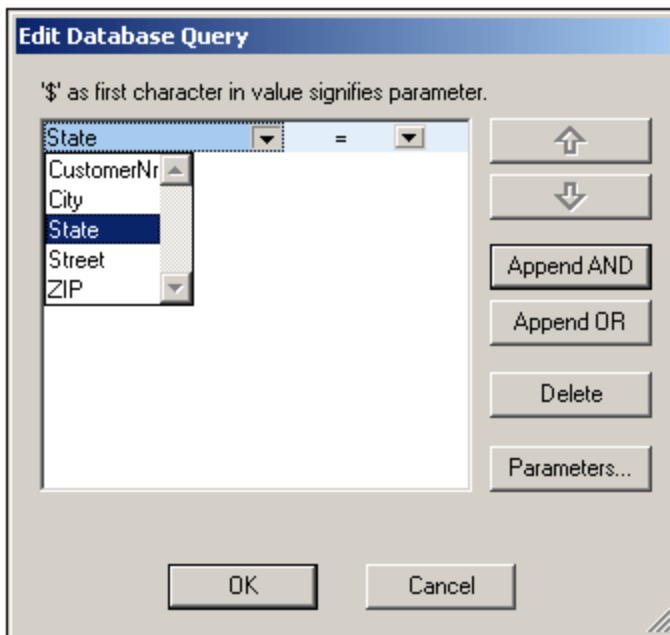
Note: If you get a message saying that too many tables are open, then you can reduce the number of tables that are open by using a query to filter out some tables.

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).



3. 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](#)⁶⁴¹ section.
4. 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](#)⁶⁴².

Expressions in criteria

Expressions in DB Query criteria consist of a field name, an operator, and a value. The **available field names** are the child elements of the selected top-level data table; the names of these fields are listed in a combo-box (see *screenshot above*). The **operators** you can use are listed below:

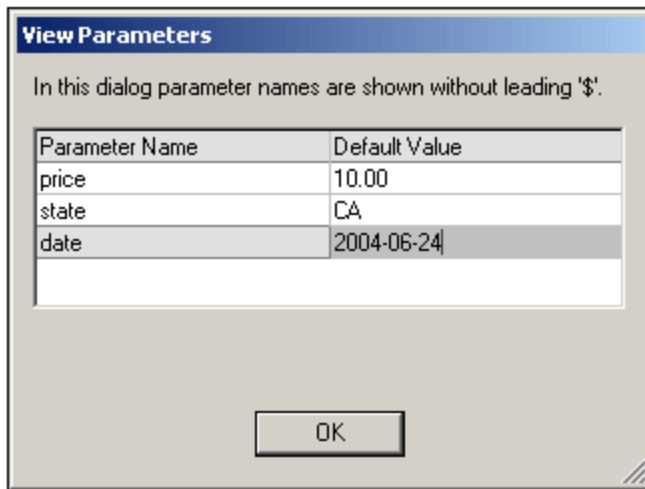
=	Equal to
<>	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
LIKE	Phonetically alike
NOT LIKE	Phonetically not alike
IS NULL	Is empty
NOT NULL	Is not empty

If `IS NULL` or `NOT NULL` is selected, the Value field is disabled. **Values** must be entered without quotes (or any other delimiter). Values must also have the same formatting as that of the corresponding DB field; otherwise the expression will evaluate to `FALSE`. For example, if a criterion for a field of the `date` datatype in an MS Access DB has an expression `StartDate=25/05/2004`, the expression will evaluate to `FALSE` because the `date` datatype in an MS Access DB has a format of `YYYY-MM-DD`.

Using parameters with DB Queries

You can enter the name of a **parameter** as the value of an expression when creating queries. Parameters are variables that can be used instead of literal values in queries. When you enter it in an expression, its value is used in the expression. Parameters that are available have been defined by the SPS designer in the SPS and can be viewed in the View Parameters dialog (see *screenshot below*). Parameters have been assigned a default value in the SPS, which can be overridden by passing a value to the parameter via the command line (if and when the output document is compiled via the command line).

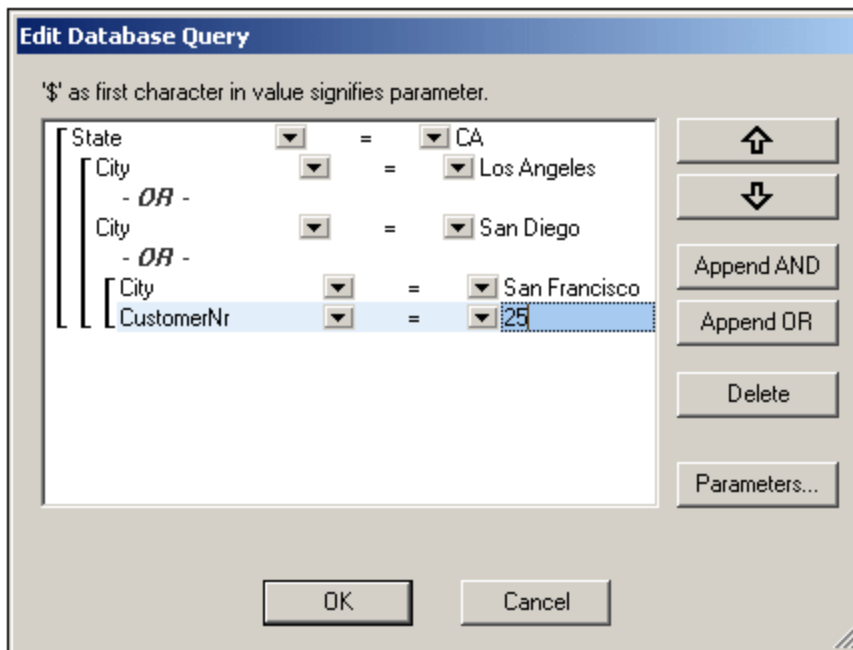
To view the parameters defined for the SPS, click the **Parameters** button in the Edit Database Query dialog. This opens the **View Parameters** dialog (see *screenshot*).



The View Parameters dialog contains **all** the parameters that have been defined for the stylesheet in the SPS and parameters must be edited in the stylesheet design.

Re-ordering criteria in DB Queries

The logical structure of the DB Query and the relationship between any two criteria or sets of criteria is indicated graphically. Each level of the logical structure is indicated by a square bracket. Two adjacent criteria or sets of criteria indicate the AND operator, whereas if two criteria are separated by the word **OR** then the OR operator is indicated. The criteria are also appropriately indented to provide a clear overview of the logical structure of the DB Query.



The DB Query shown in the screenshot above may be represented in text as:

```
State=CA AND (City=Los Angeles OR City=San Diego OR (City=San Francisco AND  
CustomerNr=25))
```

You can re-order the DB Query by moving a criterion or set of criteria up or down relative to the other criteria in the DB Query. To move a criterion or set of criteria, do the following:

1. Select the criterion by clicking on it, or select an entire level by clicking on the bracket that represents that level.
2. Click the Up or Down arrow button in the dialog.


The following points should be noted:

- If the adjacent criterion in the direction of movement is at the same level, the two criteria exchange places.
- A set of criteria (i.e. criterion within a bracket) changes position within the same level; it does not change levels.
- An individual criterion changes position within the same level. If the adjacent criterion is further outward/inward (i.e. not on the same level), then the selected criterion will move outward/inward, **one level at a time**.

To delete a criterion in a DB Query, select the criterion and click **Delete**.

Modifying a DB Query



To modify a DB Query:

1. Click the Query button . The Edit Database Query dialog box opens. You can now edit the expressions in any of the listed criteria, add new criteria, re-order criteria, or delete criteria in the DB Query.
2. Click **OK**. The data from the DB is automatically re-loaded into Authentic View so as to reflect the modifications to the DB Query.

11.2.3.3 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](#)⁶³⁹).

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.

Note the following points:

- 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.

Note: Saving data to the DB resets the Undo command, so you cannot undo actions that were carried out prior to the save.

11.2.4 Working with Dates

There are two ways in which dates can be edited in Authentic View:

- Dates are entered or modified using the [Date Picker](#)⁶⁴⁵.
- Dates are entered or modified by [typing in the value](#)⁶⁴⁶.

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.

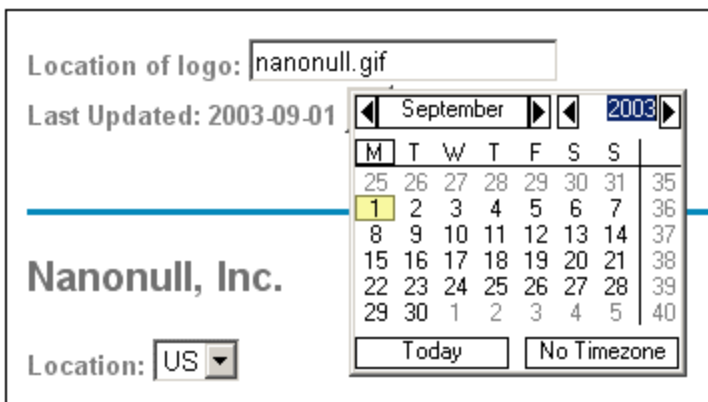
In order to ensure that the date is entered in the correct format, the SPS designer can include the graphical Date Picker in the design. This would ensure that the date selected in the Date Picker is entered in the correct lexical format. If there is no Date Picker, the Authentic View should take care to enter the date in the correct lexical format. Validating the XML document could provide useful tips about the required lexical format.

11.2.4.1 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.

11.2.4.2 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.

Invoice Number: 001 2006-03-10 Customer: The ABC Company Invoice Amount: 40.00

Errors

The following types of error will be flagged:

- 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*).

Invoice Number: 001 2006-03-32 Customer: ERROR: Invalid value for datatype date in element Invoice Amount: 40.00

- If you try to change the format of the date, the date turns red to alert you to the error. (In the screenshot below, slashes are used instead of hyphens).

Invoice Number: 001 2006/03/10 Customer: The ABC Company Invoice Amount: 40.00

11.2.5 Defining Entities

About 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.

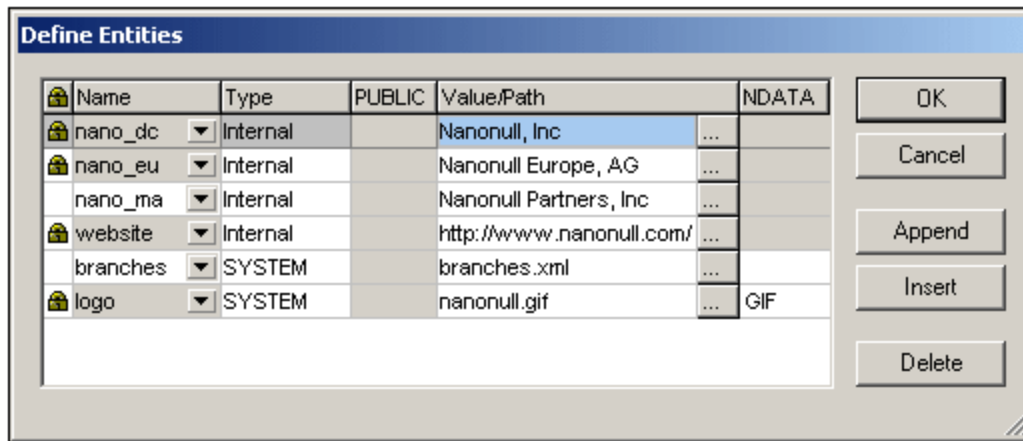
Types of entity

There are two broad types of entity 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.

Defining entities

To define an entity:

1. Click **Authentic | Define XML Entities**. This opens the Define Entities dialog (*screenshot below*).



2. Enter the name of your entity in the Name field. This is the name that will appear in the Entities Entry Helper.
3. Enter the type of entity from the drop-down list in the Type field. The following types are possible: An **Internal** entity is one for which the text to be used is stored in the XML document itself. Selecting **PUBLIC** or **SYSTEM** specifies that the resource is located outside the XML file, and will be located with the use of a public identifier or a system identifier, respectively. A system identifier is a URI that gives the location of the resource. A public identifier is a location-independent identifier, which enables some processors to identify the resource. If you specify both a public and system identifier, the public identifier resolves to the system identifier, and the system identifier is used.
4. If you have selected PUBLIC as the Type, enter the public identifier of your resource in the PUBLIC field. If you have selected Internal or SYSTEM as your Type, the PUBLIC field is disabled.
5. In the Value/Path field, you can enter any one of the following:
 - If the entity type is Internal, enter the text string you want as the value of your entity. Do not enter quotes to delimit the entry. Any quotes that you enter will be treated as part of the text string.
 - If the entity type is SYSTEM, enter the URI of the resource or select a resource on your local network by using the Browse button. If the resource contains parsed data, it must be an XML file (i.e., it must have a .xml extension). Alternatively, the resource can be a binary file, such as a GIF file.

- If the entity type is PUBLIC, you must additionally enter a system identifier in this field.
6. The NDATA entry tells the processor that this entity is not to be parsed but to be sent to the appropriate processor. The NDATA field must therefore contain some value to indicate that the entity is an unparsed entity.

Dialog features

You can do the following in the Define Entities dialog:

- Append entities
- Insert entities
- Delete entities
- Sort entities by the alphabetical value of any column by clicking the column header; clicking once sorts in ascending order, twice in descending order.
- Resize the dialog box and the width of columns.
- Locking. Once an entity is used in the XML document, it is locked and cannot be edited in the Define Entities dialog. Locked entities are indicated by a lock symbol in the first column. Locking an entity ensures that the XML document is 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. `&`.
- 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).

11.2.6 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.

11.2.7 Keystrokes in Authentic View

The Enter key

In Authentic View the **Enter** key is used to append additional elements when it is in certain cursor locations. For example, if the chapter of a book may (according to the schema) contain several paragraphs, then pressing **Enter** inside the text of the paragraph causes a new paragraph to be appended immediately after the current paragraph. If a chapter can contain one title and several paragraphs, pressing **Enter** inside the chapter but outside any paragraph element (including within the title element) causes a new chapter to be appended after the current chapter (assuming that multiple chapters are allowed by the schema).

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.

12 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.

Altova Product	Authentic Scripts Creation	Authentic Scripts Enabled
StyleVision Enterprise	Yes	Yes
StyleVision Professional	Yes	Yes
StyleVision Basic *	No	No
XMLSpy Enterprise	No	Yes
XMLSpy Professional	No	Yes
Authentic Desktop Enterprise	No	Yes
Authentic Browser Ent Trusted **	No	Yes
Authentic Browser Ent Untrusted	No	Yes

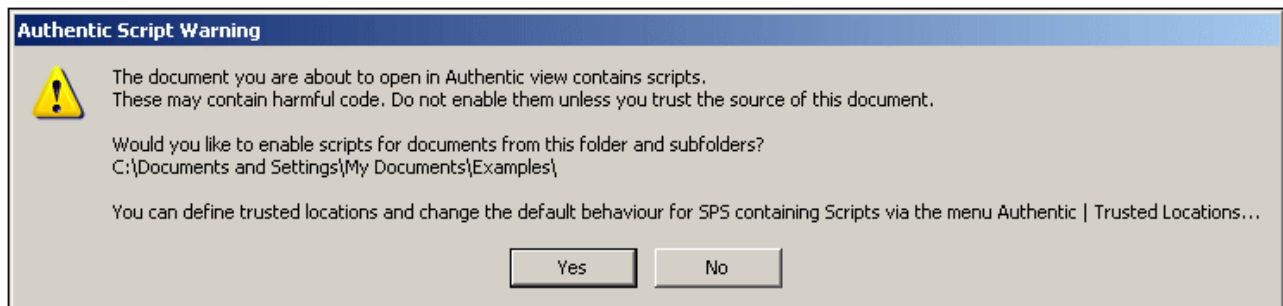
* *No AuthenticView*

** *Scripted designs displayed. No internal macro execution or event handling. External events fired.*

Authentic Scripts behave in the same way in all Altova products, so no product-specific code or settings are required.

Authentic Script Warning Dialog

If a PXF file, or an XML file linked to an SPS, contains a script and the file is opened or switched to Authentic View (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](#)⁶⁵³ 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](#)⁶⁵², which describes the interface in which a scripting project containing the scripts for the SPS are saved
- [Macros](#)⁶⁵³, which shows how macros can be associated with design elements, Authentic toolbar buttons, and Authentic context menus
- [Event Handlers](#)⁶⁵⁹, which shows how event handler scripts are created and used
- [Scripting Options](#)⁶⁶⁰, which documents the options

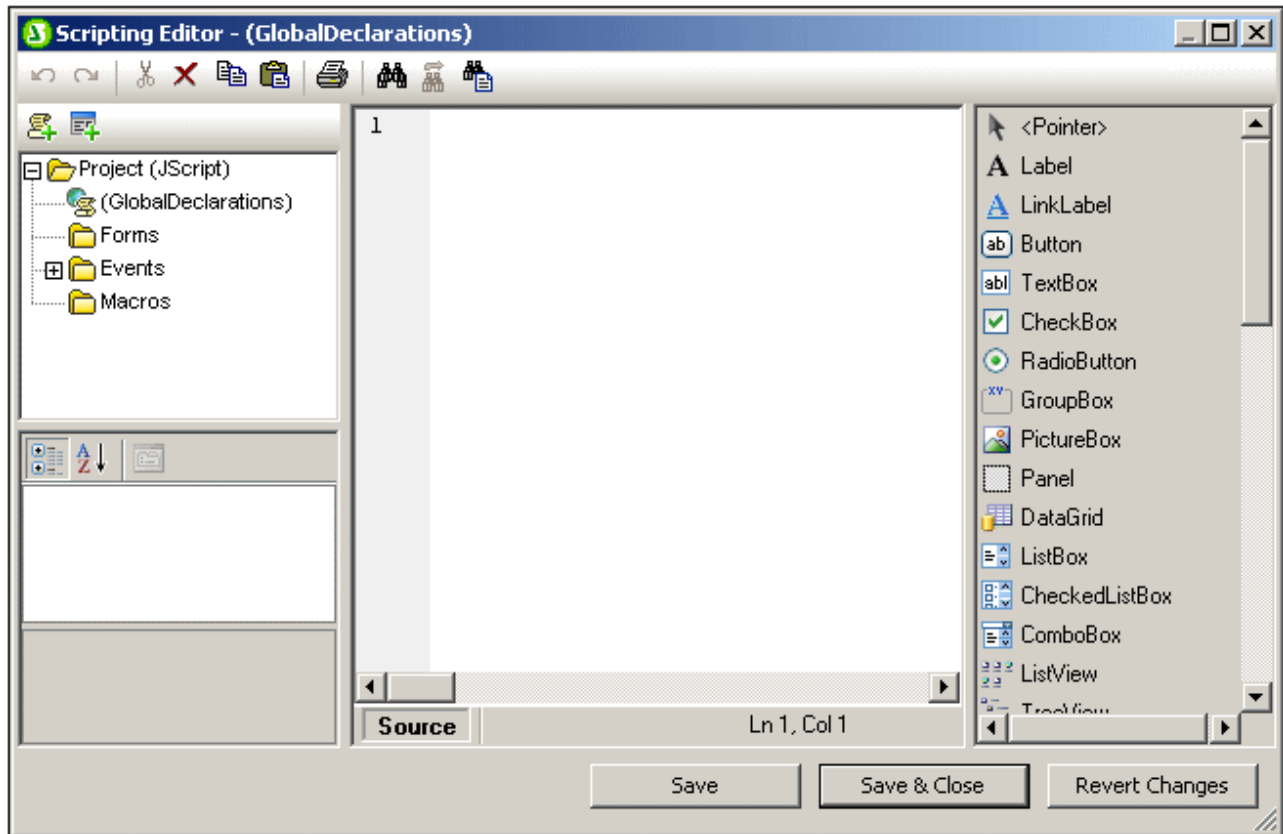
Example files

Example SPS files with Authentic Scripting are in the `Authentic\Scripting` folder of the `Examples` project in the Project window of StyleVision.

12.1 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](#) ⁸³³.



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](#) ⁸⁵⁴ and its features are described in detail in the [Programmers' Reference](#) ⁸⁵³.

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.

12.2 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](#)⁶⁵³, [a toolbar button click](#)⁶⁵⁶, or the selection of [a user-defined context menu item](#)⁶⁵⁵.

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.

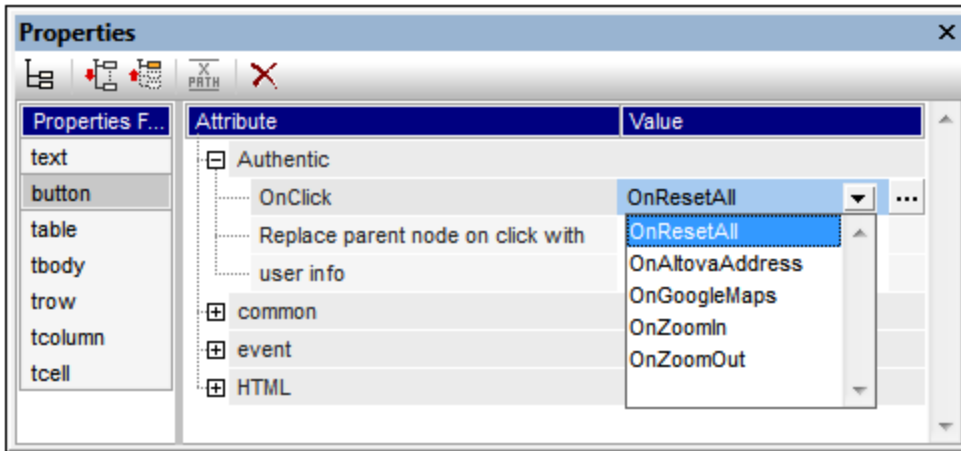
In this section

In this section we look at how macros can be associated with valid user actions:

- [Macros on Design Elements](#)⁶⁵³
- [Macros on Toolbar Buttons](#)⁶⁵⁶
- [Macros on Context Menu Items](#)⁶⁵⁵

12.2.1 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:



Different design elements support different user actions for which macros can be assigned. The table below gives a complete list.

User action	Supported design element	Can cancel event
OnClick	Button, Image	No
OnBeforeLinkClick	Link	Yes
OnBeforeChange	Content, Rest-of-Content, Input Field, Multiline Input Field, Checkbox, Radiobutton, Combobox	Yes
OnAfterChange	Content, Rest-of-Content, Input Field, Multiline Input Field, Checkbox, Radiobutton, Combobox	No
OnSetFocus	Content, Rest-of-Content, Input Field, Multiline Input Field, Checkbox, Radiobutton, Combobox	No
OnKillFocus	Content, Rest-of-Content, Input Field, Multiline Input Field, Checkbox, Radiobutton, Combobox	No

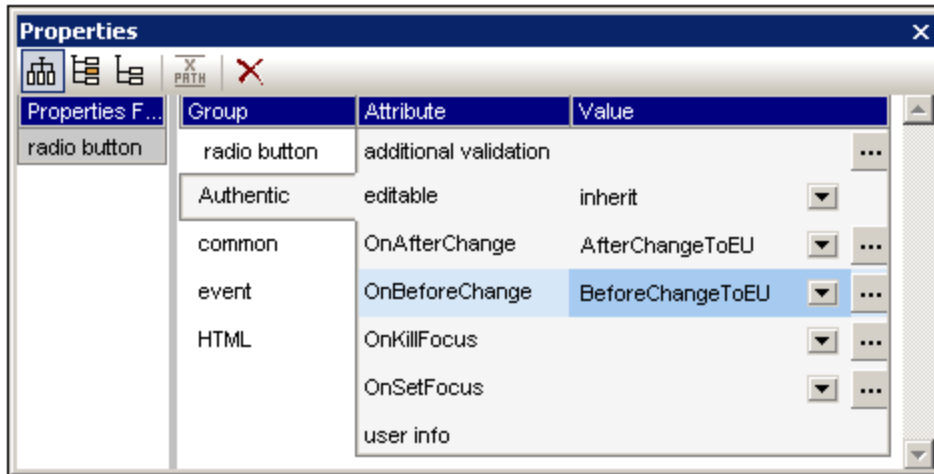
Macros assigned to these user actions will be called only on primary user input. They are not called on an undo action, or when the action was from outside Authentic View (for example, from a COM API call). If a macro is to be used to cancel further event processing, the user needs to place a COM API call `AuthenticView.DoNotPerformStandardAction()` in the macro where it is required. For example, it can be put in the macro `BeforeChangeToEU` (*listing below*) which has been assigned to a `OnBeforeChange` event for a radio button:

```
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` 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

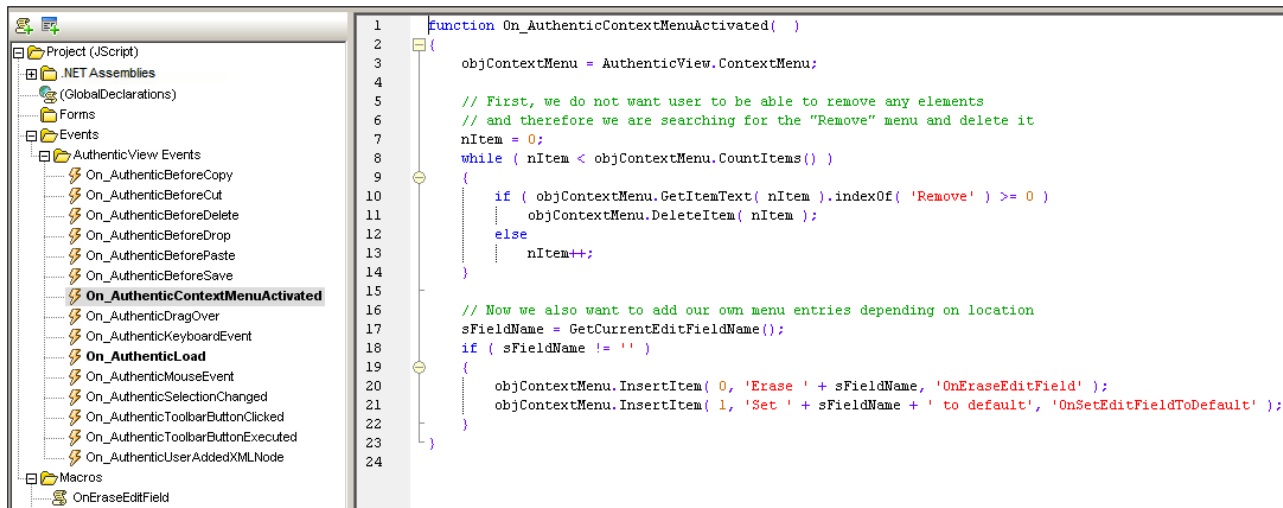
After a macro has been assigned in the Properties window (*screenshot below*), it can be quickly accessed for editing in the Scripting Editor by clicking the **Edit Macro** button to the right of its name (*see screenshot below*).



12.2.2 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](#)⁸⁷⁷. 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.

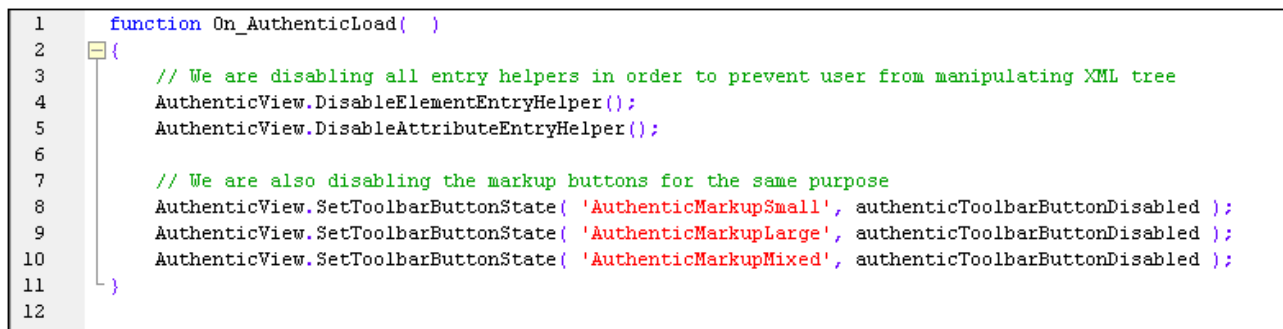


```

1 function On_AuthenticContextMenuActivated( )
2 {
3     objContextMenu = AuthenticView.ContextMenu;
4
5     // First, we do not want user to be able to remove any elements
6     // and therefore we are searching for the "Remove" menu and delete it
7     nItem = 0;
8     while ( nItem < objContextMenu.CountItems() )
9     {
10        if ( objContextMenu.GetItemText( nItem ).indexOf( 'Remove' ) >= 0 )
11            objContextMenu.DeleteItem( nItem );
12        else
13            nItem++;
14    }
15
16    // Now we also want to add our own menu entries depending on location
17    sFieldName = GetCurrentEditFieldName();
18    if ( sFieldName != '' )
19    {
20        objContextMenu.InsertItem( 0, 'Erase ' + sFieldName, 'OnEraseEditField' );
21        objContextMenu.InsertItem( 1, 'Set ' + sFieldName + ' to default', 'OnSetEditFieldToDefault' );
22    }
23 }
24

```

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.



```

1 function On_AuthenticLoad( )
2 {
3     // We are disabling all entry helpers in order to prevent user from manipulating XML tree
4     AuthenticView.DisableElementEntryHelper();
5     AuthenticView.DisableAttributeEntryHelper();
6
7     // We are also disabling the markup buttons for the same purpose
8     AuthenticView.SetToolBarButtonState( 'AuthenticMarkupSmall', authenticToolBarButtonDisabled );
9     AuthenticView.SetToolBarButtonState( 'AuthenticMarkupLarge', authenticToolBarButtonDisabled );
10    AuthenticView.SetToolBarButtonState( 'AuthenticMarkupMixed', authenticToolBarButtonDisabled );
11 }
12

```

12.2.3 Custom Buttons

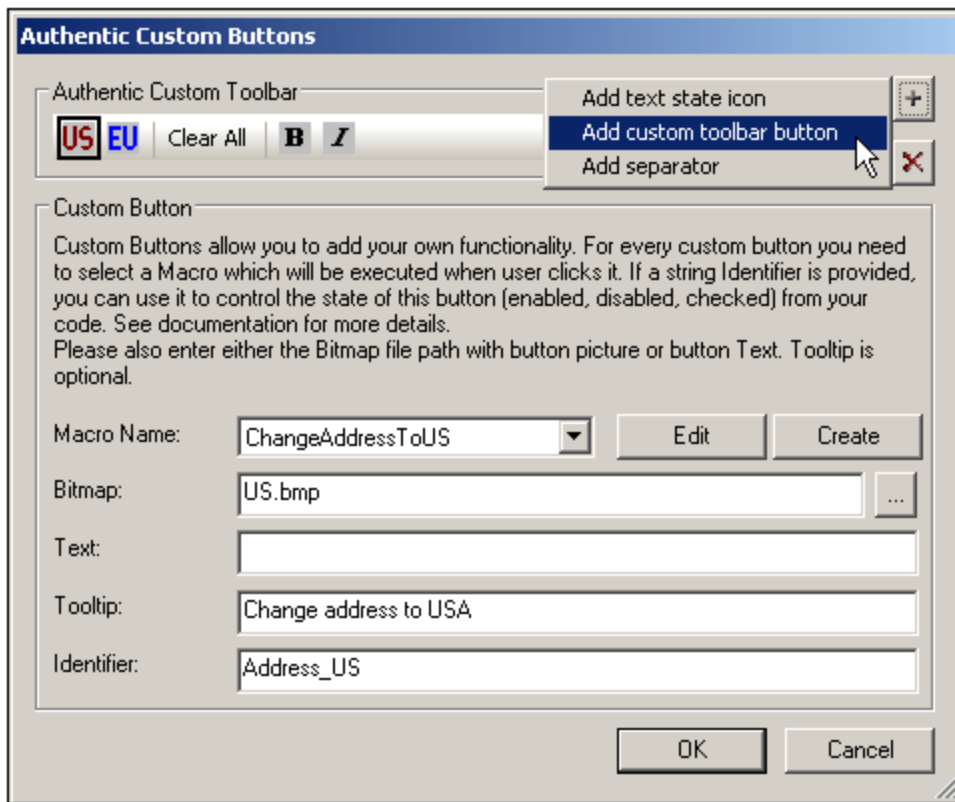
In the SPS design you can create a custom button to add to the Authentic toolbar and specify what macro it will trigger. To add a custom button to the Authentic toolbar, first select the command **Authentic | Custom Toolbar Buttons**, then click the **Add** button at the top right of the Authentic Custom 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:

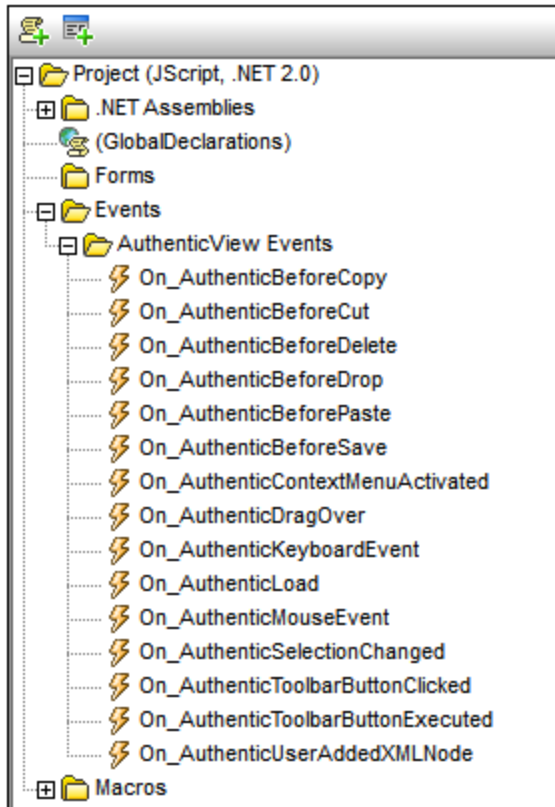
- The location of an image for the button (in the *Bitmap* field) or text for the button (in the *Text* field).
- In the *Element Name* field (available for Text State Icons), enter the name of the element with which you wish to associate the Text State Icon,
- In the *Macro Name* combo box (available 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](#)⁶⁵⁴ 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](#)⁶⁵⁴.
- 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](#)⁶⁷⁷.

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.

12.3 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.

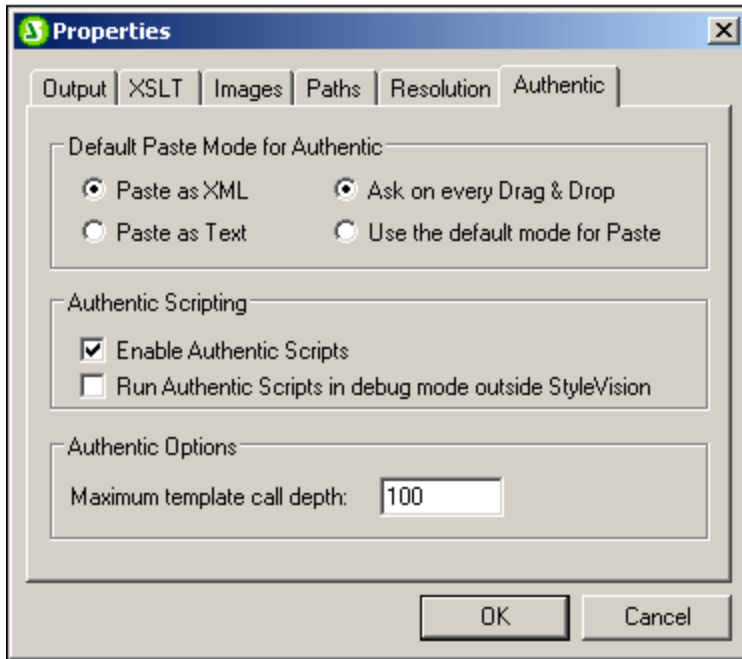
To add an event handler script, start the Scripting Editor (**Authentic | Edit Authentic Scripts**). The available event handlers are displayed in the Project pane under the `AuthenticView Events` node.



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](#)⁶⁵⁵ for an example of how event handlers can be used.

12.4 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.

13 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](#)⁷³⁴ command:

- XSLT files for HTML, Text, and RTF output.
- HTML, Text, and RTF 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, Text, and RTF output files are all one-step processes in which the XML document is transformed by an XSLT stylesheet to the output format.

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](#)⁷³⁴ 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](#)⁶⁶⁵, 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](#)⁶⁶⁵.
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](#)⁶⁶⁷.

13.1 Command Line Interface

StyleVision functionality can be called from the command line in two ways:

- By calling the [StyleVision executable](#)⁶⁶². This provides a access to StyleVision's XSLT-file-generation functionality. The XSLT files are generated from the SPS file.
- By using [StyleVision Server](#)⁶⁶³ 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](#)⁶⁶³, 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](#)⁶⁶⁹.

13.1.1 StyleVision

The syntax for command line use is:

```
StyleVision [<SPS File>] [<options>]
```

where

StyleVision	calls StyleVision, which is located in the StyleVision application folder
<SPS File>	specifies the SPS file
<options>	One or more of the options listed below.

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. Note that FO, PDF, and Word 2007+output-related options are available in the Enterprise edition only; these options are indicated with the words *Enterprise edition* in the list below.

- **XSLT file output**

<code>-OutXSLT=<file></code>	Writes XSLT-for-HTML to the specified file
<code>-OutXSLRTF=<file></code>	Writes XSLT-for-RTF to the specified file
<code>-OutXSLText=<file></code>	Writes XSLT-for-Text to the specified file
<code>-OutXSLFO=<file></code>	Writes XSLT-for-FO to the specified file (<i>Enterprise edition only</i>)
<code>-OutXSLWord2007=<file></code>	Writes XSLT-for-Word 2007+ to the specified file (<i>Enterprise edition only</i>)

- **DB data output**

<code>-OutDBXML=<file></code>	Writes XML generated from DB to the specified file.
<code>-OutDBSchema=<file></code>	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.

13.1.2 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](#)⁴⁸² as its two arguments, and the desired output formats

as its parameters. The XSLT stylesheets for the transformation are obtained from the [PXF file](#)⁴⁸² submitted as input.

An 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](#)⁶⁶⁷ 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 from the specified PXF file:

- HTML (.html) file/s using the XML and XSLT-for-HTML files specified in the PXF, or using alternative XML files
- Text file using the text document and XSLT-for-Text files specified in the PXF
- RTF file using the XML and XSLT-for-RTF files specified in the PXF, or using alternative XML files

13.2 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**⁷³⁴ 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, Text, and RTF 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](#)⁶⁶⁹ 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](#)⁶⁶⁹.

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](#).

13.2.1 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](#)⁷³⁴ 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 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+XBRL) 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
```

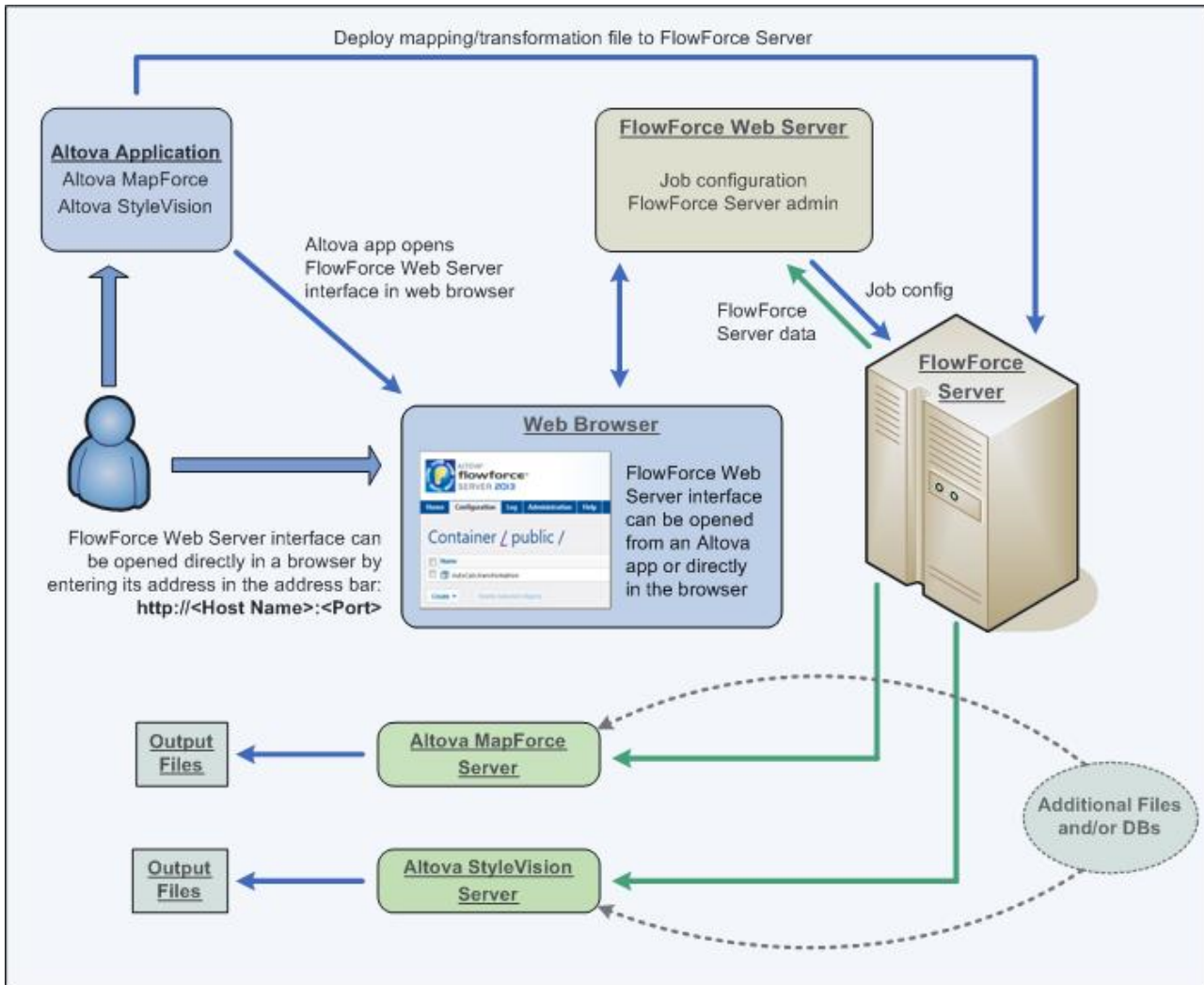
The first command calls RaptorXML and produces `test.fo` as output. The second command passes `test.fo` to the FOP processor, which generates the PDF file `test.pdf`. For more information about batch processing and how batch files can be used to automate processes, see the following section: [How to Automate Processing](#)⁶⁶⁹.

13.3 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](#)⁴⁸² is [deployed to FlowForce Server](#)⁷³⁶ (with the [File | Deploy to FlowForce](#)⁷³⁶ 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.)



Note 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](#).

13.4 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:

```
@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.

14 StyleVision in Visual Studio

StyleVision can be integrated into the Microsoft Visual Studio 2012/2013/2015/2017/2019/2022. 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](#)⁶⁷¹ and the integration of the StyleVision plugin in Visual Studio.
- [Differences](#)⁶⁷² between the Visual Studio version and the standalone version.

14.1 Installing the StyleVision Plugin

To install the StyleVision Plug-in for Visual Studio, take the steps below:

1. Install Microsoft Visual Studio 2012/2013/2015/2017/2019/2022. Note that from Visual Studio 2022 onwards, Visual Studio is being made available only as a 64-bit application.
2. Install StyleVision (Enterprise or Professional Edition). If you have installed Visual Studio 2022+, then you must install the 64-bit version of StyleVision.
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.

Once the integration package has been installed, you will be able to use StyleVision in the Visual Studio environment.

Important

You must use the integration package corresponding to your StyleVision version (current version is 2024). The integration package is not edition-specific and can therefore be used for both Enterprise and Professional editions.

14.2 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](#)³¹.)

- 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.

Note: In Visual Studio 2019 and later, StyleVision functionality can be accessed in the **Extensions** menu of Visual Studio. In earlier versions of Visual Studio, StyleVision features are available in top-level menus of Visual Studio.

15 StyleVision in Eclipse

Eclipse is an open source framework that integrates different types of applications delivered in the form of plugins. The StyleVision Integration Package for Eclipse enables you to integrate and access the functionality of StyleVision in the Eclipse Platform for Windows. Supported Eclipse versions are: 2024-03 (4.31), 2023-12 (4.30), 2023-09 (4.29), 2023-06 (4.28).

In this section, we describe the following:

- [How to install the Integration Package for Eclipse and integrate StyleVision in Eclipse](#) ⁶⁷⁴
- [StyleVision Perspective in Eclipse](#) ⁶⁷⁶
- [Other StyleVision Entry Points in Eclipse](#) ⁶⁷⁹

15.1 Install the Integration Package for Eclipse

Prerequisites

- Eclipse 2024-03 (4.31), 2023-12 (4.30), 2023-09 (4.29), 2023-06 (4.28) (<http://www.eclipse.org>), 64-bit.
- A Java Runtime Environment (JRE) or Java Development Kit (JDK) for the 64-bit platform.
- StyleVision Enterprise or Professional Edition 64-bit.

Note: All the prerequisites listed above must have the 64-bit platform. Integration with older Eclipse 32-bit platforms is no longer supported, although it may still work.

After the prerequisites listed above are in place, you can install the StyleVision Integration Package (64-bit) to integrate StyleVision in Eclipse. The integration can be carried out either during the installation of the Integration Package or manually from Eclipse after the Integration Package has been installed. The StyleVision Integration Package is available for download at <https://www.altova.com/components/download>.

Note: Eclipse must be closed while you install or uninstall the StyleVision Integration Package.

Integrate StyleVision during installation of the Integration Package

You can integrate StyleVision in Eclipse during the installation of the StyleVision Integration Package. Do this as follows:

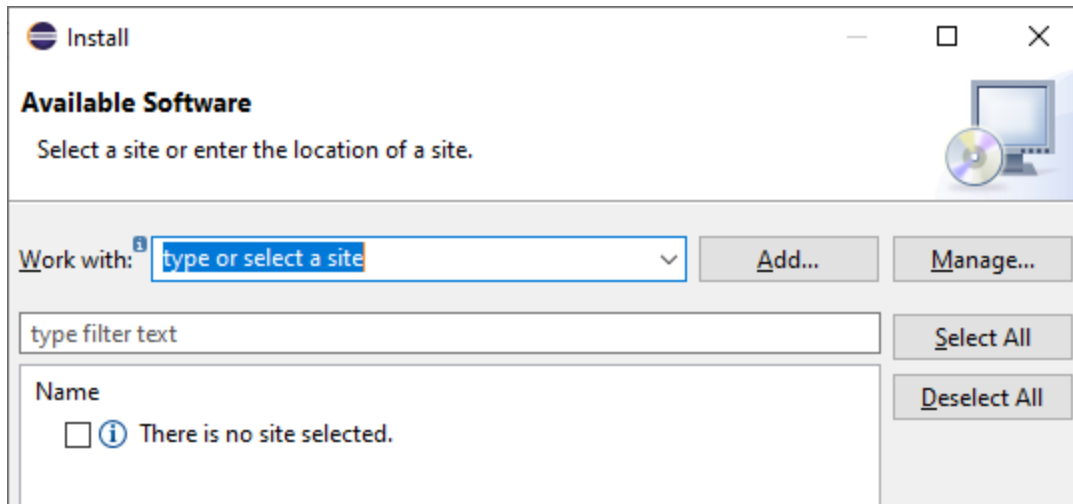
1. Run the StyleVision Integration Package to start the installation wizard.
2. Go through the initial steps of the installation with the wizard.
3. In the Integration step, select *Let this wizard integrate Altova StyleVision plug-in into Eclipse*, and browse for the directory where the Eclipse executable (`eclipse.exe`) is located.
4. Click **Next** and complete the installation.

The StyleVision perspective and menus will be available in Eclipse the next time you start it.

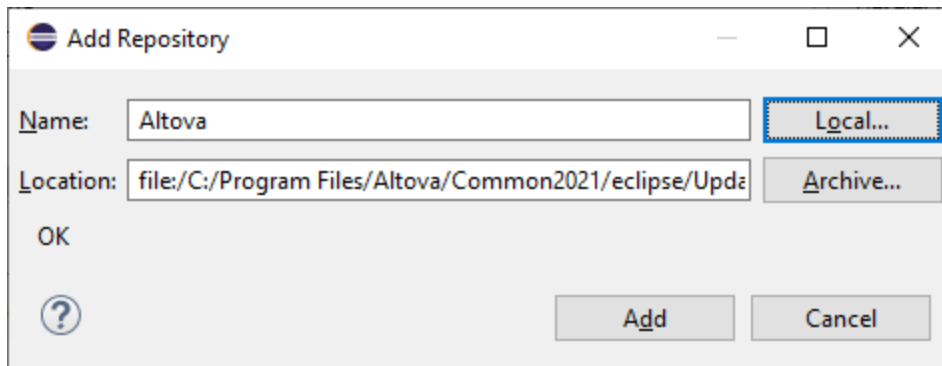
Integrate StyleVision in Eclipse manually

After you have installed the StyleVision Integration Package, you can manually integrate StyleVision in Eclipse as follows:

1. In Eclipse, select the menu command **Help | Install New Software**.
2. In the Install dialog box, click **Add**.



3. In the Add Repository dialog box, click **Local**. Browse for the folder `C:\Program Files\Altova\Common2024\eclipse\UpdateSite`, and select it. Provide a name for the site (such as "Altova").



4. Repeat the steps 2-3 above, this time selecting the folder `C:\Program Files\Altova\Authentic\eclipse\UpdateSite` and providing a name such as "Altova StyleVision".
5. On the Install dialog box, select *Only Local Sites*. Next, select the "Altova category" folder and click **Next**.
6. Review the items to be installed and click **Next** to proceed.
7. To accept the license agreement, select the respective check box.
8. Click **Finish** to complete the installation.

Note: If there are problems with the plug-in (missing icons, for example), start Eclipse from the command line with the `-clean` flag.

15.2 StyleVision Perspective in Eclipse

In Eclipse, a perspective is a GUI view that is configured with the functionality of a specific application. After StyleVision has been integrated in Eclipse, a new perspective, named StyleVision, becomes available in Eclipse. This perspective is a GUI that resembles the StyleVision GUI and includes a number of its components.

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 (*see below*), thus allowing you to edit or process that file in another environment.

There are therefore two main advantage of perspectives:

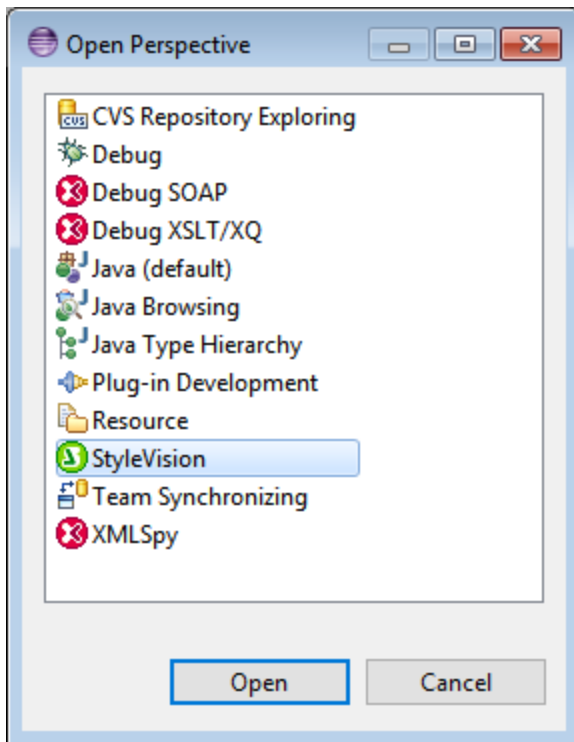
1. Being able to quickly change the working environment of the active file, and
2. Being able to switch between files without having to open a new development environment (the associated environment is available in a perspective)

Working with the StyleVision perspective involves the following key procedures, which are described further below:

- Switching to the StyleVision perspective.
- Setting preferences for the StyleVision perspective.
- Customizing the StyleVision perspective.

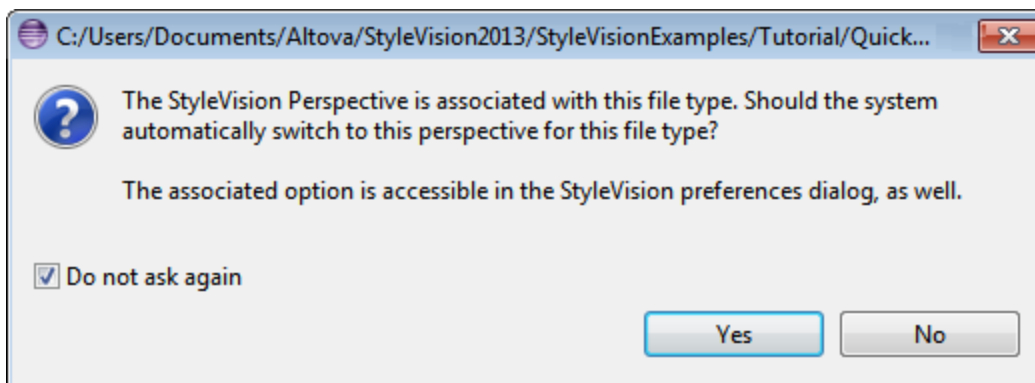
Switch to the StyleVision perspective

In Eclipse, select the command **Window | Perspective | Open Perspective | Other**. In the dialog that appears (*screenshot below*), select **StyleVision**, and click **Open**.



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, you can set the required perspective to 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**.

Preferences for the StyleVision perspective

To access the Preferences of a perspective, 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**.

The preferences of a perspective include:

- To automatically switch to the StyleVision perspective when a file of an associated filetype is opened (*see above*)
- Options for including or excluding individual StyleVision toolbars
- Access to StyleVision options.

Customize 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, make that perspective the active perspective and select the command **Window | Perspective | Customize Perspective**.

- In the *Toolbar Visibility* and *Menu Visibility* tabs, you can specify which toolbars and menus are to be displayed.
- In the *Action Set Availability* tab, you can add action sets to their parent menus and to the toolbar. If you wish to enable an action 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 **Apply and Close** to complete the customization and for the changes to take effect.

15.3 Other Stylevision Entry Points in Eclipse

In addition to the StyleVision perspective, two other entry points in Eclipse can be used to access StyleVision functionality:

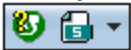
- StyleVision menu
- StyleVision toolbar

StyleVision menu in Eclipse

The **StyleVision** menu of Eclipse contains StyleVision commands that provide key StyleVision functionality. These commands occur in various menus of the standalone version of StyleVision.

StyleVision toolbar in Eclipse

The StyleVision toolbar in Eclipse (*screenshot below*) contains two buttons.



These buttons do the following:

- Open the StyleVision Help
- Provide access to StyleVision commands (as an alternative to accessing them from the **StyleVision** menu, *see above*).

Note: Toolbar commands are not supported. If you have set up a toolbar command in StyleVision that runs a command or script, then this toolbar command will not be available in the plug-in.

16 Menu Commands and 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](#)⁶⁸¹.
- A description of the [Edit XPath Expression dialog](#)⁶⁸⁵.
- A description of all the [toolbars with their icons](#)⁷⁰², 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](#)¹²⁴
- [SPS File Structure](#)²⁰⁰
- [SPS File Advanced Features](#)²⁶⁹
- [SPS File Presentation](#)³⁴⁹
- [SPS File Additional Functionality](#)⁴¹⁵
- [SPS File and Databases](#)⁴⁸⁸

For command line usage, see [Command Line Interface: StyleVisionBatch](#)⁶⁶².

See also

- [User Interface](#)³¹
- [Quick Start Tutorial](#)⁶³

16.1 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](#) ⁶⁸¹
- XML document content
- Data-entry devices
- Predefined formats
- XPath objects
- URI objects

Each of these component types can:

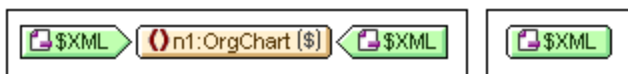
- be moved using drag and drop;
- be cut, copied, pasted, and deleted using (i) the commands in the [Edit menu](#) ⁷⁴⁵, or (ii) the standard Windows shortcuts for these commands;
- have formatting applied to it;
- have a context menu pop up when right-clicked.

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.

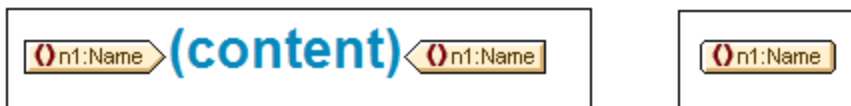
The following types of node are represented:

- **Document node**



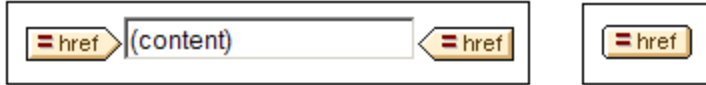
The **document node** (indicated with \$XML) represents the XML document as a whole. It is indicated with a green \$XML tag when the schema source is associated with an XML document, and with \$DB when the schema source is associated with a DB. The document node in the screenshot at left is expanded and contains the `OrgChart` element, which is collapsed. The document node in the screenshot at right is collapsed; its contents are hidden.

- **Element node**



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 kind of templates that can be included in the design, see the section, [Templates and Design Fragments](#)²⁴⁴.

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..

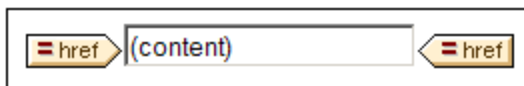
Note: When applied to an element node, the `contents` placeholder does not output the values of attributes of that element. To output attribute nodes, you must explicitly include the attribute in the template (main or global).

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**



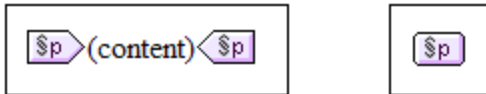
- **Radio buttons**



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](#)¹⁷⁴.

Predefined formats

Predefined formats are shown in mauve tags, which can be expanded/collapsed by double-clicking.

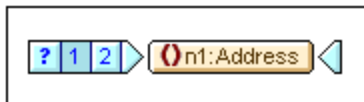


The screenshot above shows tags for the predefined format `p` (*para*), expanded (*at left*) and collapsed (*at right*). To apply a predefined format, highlight the items around which the predefined format is to appear (by clicking a component and/or marking text), and [insert the predefined format](#)³⁵⁰.

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**



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](#) ²⁷⁰.

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)**



Bookmark tags are yellow and indicated with the character `A` (*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**



Link tags are yellow and indicated with the character `A` (*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](#) ⁷⁰²
- [Design sidebars](#) ³⁹
- [Content Editing Procedures](#) ¹²⁴

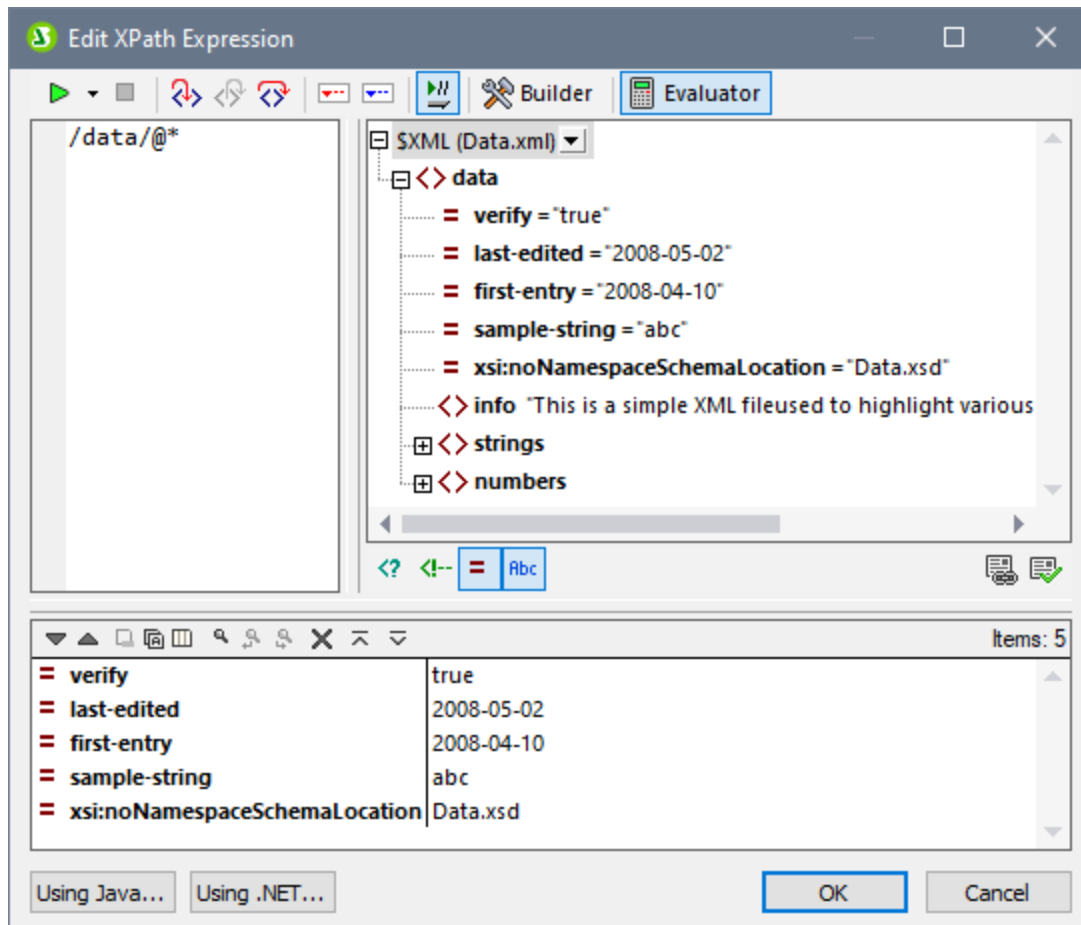
16.2 Edit XPath Expression Dialog

The **Edit XPath Expression** dialog (*screenshot below*) is used to build, test, and edit XPath expressions. It is accessible at all places in Design View where an XPath expression may be entered, such as when entering expressions for [conditional processing](#)²⁸⁰ or the values of [Styles](#)⁵⁴ and [Properties](#)⁵⁵.

The dialog automatically supports the **XPath version** that corresponds to the [XSLT version](#)¹⁰⁵ of the SPS (XPath 1.0 for XSLT 1.0; XPath 2.0 for XSLT 2.0; and XPath 3.1 for XSLT 3.0). To switch the XPath version, switch the [XSLT version of the SPS](#)¹⁰⁵.

Dialog layout

The Edit XPath Expression dialog contains the following panes (*see screenshot below*): (i) an Expression pane (*top left*); (ii) a Sources pane (*top right*); (iii) a Results pane (*bottom*). In Builder Mode, the Results pane is augmented by additional entry helper panes.

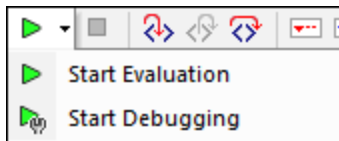


Evaluation Mode and Debug Mode

The Edit XPath Expression dialog has two modes:

- [Evaluation Mode](#) ⁶⁸⁶, in which an XPath expression is evaluated with respect to the assigned Working XML File/s. The expression is entered in the **Expression pane**, and the result is displayed in the **Results pane**. You can click nodes in the result to go to that node in the Sources pane of the dialog.
- [Debug Mode](#) ⁶⁸⁹, in which you can debug an XPath expression as it applies to the assigned Working XML File/s. You can set breakpoints and tracepoints, and go step-by-step through the evaluation. At each step you can see the content of variables, as well as set custom Watch expressions to check additional aspects of the evaluation.

To switch between the two modes, select the appropriate command in the **Start Evaluation/Debugging** dropdown menu that is located in the left-hand corner of the window's toolbar (see *screenshot below*).



How to use the two modes is described in the sub-sections of this section.

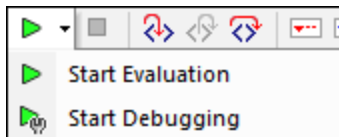
XPath/XQuery Expression Builder

In both modes, the [Expression Builder](#) ⁶⁹⁷ can be used to help you construct syntactically correct expressions.

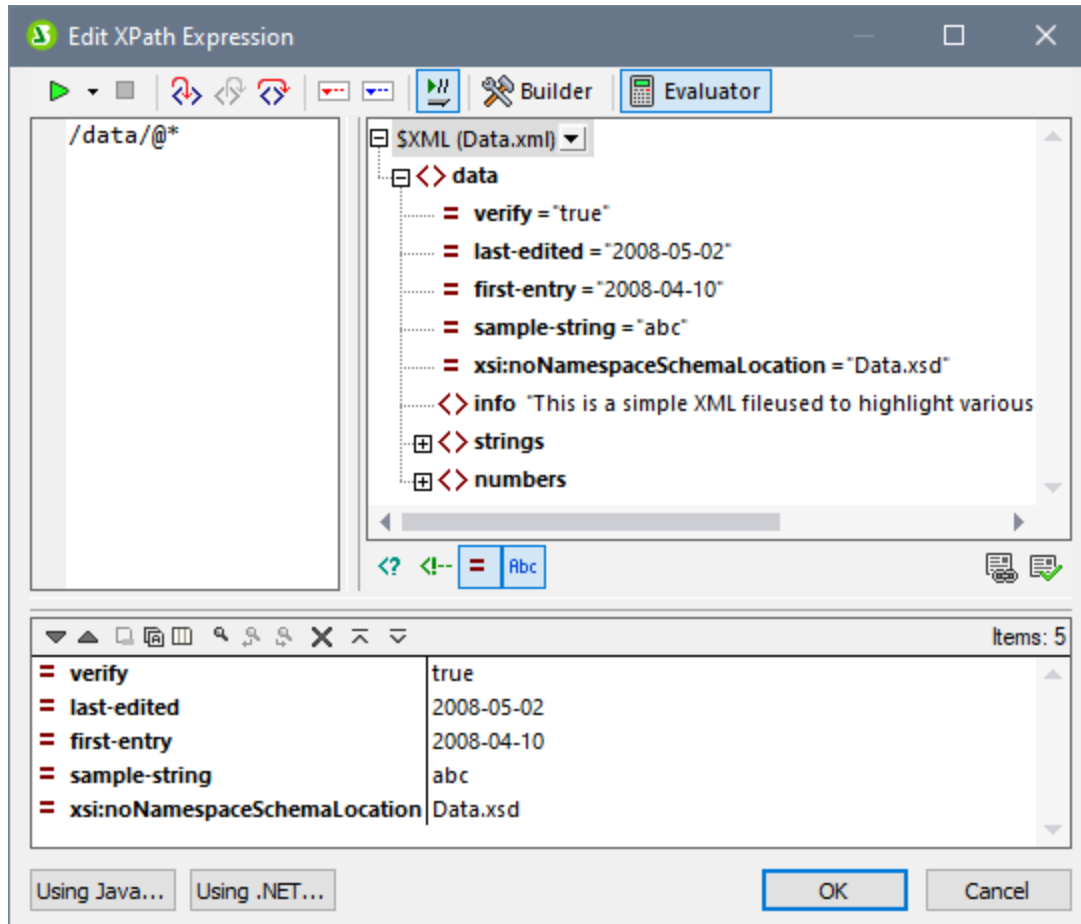
Switch [Expression Builder](#) ⁶⁹⁷ on/off with the **Builder Mode** button of the main toolbar .

16.2.1 Evaluator

Select Evaluation Mode by selecting **Start Evaluation** in the **Start Evaluation/Debugging** dropdown menu (see *screenshot below*).



In Evaluation Mode, click the **Evaluator** button (see *screenshot below*). The evaluator has the following panes (see *screenshot below*): (i) an Expression pane (*top left*); (ii) a Sources pane (*top right*); (iii) a Results pane (*bottom*).



The XPath expression and its evaluation

The XPath expression is entered in the Expression pane. The results of the evaluation are displayed in the Results pane (see screenshot above).

Note the following points:

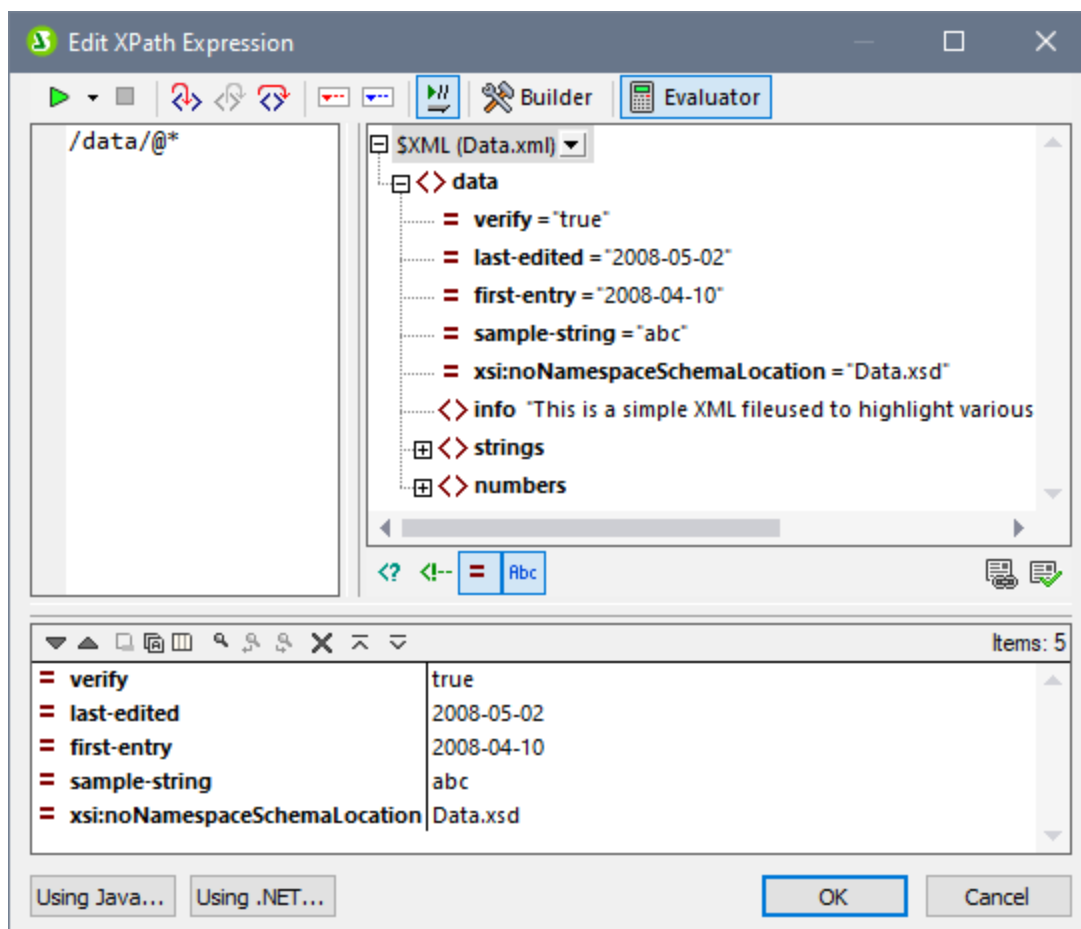
- In order for an expression to be evaluated against an XML file, that file must be assigned as the [Working XML file of one of the sources of the SPS](#) ¹⁰³.
- Results can be displayed even as you type the expression (select the **Evaluate on Typing** icon in the toolbar), or they can be displayed when you click the toolbar button **Start Evaluation/Debugging (F5)** (located at top left of the toolbar).
- To enter the XPath locator expression of a node in a source tree, double-click that node in the Sources pane.
- In the Sources pane, you can switch on/off the display of: (i) processing instructions, (ii) comments, (iii) attributes, and (iv) elements. Do this via the buttons below the Sources pane.
- The context node is that of the design component within which the expression is being created. To set another node as the context node of the expression: (i) select the node in the Sources pane, and (ii) click **Set Evaluation Context** (located below the Sources pane). To save this context node for the expression, click the toggle command button **Remember Evaluation Context** (located below the

Sources pane). Note, however, that the actual context node for the expression will be the context node of the current design component, and this is the context node that will be used at runtime.

- You can use the functions of the Java and .NET programming languages in the XPath expression. The buttons **Using Java** and **Using .NET** at the bottom of the dialog display 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](#)¹⁰⁶⁸ section of this **documentation**.
- To create the expression over multiple lines (for easier readability), use the **Return** key.
- To increase/decrease the size of text in the expression field, click in the expression field, then press **Ctrl** and turn the scroll wheel. **Note that this also applies in the Results pane.**
- Instead of manually entering the locator path expression of a node, you can do the following: (i) Place the cursor at the point in the XPath expression where you want to enter the locator path; (ii) In the Sources tree, double-click the node you want to target. This enters the locator path of the selected node in the expression. The locator path will be an absolute path starting at the root node of the document.

Results pane

The Results pane is shown in the screenshot below, at bottom. Note that it has its own toolbar.



The Results pane has the following functionality:

- The result list consists of two columns: (i) a node name or a datatype; (ii) the content of the node.

- If the XPath expression returns nodes (such as elements or attributes), you can select whether the entire contents of the nodes should be shown as the value of the node. To do this, switch on the toggle *Show Complete Result*.
- When the result contains a node (including a text node)—as opposed to expression-generated literals—clicking that node in the Results pane highlights the corresponding node in the XML document in the Sources tree.
- You can copy both columns of a result sub-line, or only the value column. To copy all columns, right-click a sub-line and toggle on **Copying Includes All Columns**. (Alternatively you can toggle the command on/off via its icon in the toolbar of the Results pane.) Then right-click the sub-line you want to copy and select either **Copy Subline** (for that subline) or **Copy All** (for all sublines).

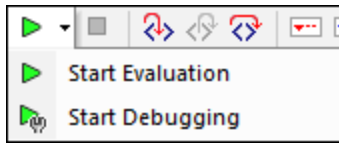
Toolbar of the Results pane

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

Icon	What it does
<i>Next, Previous</i>	Selects, respectively, the next and previous item in the result list
<i>Copy the selected text line to the clipboard</i>	Copies the value column of the selected result item to the clipboard. To copy all columns, toggle on the <i>Copying includes all columns</i> command (<i>see below</i>)
<i>Copy all messages to the clipboard</i>	Copies the value column of all result items to the clipboard, including empty values. Each item is copied as a separate line
<i>Copying includes all columns</i>	Switches between copying (i) all columns, or (ii) only the value column. The column separator is a single space
<i>Find</i>	Opens a <i>Find</i> dialog to search for any string, including special characters, in the result list
<i>Find previous</i>	Finds the previous occurrence of the term that was last entered in the <i>Find</i> dialog
<i>Find next</i>	Finds the next occurrence of the term that was last entered in the <i>Find</i> dialog
<i>Expand with children</i>	Expands the selected item and all its descendants
<i>Collapse with children</i>	Collapses the selected item and all its descendants
<i>Clear</i>	Clears the result list




16.2.2 Debugger

The Debugger enables you to debug an XPath expression in the context of a [Working XML File](#)¹⁰³. To access the Debugger, selecting **Start Debugging** in the **Start Evaluation/Debugging** dropdown menu (*screenshot below*). This sets the mode to Debug Mode. You can then switch between the Builder (for help with building the expression) and Evaluator (for debugging the expression). To start debugging, click **Start Evaluation/Debugging (F5)**



After you have entered an expression, you can start debugging by clicking **Start Evaluation/Debugging (F5)** (after making sure that you are in Debug Mode).

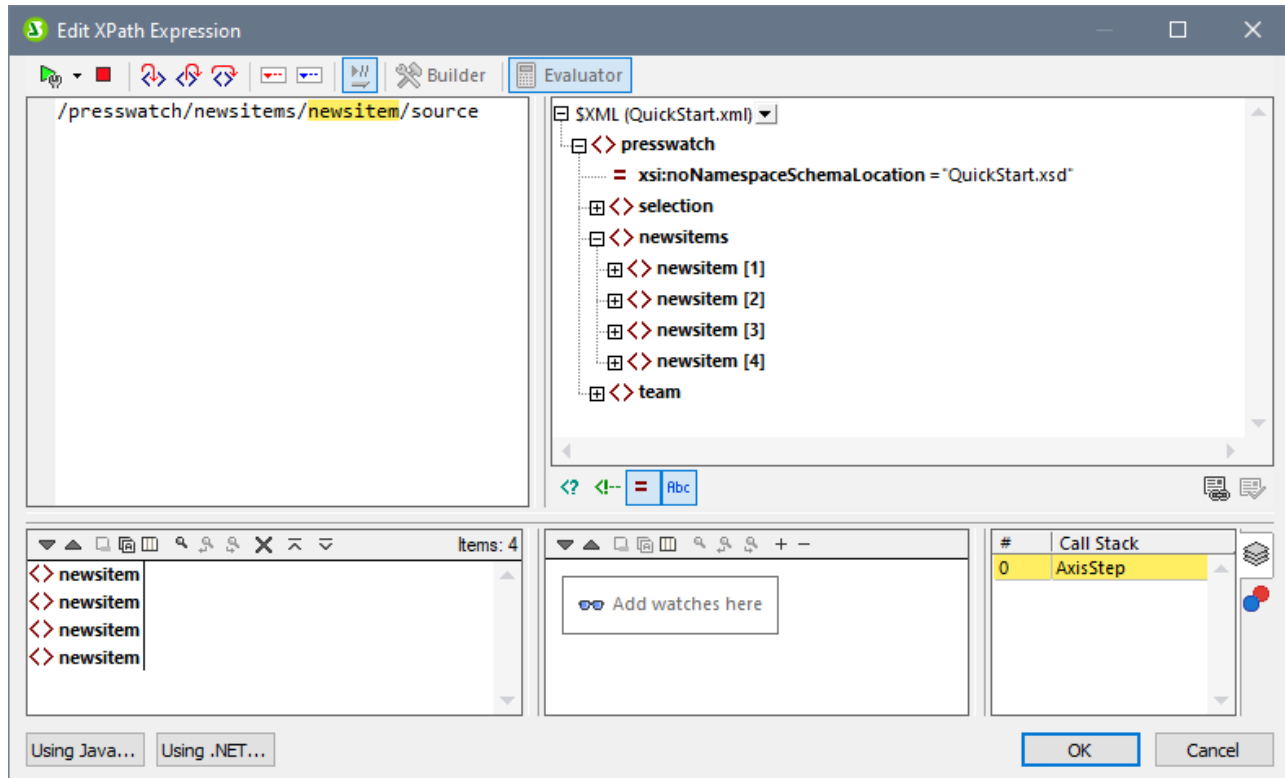
Buttons for setting up Debug Mode

	Start Evaluation/Debugging (F5)	Starts the debugger
	Switch to Builder	Switches to Expression Builder mode, which provides context-sensitive entry helpers to help construct expressions
	Evaluation on typing	Switches on the evaluation of expressions while the expression is being typed

Layout of Debug Mode

In Debug Mode, two additional panes are added to the Results pane (*see screenshot below*):

- the Call Stack and Debug Points pane, each of which has a separate tab in the pane
- the Variables and Watch Expressions pane; both watch expressions and variables are shown in the same pane.



Debugger Mode offers the following features:


- Enables you to step into the XPath evaluation process, one step at a time to see how the XPath expression is being evaluated. Use the **Step Into (F11)** toolbar button for this. At each evaluation step, the part of the expression being currently evaluated is highlighted in yellow (see *screenshot above*), while the result of evaluating that step is shown in the Results pane. For example, in the screenshot above, all the `section` descendant elements of the `book` element have been selected.
- Set breakpoints where you want to pause the evaluation and check results at these points. You can step through the evaluation by pausing only at breakpoints. Use the **Start Debugging (F5)** toolbar button for this. This is quicker than pausing at every step with **Step Into (F11)**.
- Set tracepoints to see a report of results at the steps marked as tracepoints. The evaluation will not pause (except at breakpoints), but the tracepoint results will be displayed in a list in the Results pane.
- Watch expressions can be used to check information (such as document data or aspects of the evaluation). This is especially useful at breakpoints.
- Variables that are in scope, including their values, are displayed in the Variables and Watch Expressions pane.
- Processor calls of an evaluation step are shown in the Call Stack tab of the Call Stack and Debug Points pane.
- If breakpoints and tracepoints have been set, then these are displayed in the Debug Points tab of the Call Stack and Debug Points pane.








For more information about these features, see their descriptions below.

Running the Debugger

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

1. Enter the XPath expression in the expression pane.
2. Set any breakpoints or tracepoints you want. A breakpoint is a point at which the evaluation is paused. A tracepoint is a point in the evaluation that is recorded; tracepoints thus provide a traceable path of evaluation results.
3. If you click **Start Debugger**, evaluation is carried out in one step to the end unless a breakpoint has been marked in the expression. Click **Start Debugger** repeatedly to progress through each breakpoint to the end of the evaluation.
4. Use the Step Into/Out/Over functionality to go step-by-step through the evaluation.

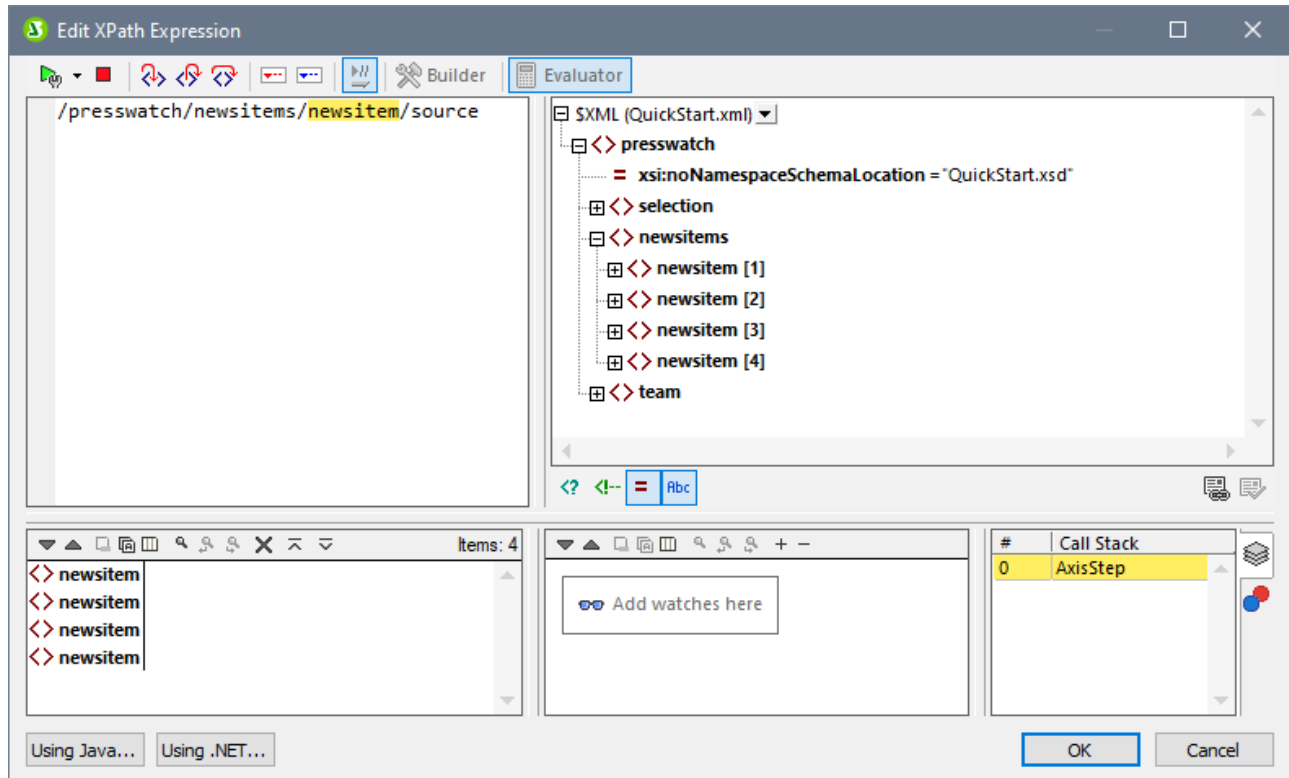
 **Buttons for debugging**

	Start Debugger (F5)	Starts the debugger. Evaluation goes directly to the end, stopping only for breakpoints
	Stop Debugger (Shift+F5)	Exits the evaluation and stops the debugger
	Step Into (F11)	Proceeds through the evaluation, one step at a time.
	Step Out (Shift+F11)	Steps out of the current evaluation step, and goes to the parent step
	Step Over (Ctrl+F11)	Steps over descendant steps
	Insert/Remove Breakpoint (F9)	Inserts/removes a breakpoint at the expression step where you place the cursor
	Insert/Remove Tracepoint (Shift+F9)	Inserts/removes a tracepoint at the expression step where you place the cursor

Stepping in, out, and over evaluation steps

The *Step Into* functionality enables you to go step-by-step through the evaluation. Each click of this command takes you through the next step of the evaluation; the current step is shown by the highlighting in the expression (*see screenshot below*). The *Step Out* functionality takes you to a step on a higher level as the current step, whereas the *Step Over* functionality steps over lower-level steps and takes you to the next step on the same level. You can try out the *Stepping* functionality by using the expression shown in the screenshot below and clicking the three *Step* buttons to see how they work.

The screenshot below shows the evaluation when processing has been paused on reaching the locator step **newsitem**. At this step, the result shows the four `newsitem` node.



Breakpoints

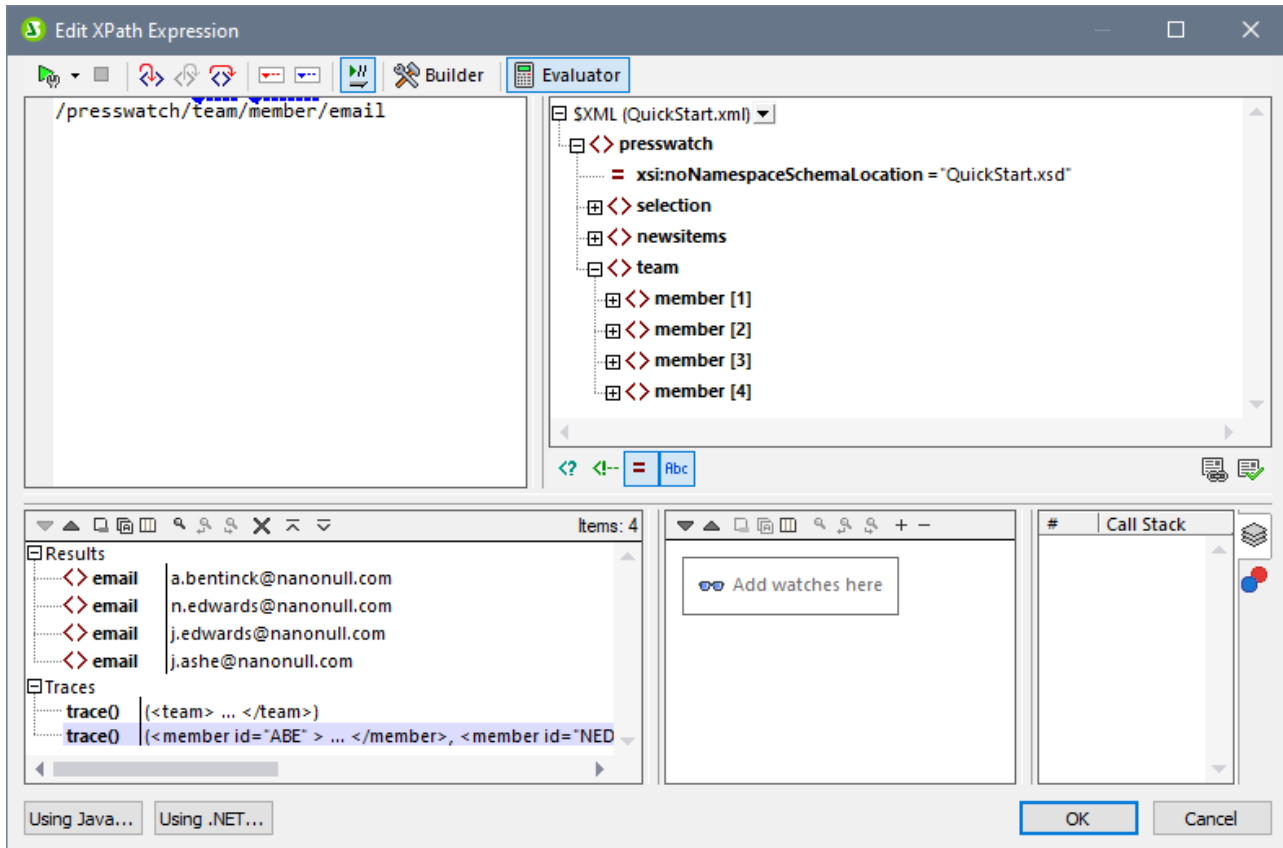
Breakpoints are points where you want the Debugger to stop after it has been started with **Start Debugger**. They are useful if you wish to analyze a specific part of the expression. When the Debugger stops at the breakpoint, you can check the result and could then use the **Step Into** functionality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

Also see [Debug Points](#) ⁶⁹⁶ below.

Tracepoints

Tracepoints are points at which the results are recorded. These results are displayed in the *Traces* tree of the *Result* tab (see *screenshot below*). This enables you to see all the evaluation results of particular parts of the expression. For example, in the screenshot below, tracepoints have been set on the `team` node and `member` node. The results at these tracepoints are shown in the *Traces* tree.

To set a tracepoint, place the cursor at the point where you want the tracepoint, and click the toolbar button **Insert/Remove Tracepoint (Shift+F9)**. The tracepoint will be marked with a dashed blue overline (see *screenshot below*). To remove a tracepoint, select it and click **Insert/Remove Tracepoint (F9)**.

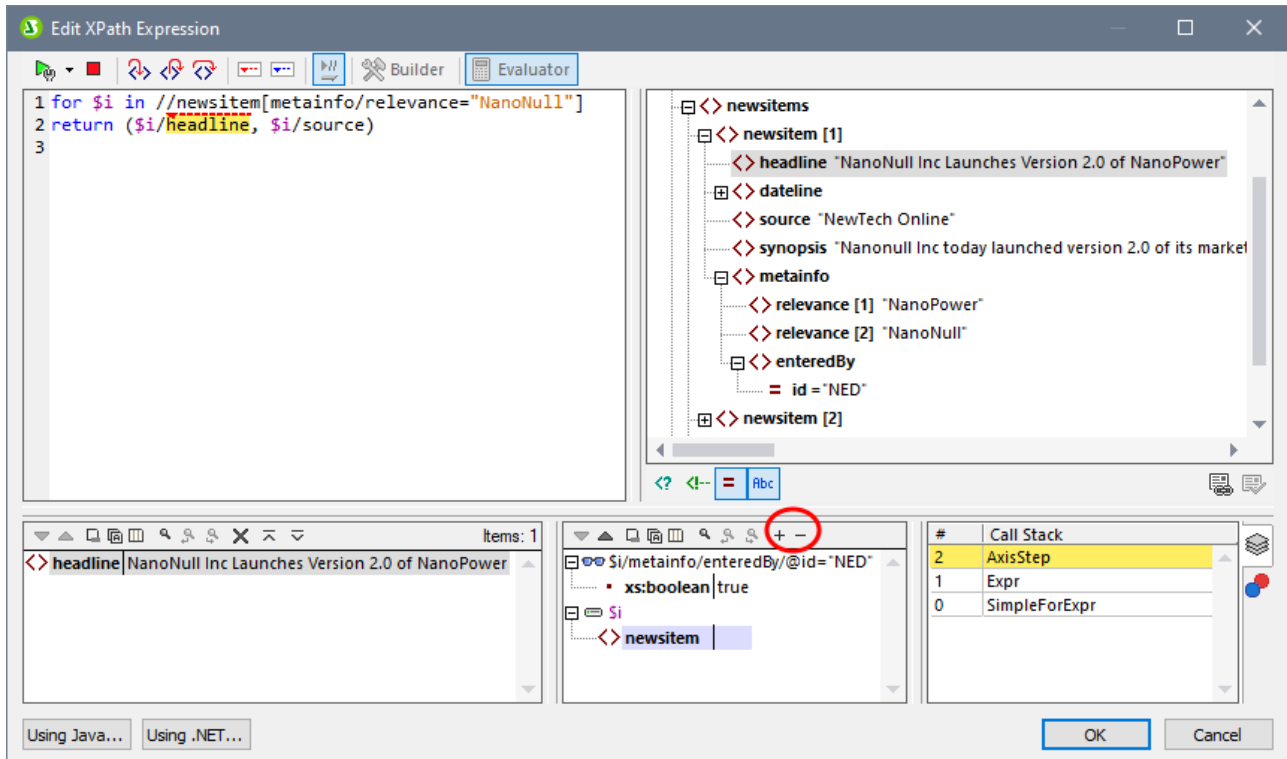


Note: If both a breakpoint and a tracepoint are set on a part of the expression, then the overline is composed of alternating red and blue dashes.

Also see [Debug Points](#)⁶⁹⁶ below.

Variables, Watch Expressions, and Call Stack

Variables and watch expressions are displayed in the Variables and Watch Expressions pane (*bottom center pane in the screenshot below*).



Variables

Variables that have been declared in the expression and that are in scope in the current evaluation step will be displayed together with their respective current values. For example, in the screenshot above, processing has been paused at the breakpoint on `headline`. The `$i` variable is in scope at this evaluation step. So `$i` is displayed with its current value, which in the screenshot above is the first `newsitem` node.

Watch expressions

Watch expressions are expressions that you can enter, either before evaluation starts or during a pause in evaluation. They can be used for the following purposes:

- To test specific conditions. For example in the screenshot above, the watch expression `$i/metainfo/enteredBy/@id="NED"` is used to test whether this news item has been entered by the team member with the id of `NED`. The result `true` in the case of the first news item tells us that this condition has been met.
- To find data within a certain context. For example, within the context of a `company` element, we could enter a watch expression `@id` to look up that company's customer code in the target XML document.
- To generate additional data. For example, a suitable string can be generated to indicate the total number of news items..

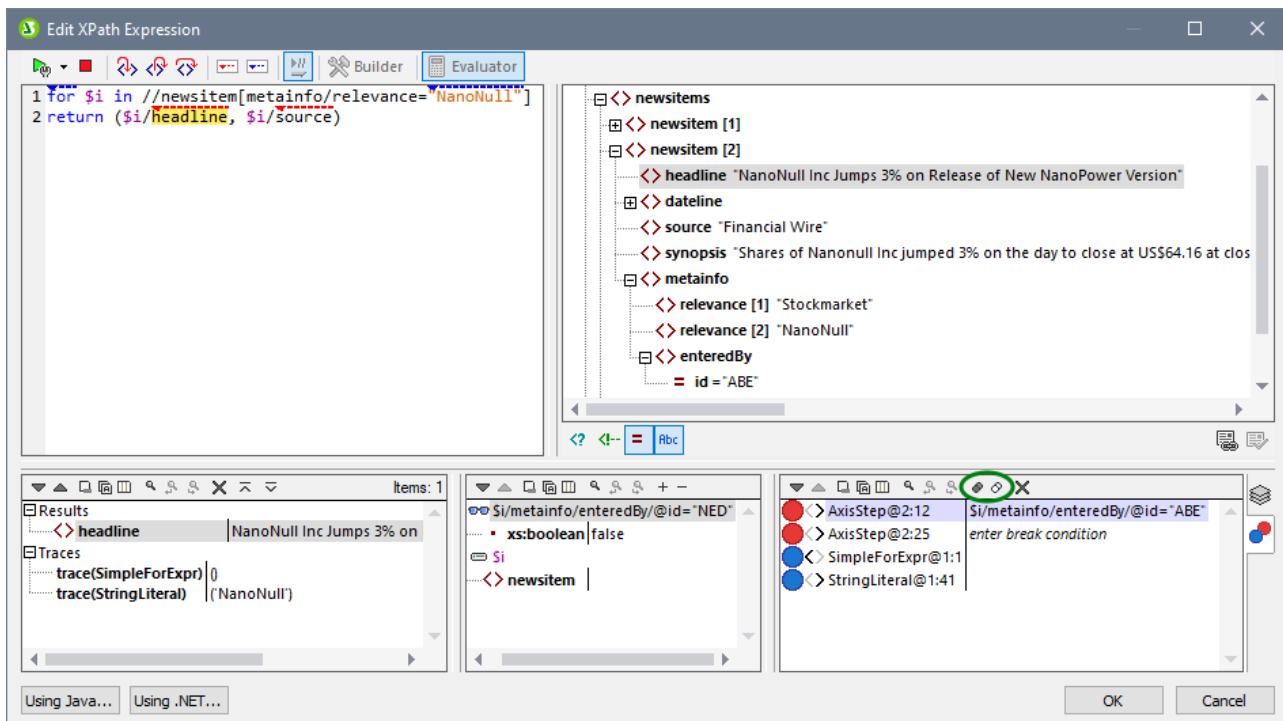
To enter a watch expression, click **Add Watch Entry** in the pane's toolbar (*encircled in red in the screenshot above*), then enter the expression and click **Enter** when done. To remove a watch expression, select it and click **Remove Selected Watch Entry** in the toolbar. If, during debugging, the expression cannot be correctly evaluated for some reason (for example, if one of its variables is out of scope), then the watch expression turns red.

Call stack

The *Call Stack* tab of the Call Stack and Debug Points pane (*bottom right pane in the screenshot above*) displays the processor calls up to that point in the debugging. The current processor call is highlighted in yellow. Note that only the calls that directly led to the current evaluation step are displayed.

Debug Points

The Debug Points tab of the Call Stack and Debug Points pane (*bottom right pane in the screenshot below*) shows the breakpoints (with solid red circles) and tracepoints (solid blue circles) that you have set on the expression. Each debug point is listed with its line and character number. For example, `AxisStep@2:12` means that there is a debug point on line 2, character 12 of the expression in the Expression pane.



Note the following features:

- For breakpoints, you can enter a **break condition** by (i) double-clicking *Enter break condition* in the Debug Points pane, (ii) entering the expression for the condition, and (iii) pressing **Enter**. That breakpoint will be enabled only when the condition evaluates to `true`. For example, in the screenshot above, the break condition `$i/metainfo/enteredBy/@id="ABE"` will enable the breakpoint on the headline of each news item that was entered by the team member with the id `ABE`. The screenshot shows the evaluation paused at this breakpoint. (Notice also that the Watch expression at this breakpoint returns `false`.)
- You can enable/disable all debug points by clicking their respective toolbar buttons: **Enable All Debug Points** and **Disable All Debug Points** (*buttons encircled in green in the screenshot above*). When a debug point is disabled, it is deactivated for all evaluations till it is enabled again.
- You can enable/disable individual breakpoints in their respective context menus.

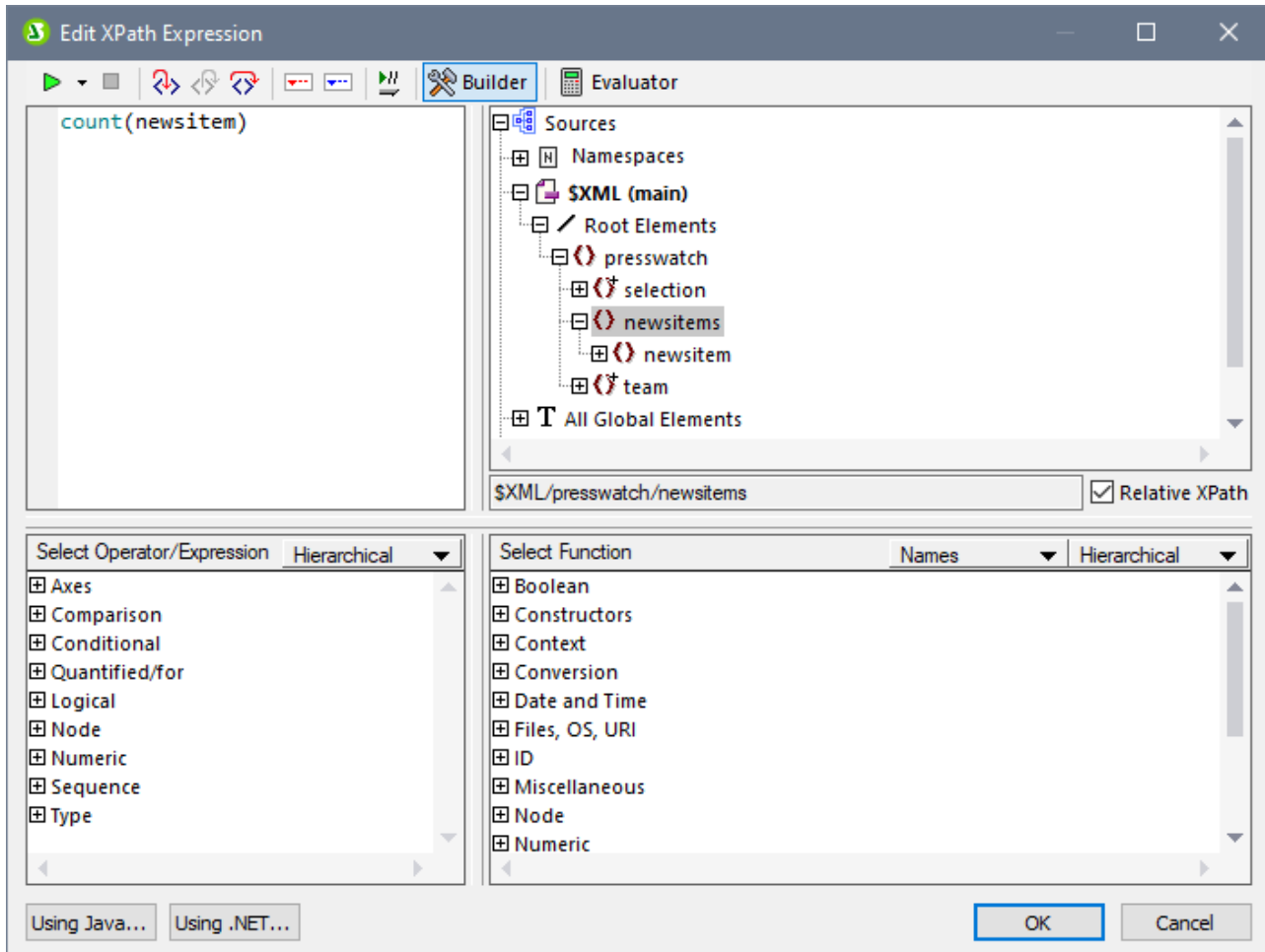
Toolbar commands in panes

The panes of the Edit XPath Expression dialog in Debug Mode (see *screenshot above*) contain buttons that provide navigation, search, and copy functionality. These buttons, starting from the left, are described in the table below. The corresponding commands are also available in the context menu of listed items.

Icon	What it does
<i>Next, Previous</i>	Selects, respectively, the next and previous item in the result list
<i>Copy the selected text line to the clipboard</i>	Copies the value column of the selected result item to the clipboard. To copy all columns, toggle on the <i>Copying includes all columns</i> command (see below)
<i>Copy all messages to the clipboard</i>	Copies the value column of all result items to the clipboard, including empty values. Each item is copied as a separate line
<i>Copying includes all columns</i>	Switches between copying (i) all columns, or (ii) only the value column. The column separator is a single space
<i>Find</i>	Opens a <i>Find</i> dialog to search for any string, including special characters, in the result list
<i>Find previous</i>	Finds the previous occurrence of the term that was last entered in the <i>Find</i> dialog
<i>Find next</i>	Finds the next occurrence of the term that was last entered in the <i>Find</i> dialog
<i>Expand with children</i>	Expands the selected item and all its descendants
<i>Collapse with children</i>	Collapses the selected item and all its descendants
<i>Clear</i>	Clears the result list

16.2.3 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.



There are three entry helper panes:

- A schema tree for entering element and attribute nodes in the XPath expression. If the *Relative XPath* check box is checked, 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). The context node is shown below the schema tree pane. An absolute XPath expression starts at the document root, and is used for the selected node if the *Relative XPath* check box is unchecked.
- An entry helper pane for operators and expressions. These include: (i) axes (`ancestor::`, `parent::`, etc), (ii) operators (for example `eq` and `div`), and (iii) expressions (`for # in # return #`, etc). The items of the pane can be either listed alphabetically or grouped by functional category. Select the option you want by choosing *Hierarchical* or *Flat* from the dropdown menu in the title bar of 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 choosing *Hierarchical* or *Flat* from the dropdown menu in the title bar of the pane. The *Names/Types* option enables you to choose whether the arguments of functions are displayed as names or 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 *Names* or *Types* from the dropdown menu in the title bar of the pane.
- 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.

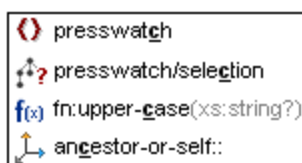
- Context node and schema tree
The *Selection* text box in the *Sources* pane immediately shows you the context node. The expression will be inserted at a location within this context node, and it will be evaluated with this node as its context.
- Inserting a node from the schema tree
In the *Sources* pane, the entire schema is displayed. Double-click a node in the schema tree to insert it in the XPath expression. If the *Relative XPath* check box is checked, the selected node will be inserted with a location path expression that is relative to the context node.
- Namespace information
The schema tree in the *Sources* 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 *Names* or *Types* in the title bar of the pane. 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 displays 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](#)¹¹⁴⁷ section of this **documentation**.

Intelligent editing during direct text entry

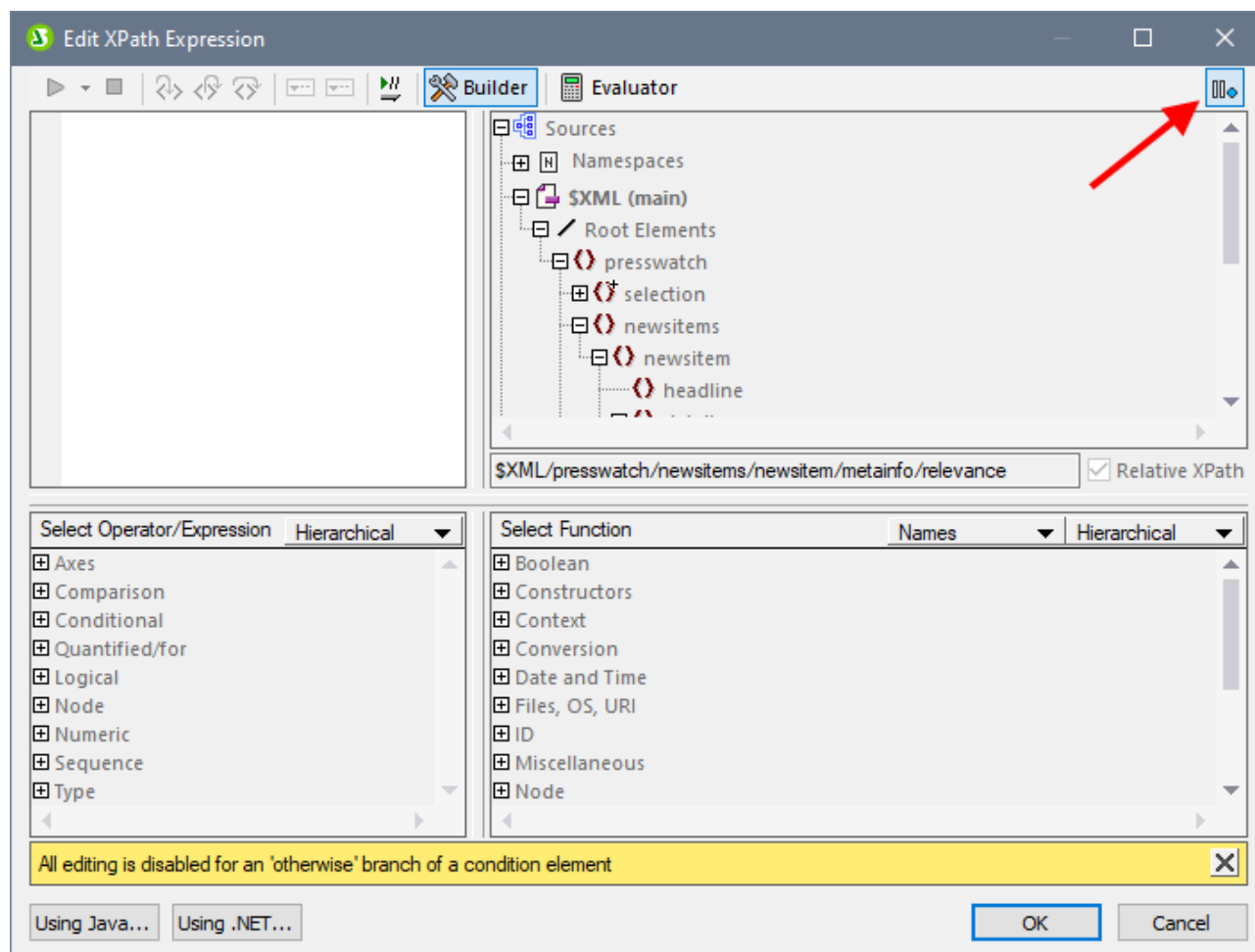
If you type an expression directly in the *Expression* text box, options that are available at that point are displayed in a popup (see *screenshot below*).



These include elements, XPath functions, and XPath axes. 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* toggle (see *the red arrow in the 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](#)²⁸⁰.

16.3 Toolbars

A number of StyleVision commands are available as toolbar shortcuts, organized in the following toolbars:

- [Formatting](#) ⁷⁰⁴
- [Table](#) ⁷⁰⁵
- [Authentic](#) ⁷⁰⁶
- RichEdit
- [Insert Design Elements](#) ⁷⁰⁸
- [Design Filter](#) ⁷¹¹
- [Global Resources](#) ⁷¹²
- [Standard](#) ⁷¹²

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:

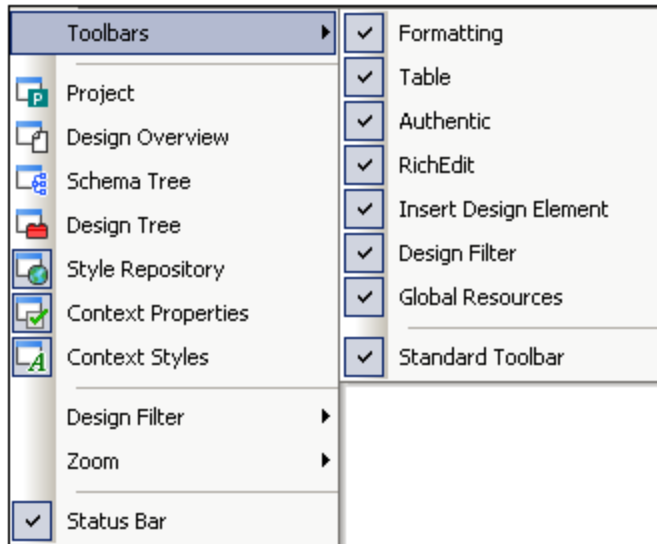
1. Grab the toolbar by its handle (if the toolbar is already in a toolbar area) or by its title bar (if the toolbar is floating).
2. Drag the toolbar to the desired toolbar area, if it exists, and drop it at the desired location in that toolbar area. If no toolbar area exists at the edge along which 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.

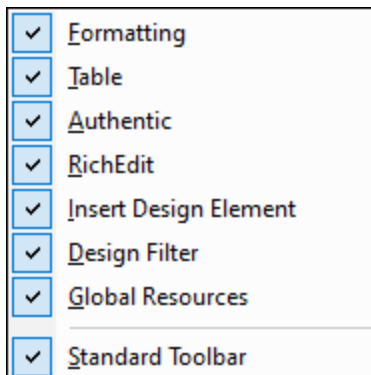
Switching the display of toolbars on and off

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.

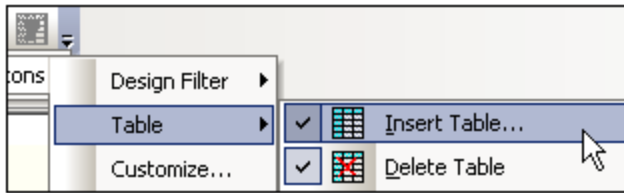


- In the Toolbars tab of the [Customize dialog](#) ⁸³⁵ ([Tools | Customize](#) ⁸³⁵), toggle the display of individual toolbars on or off by clicking a toolbar's check-box. When done, click the **Close** button to close the dialog.

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*).



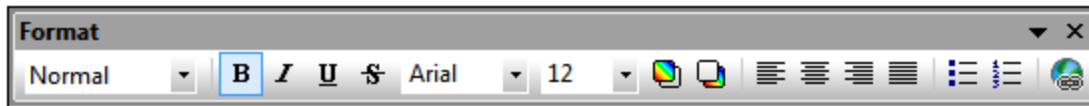
3. Place the cursor over the toolbar that contains the toolbar button to be added or removed (*screenshot above*).
4. In the menu that rolls out (*screenshot above*), click on the name of the toolbar button to add or remove that button from the toolbar.
5. Clicking the Customize item pops up the [Customize dialog](#) ⁸³⁵.

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.

16.3.1 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 properties for the RTF output.

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

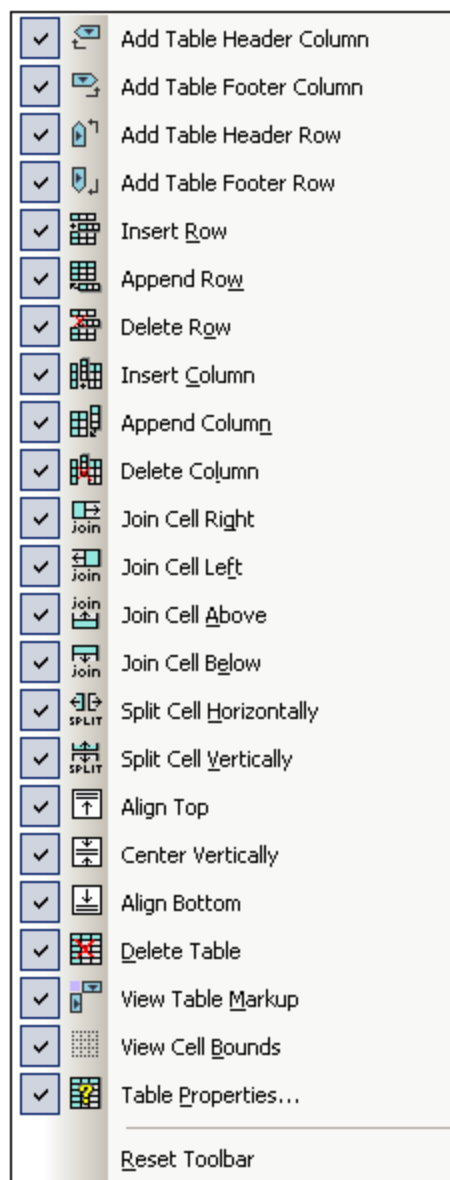
Lists can be inserted at the cursor insertion point, or the selection in the SPS can be converted to a list.

Hyperlinks

Inserts a hyperlink at the cursor insertion point. See [Hyperlink](#) ⁷⁷⁶ for a description of how to use this command.

16.3.2 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).



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.

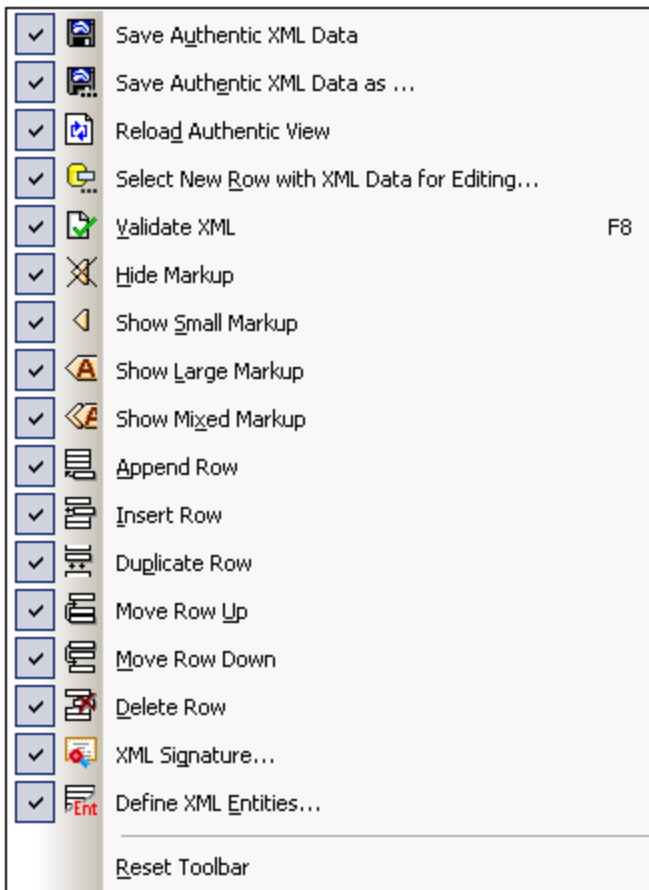
Table operations, properties, display

Placing the cursor in a static or dynamic table and clicking **Delete Table** ⁷⁹⁴ 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.

16.3.3 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](#) ⁷⁰³, 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 users's disposal.



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](#)⁸⁰⁵ 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**).

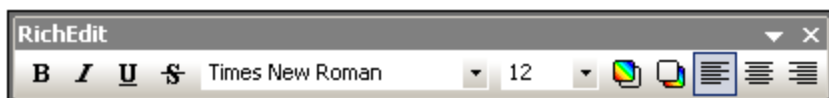
Defining XML Entities

Entities can be defined at any time while editing the Working XML File in Authentic View. Clicking the **Define XML Entities** button pops up the Define XML Entities dialog. (See [Authentic | Define Entities](#)⁸⁰⁶ for details of usage.)

16.3.4 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](#)³⁸³, the buttons and controls in the RichEdit toolbar (*screenshot below*) become enabled. Otherwise they are grayed out.

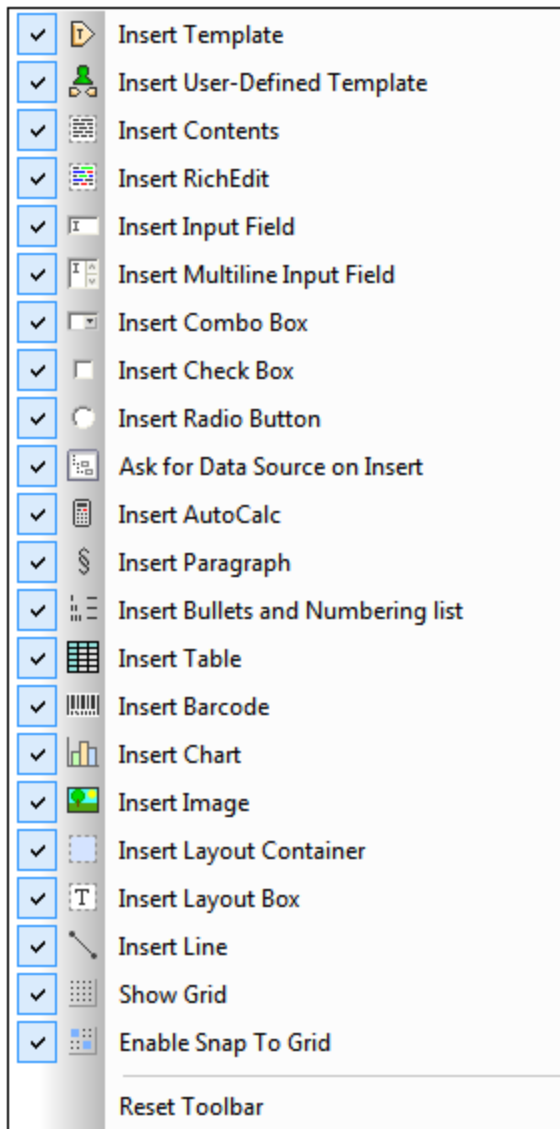


RichEdit enables the Authentic View user to specify the font, font-weight, font-style, font-decoration, font-size, color, background color and alignment of text. Select the text you wish to style in Authentic View and specify the styling you wish to apply via the buttons and controls of the RichEdit toolbar. The text that has been styled will be enclosed in the [tags of the styling element](#)³⁸³.

16.3.5 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:

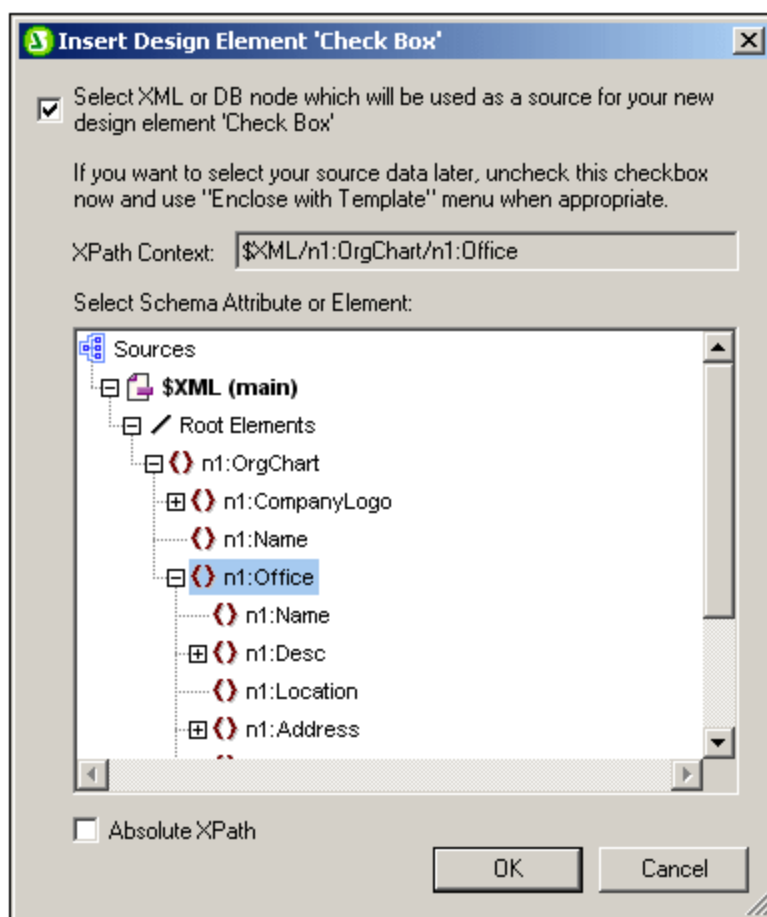
1. [Design elements](#)⁷⁰⁹, which are context-node-sensitive (the majority of elements in the toolbar),
2. [Layout elements](#)⁷¹⁰, which are independent of node context, and
3. [Grid-related toggles](#)⁷¹¹ to aid design.



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](#) ⁶⁶⁵ 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**).

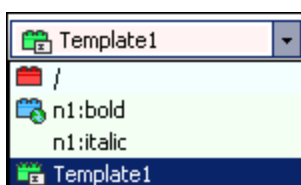
16.3.6 Design Filter

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.



Icon	Command	Description
	Show only one template	Shows the selected template only. Place the cursor in a template and click to show that template only.
	Show all template types	Shows all templates in the SPS (main, global, named, and layout) .
	Show imported templates	Toggles the display of imported templates on and off.
	Show/Hide main template	Toggles the display of the main template on and off.
	Show/Hide global templates	Toggles the display of global templates on and off.
	Show/Hide Design Fragments	Toggles the display of Design Fragments on and off.

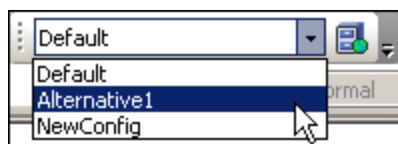
The Design Filter combo box (*screenshot below*) displays a list of all the templates in the SPS.



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.

16.3.7 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](#) ⁴¹⁶.


















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.

16.3.8 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.

Icon	Command	Shortcut	Description
	New from XML Schema / DTD ⁷¹⁴	Ctrl+N	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.
	Open ⁷²⁰	Ctrl+O	Opens an existing SPS document.
	Reload		Reloads the SPS from the last saved version.
	Save Design ⁷²⁵	Ctrl+S	Saves the active SPS document.
	Save All ⁷²⁵	Ctrl+Shift+S	Saves all open SPS documents.
	Print ⁷⁴³	Ctrl+P	Prints the Authentic View of the Working XML file.
	Print Preview ⁷⁴³		Displays a print preview of the Authentic View of the Working XML File.

Icon	Command	Shortcut	Description
	Cut ⁷⁴⁵	Shift+Del	Cuts the selection and places it in the clipboard.
	Copy ⁷⁴⁵	Ctrl+C	Copies the selection to the clipboard.
	Paste ⁷⁴⁵	Ctrl+P	Pastes the clipboard item to the cursor location.
	Delete ⁷⁴⁵	Del	Deletes the selection.
	Undo ⁷⁴⁵	Alt+ Backspace	Undoes an editing change. An unlimited number of Undo actions can be performed at a time.
	Redo ⁷⁴⁵	Ctrl+Y	Redoes an undo.
	Find ⁷⁴⁵	Ctrl+F	Finds text in Authentic View and Output Views.
	Find Next ⁷⁴⁵	F3	Finds the next occurrence of the searched text.
	Zoom ⁷⁶²		Sets the Zoom Factor of Design View.
	Show Small Design Markup		Switches markup tags to small markup format.
	Show Large Design Markup		Switches markup tags to large markup format.
	XSLT 1.0 ¹⁰⁵		Sets XSLT 1.0 as the stylesheet language.
	XSLT 2.0 ¹⁰⁵		Sets XSLT 2.0 as the stylesheet language.
	XSLT 3.0 ¹⁰⁵		Sets XSLT 3.0 as the stylesheet language.
	Spelling ⁸¹⁴		Runs a spelling check on the SPS document.
	Toggle DB Query ⁸⁰⁹		Shows/hides the Database Query dialog.

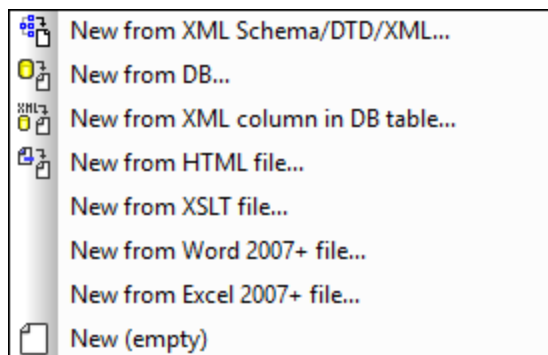
16.4 File Menu

The **File** menu contains commands for working with SPSs and related files. The following commands are available:

- [New](#)⁷¹⁴, to create a new SPS from a variety of sources.
- [Open, Reload, Close, Close All](#)⁷²⁰, to open and close the active file, and to reload the active file.
- [Save Design, Design As, All](#)⁷²⁵, which are commands to save the active SPS and all open SPS files.
- [Export as MobileTogether Design File](#)⁷³³, to generate a MobileTogether design from the active SPS file.
- [Save Authentic XML Data, Save As](#)⁷³³, enabled in Authentic View, it saves changes to the [Working XML File](#)²⁸.
- [Save Generated Files](#)⁷³⁴, to save output files that can be generated using the SPS.
- [Web Design](#)⁷³⁸, generates all the files required to run an ASPX application, in the folder location you specify.
- [Properties](#)⁷³⁸, 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](#)⁷⁴³, enabled in Authentic View and output views, these commands print what is displayed in the previews.
- [Most Recently Used Files, Exit](#)⁷⁴⁴, respectively, to select a recently used file to open, and to exit the program.

16.4.1 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 different 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](#)⁴² and a graphical tree representation is added to the schema tree (in the [Schema Tree sidebar](#)⁴⁴). In [Design View](#)³³, 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](#)⁴⁹². The SPS is created in [Design View](#)³³ with an empty main template.

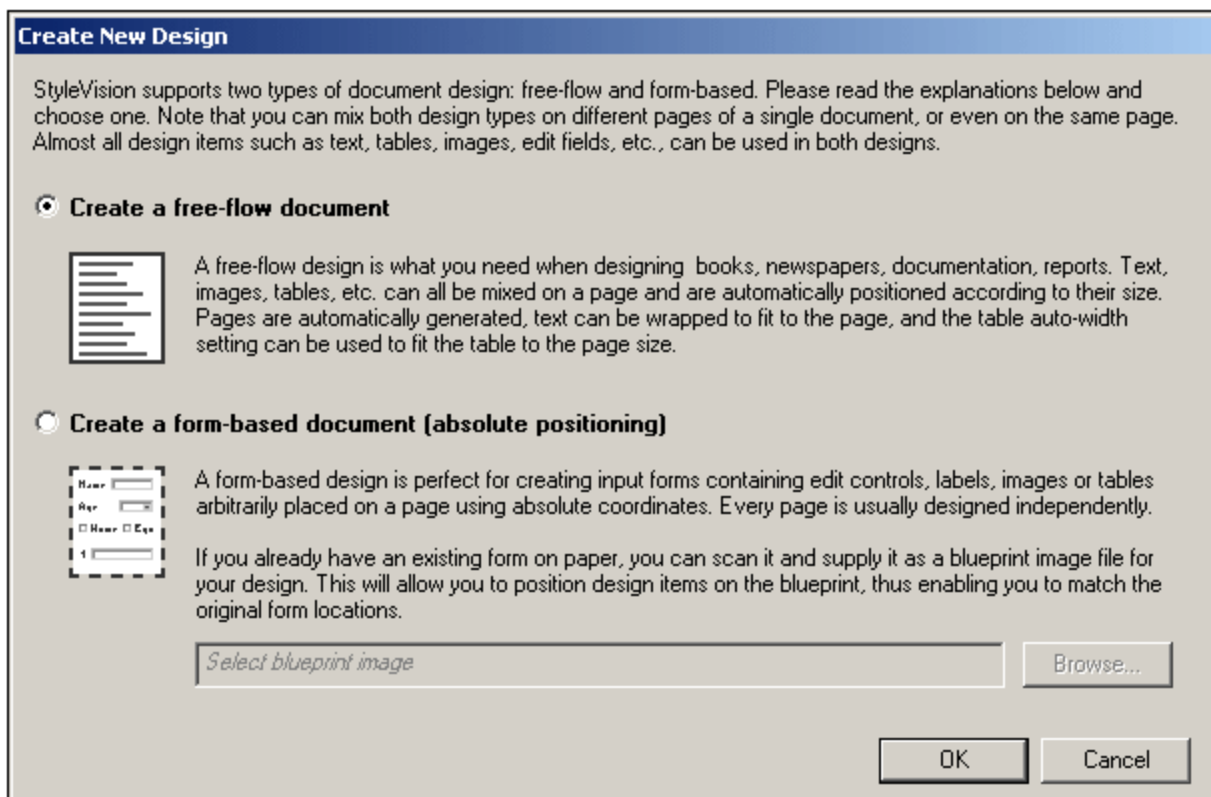
- A new SPS based on a user-defined schema you create node-by-node from an [HTML file](#)⁴⁷² (**New from HTML File**). The user-defined schema is added to the [Design Overview sidebar](#)⁴² and [Schema Tree sidebar](#)⁴⁴. In the schema tree, it will have a single document element (root element), and the HTML file is loaded in [Design View](#)³³.
- 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](#)⁴⁴² for details.
- A new SPS that contains the [content of a MS Word document as the design's static text](#)¹³².
- A new SPS that contains the [content of a MS Excel document as the design's static text](#)¹³⁵.
- 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](#)³³.

Note: A [global resource](#)⁴¹⁶ 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).



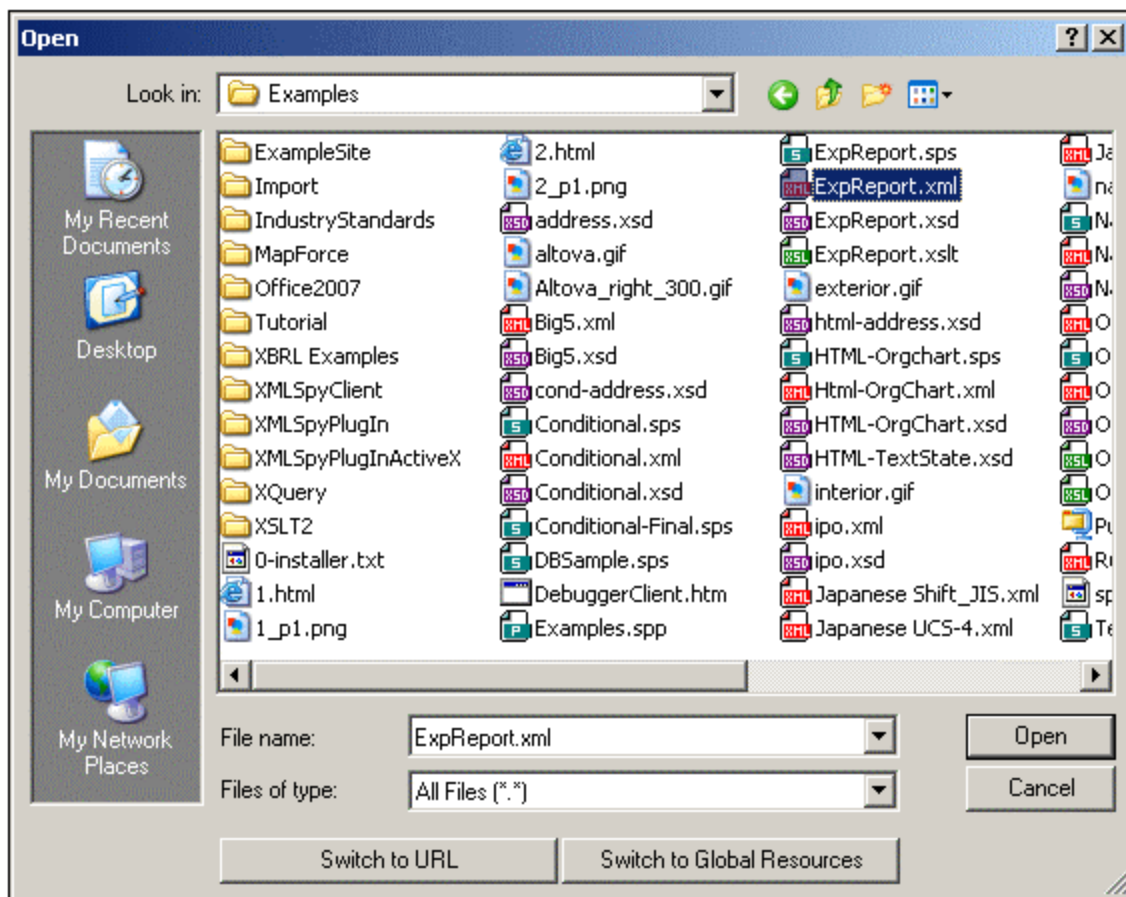
In a free-flowing document design, document content is laid out to fit the output media object or viewer (paper or screen). Items in the document content can only be placed relative to each other, and not absolutely. This kind of design is suited for documents such as reports, articles, and books.

In a form-based document, a single [Layout Container](#)¹⁸⁷ 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](#)¹⁸⁷. 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

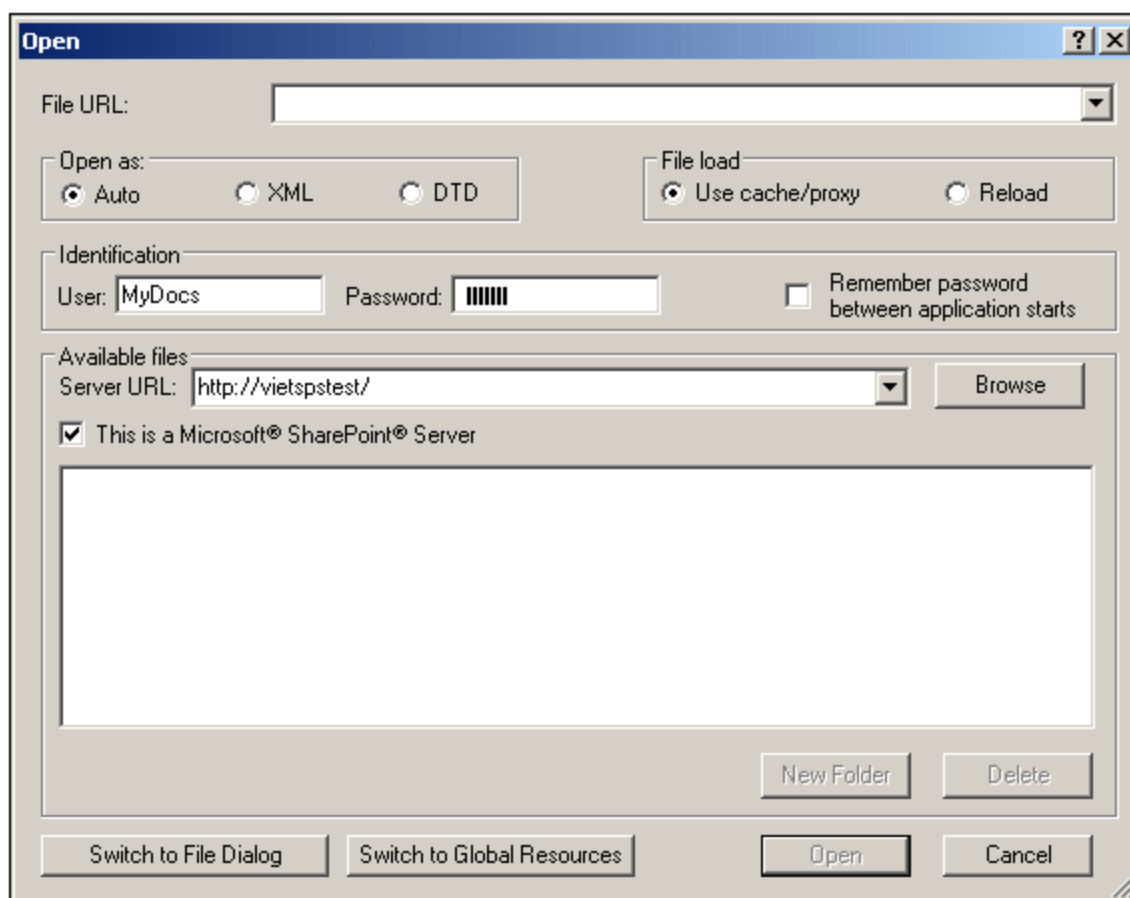
In several File Open and File Save dialogs, you can choose to select the required file or save a file via a URL or a global resource (see *screenshot below*). Select the **Switch to URL** or **Switch to Global Resource** to go to one of these selection processes.



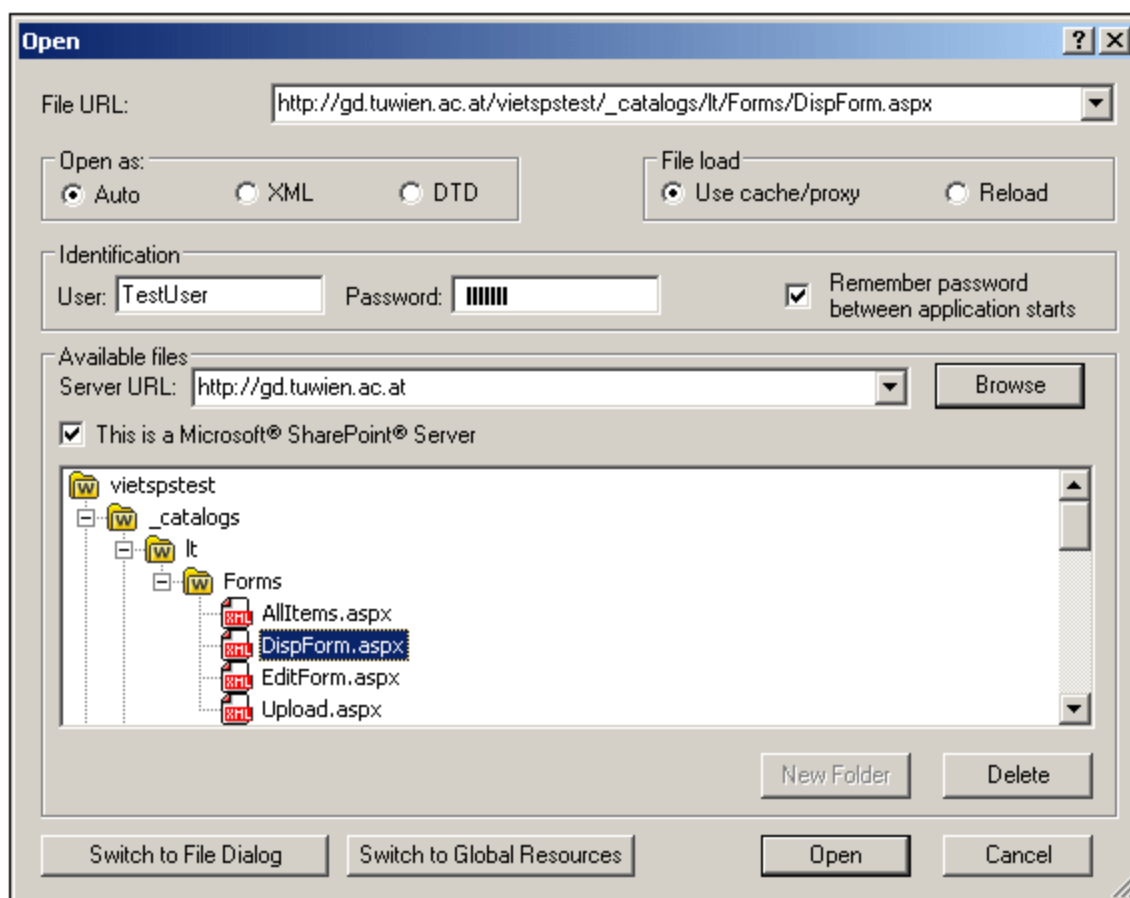
Selecting files via URLs

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.

- Click the **Open** button to load the file. The file you open appears in the main window.

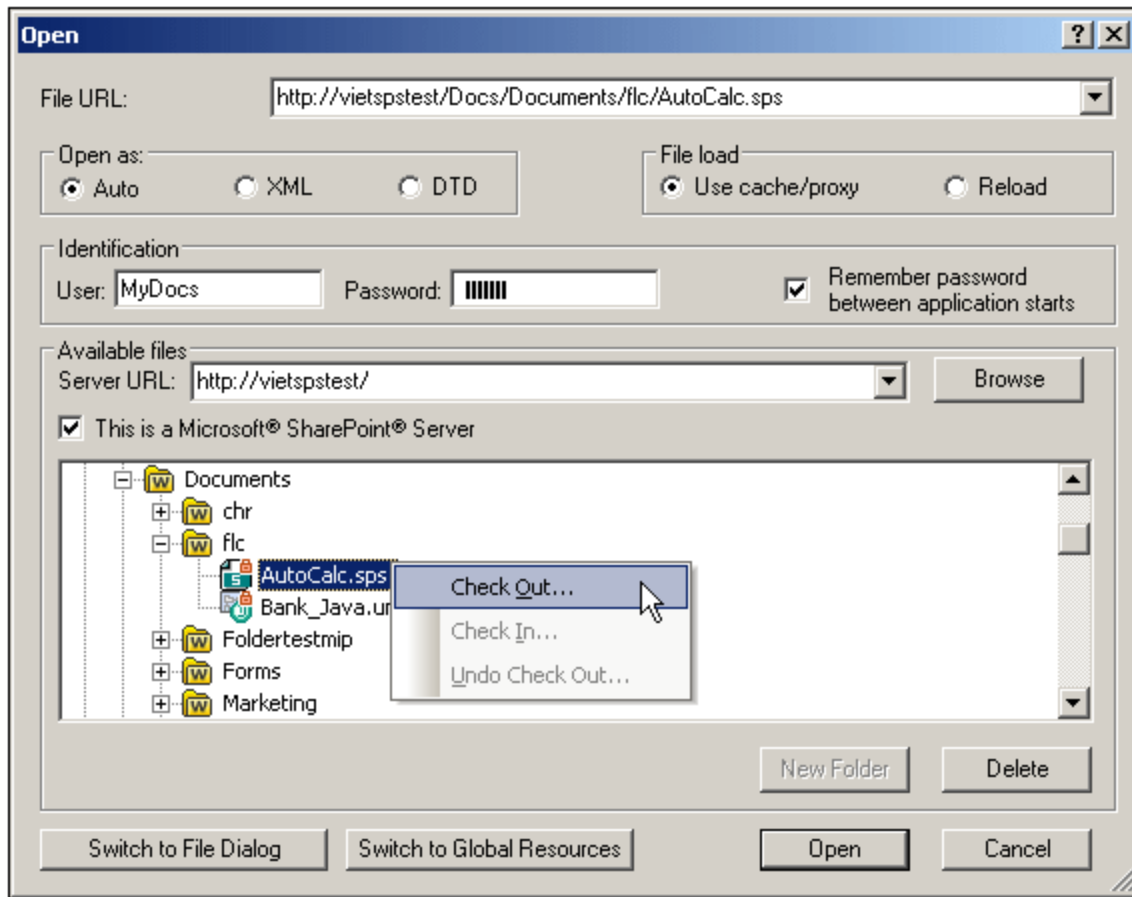
Note: The Browse function is only available on servers which support WebDAV and on Microsoft SharePoint Servers. The supported protocols are FTP, HTTP, and HTTPS.

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:

	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 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](#)⁴²⁷. For a general description of Global Resources, see the [Global Resources](#)⁴¹⁶ section in this documentation.

16.4.2 Open, Reload, Close, Close All

The **Open (Ctrl+O)** command  allows you to open an existing SPS or PXF file. The familiar [Open dialog](#)⁷¹⁶ of Windows systems is opened and allows you to select a file with an extension of `.sps`.

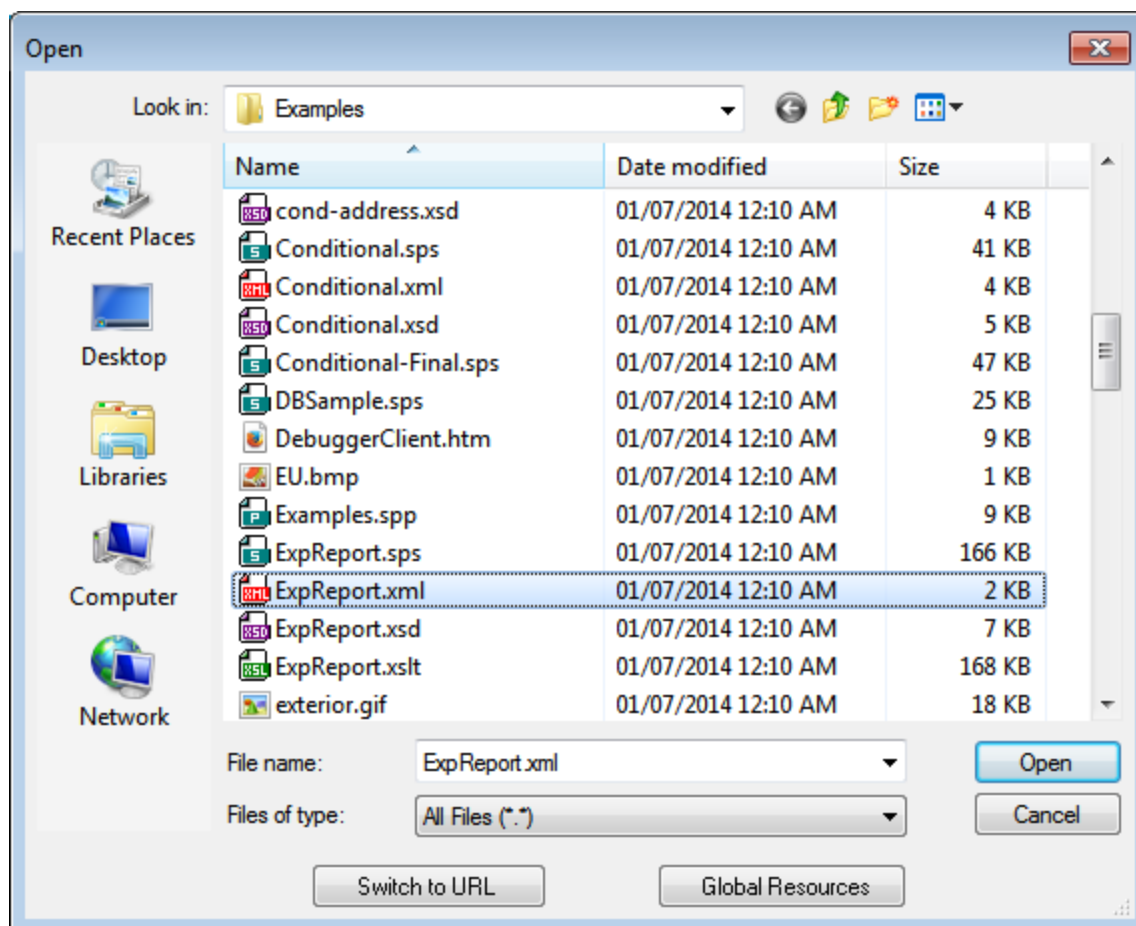
The **Reload** command reloads the SPS file from the file saved to disk. Any changes made since the file was last saved will be lost. The Working XML file will also be reloaded, enabling you to update the Working XML File if it has been changed externally.

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](#)³². 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

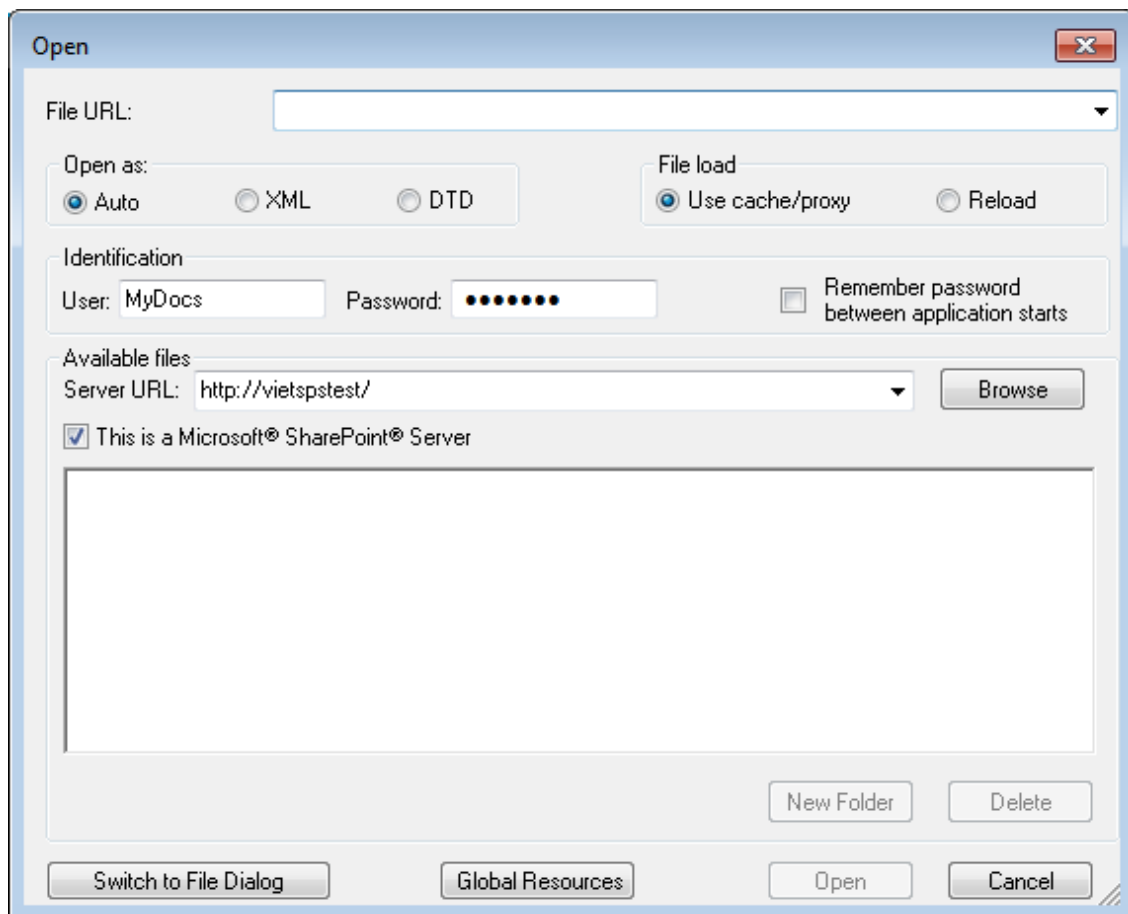
In several File Open and File Save dialogs, you can choose to select the required file or save a file via a URL or a global resource (*see screenshot below*). Click **Switch to URL** or **Global Resource** to go to one of these selection processes.



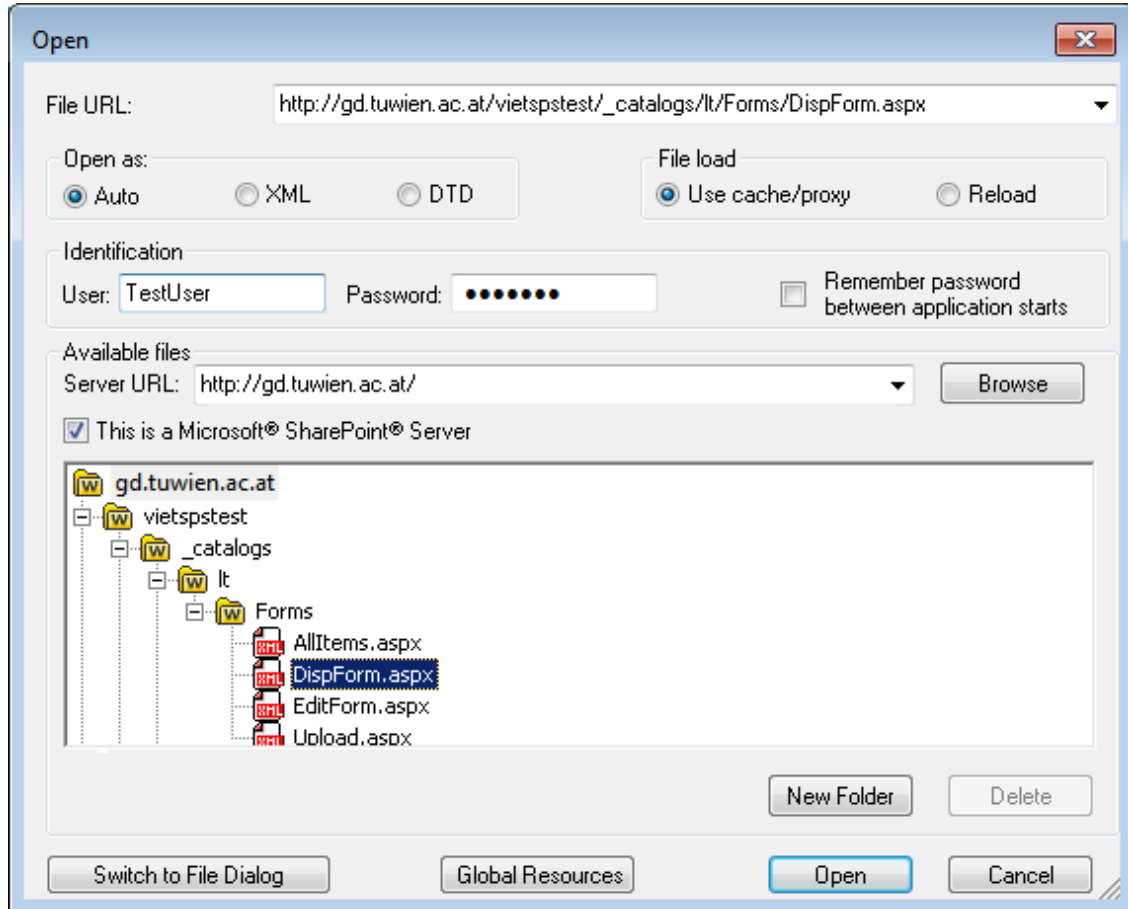
Selecting files via URLs

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.

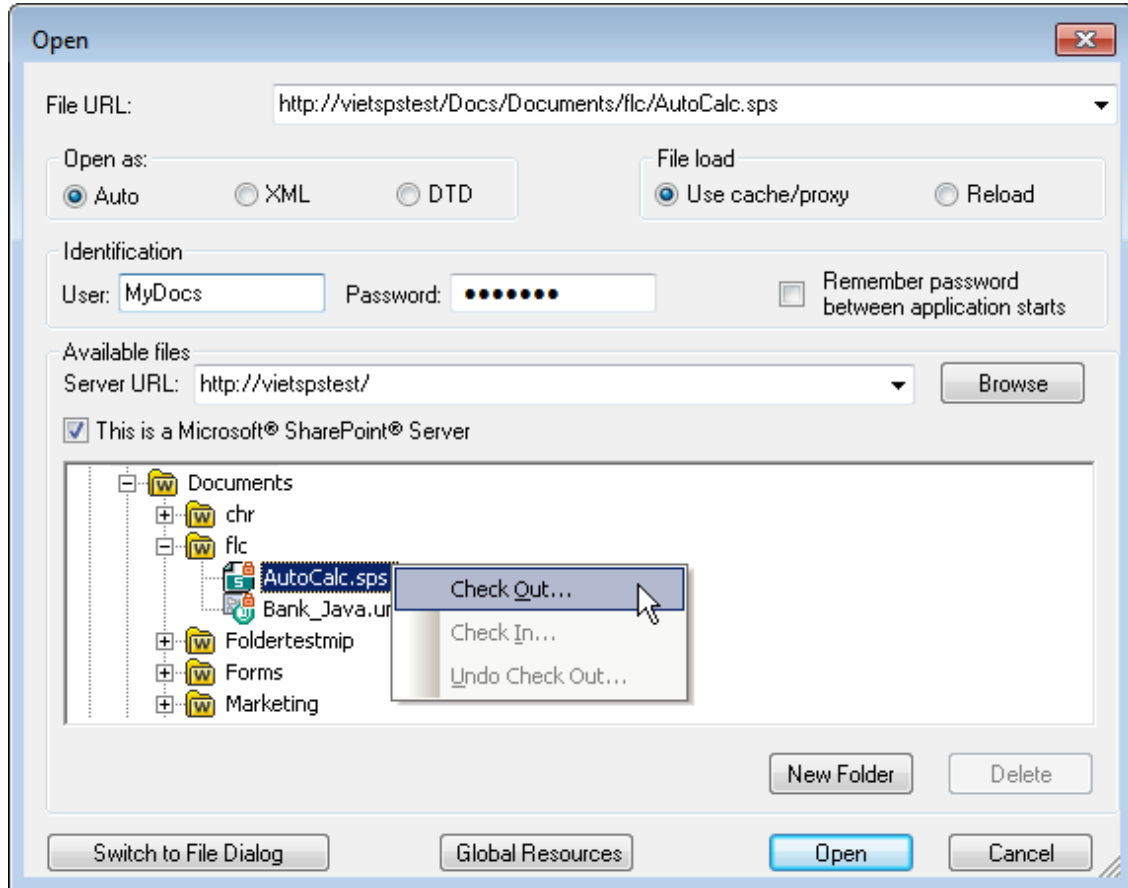
Note the following:

- 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.

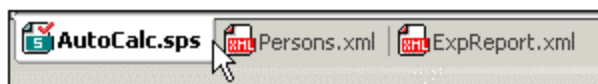


Right-clicking a file pops up a context menu containing commands available for that file (screenshot above).

- 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 **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](#)⁴²⁷. For a general description of Global Resources, see the [Global Resources](#)⁴¹⁶ section in this documentation.

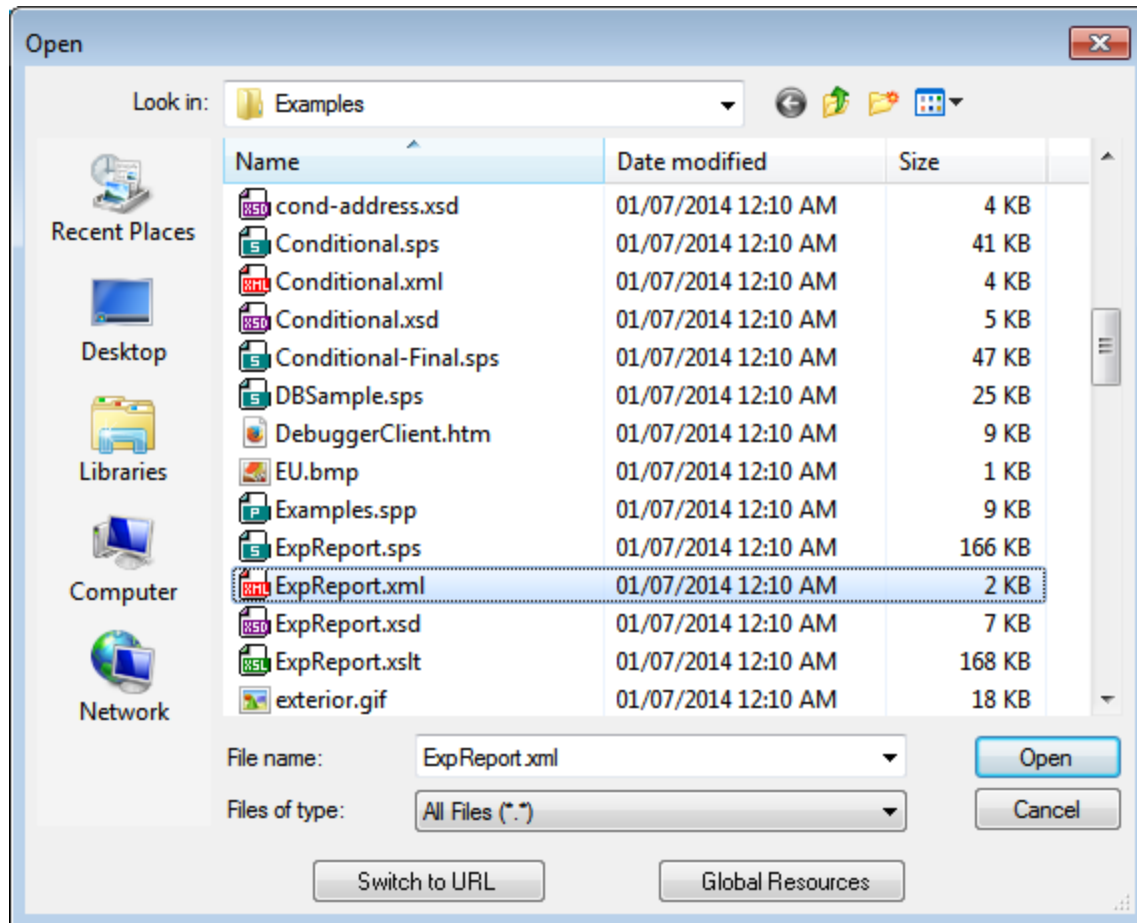
16.4.3 Save Design, Save All

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

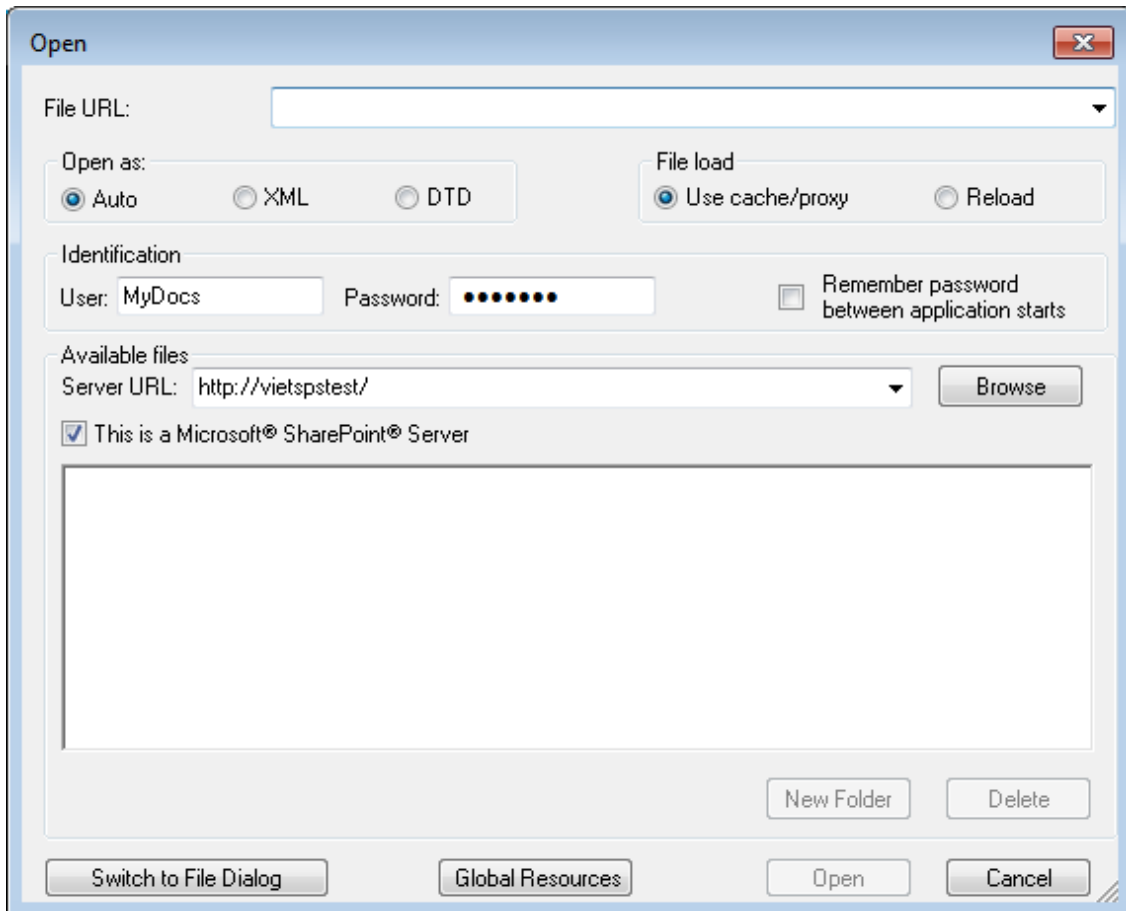
In several File Open and File Save dialogs, you can choose to select the required file or save a file via a URL or a global resource (*see screenshot below*). Click **Switch to URL** or **Global Resource** to go to one of these selection processes.



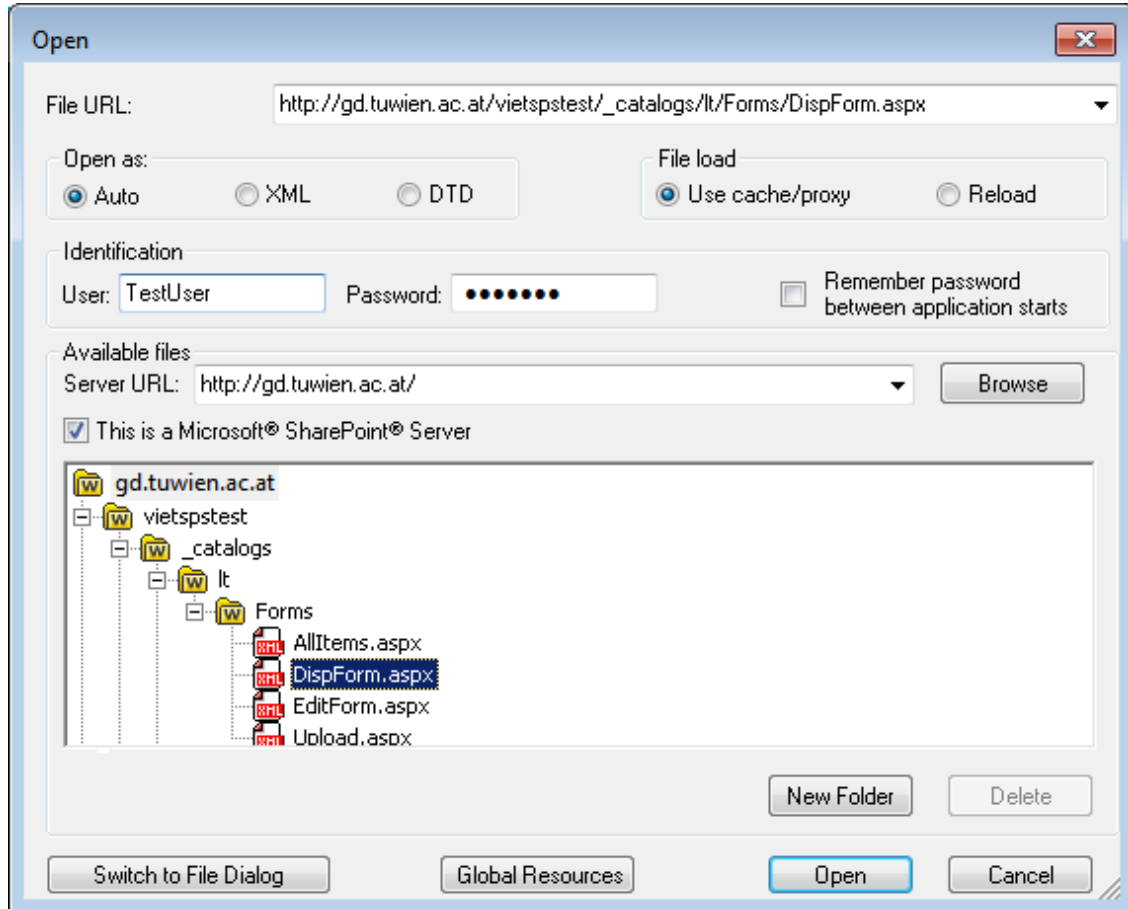
Selecting files via URLs

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.

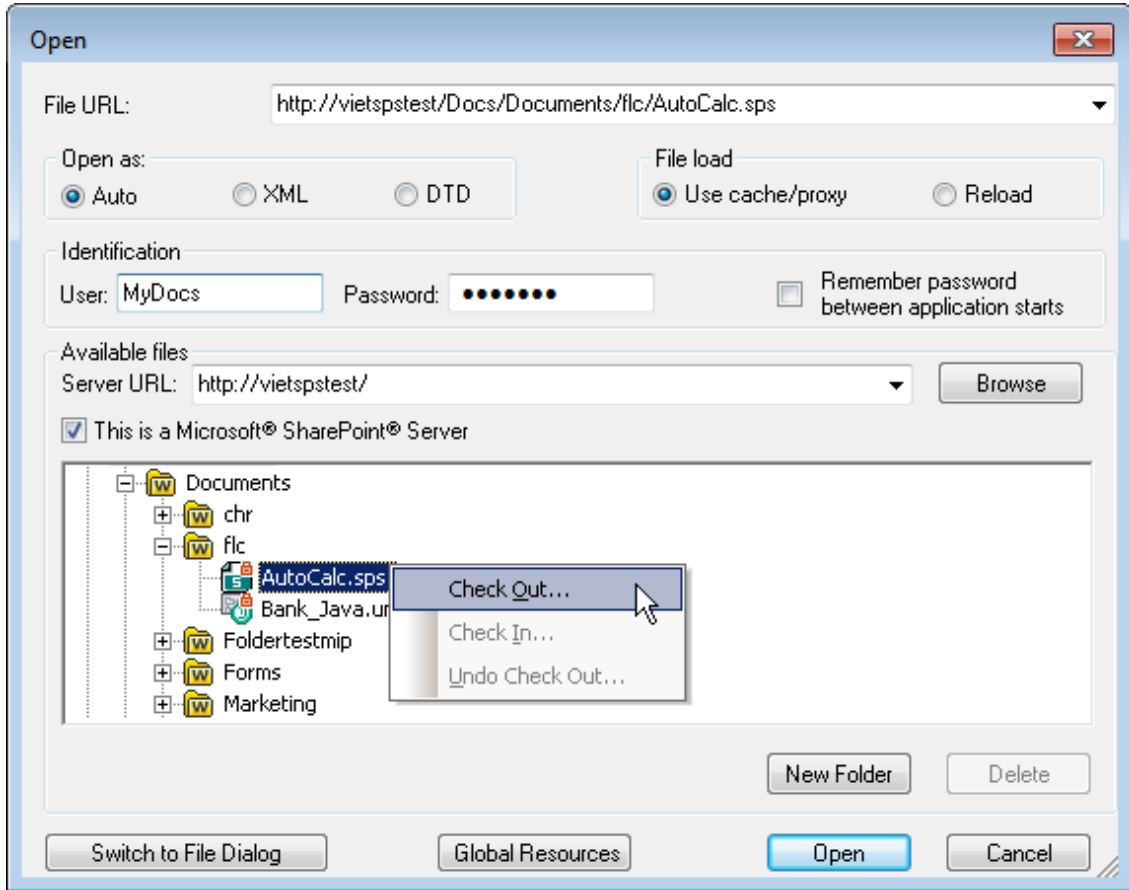
Note the following:

- 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.

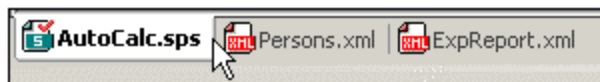


Right-clicking a file pops up a context menu containing commands available for that file (screenshot above).

- 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

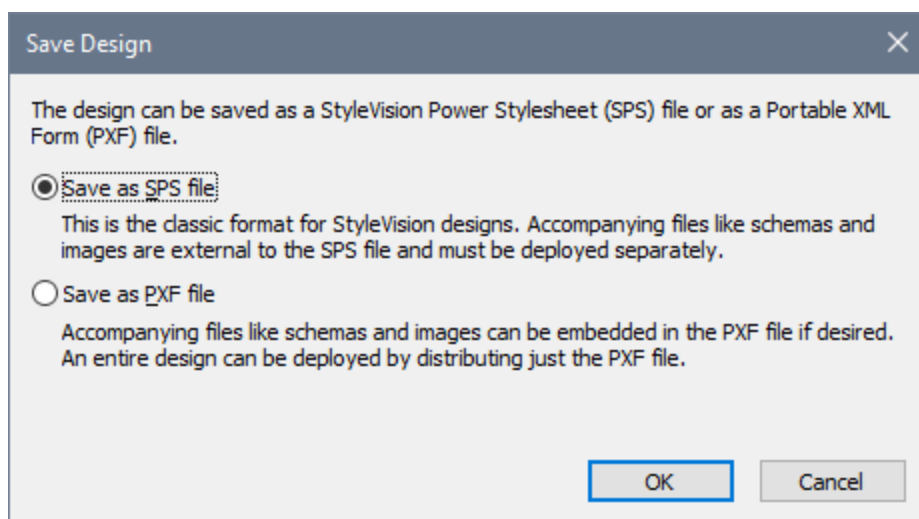
- command in the context menu. This has the effect of returning the file unchanged to the server.
- If you check out a file in one Altova application, you cannot check it out in another Altova application. The file is considered to be already checked out to you. The available commands at this point in 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 **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](#)⁴²⁷. For a general description of Global Resources, see the [Global Resources](#)⁴¹⁶ section in this documentation.

16.4.4 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**.



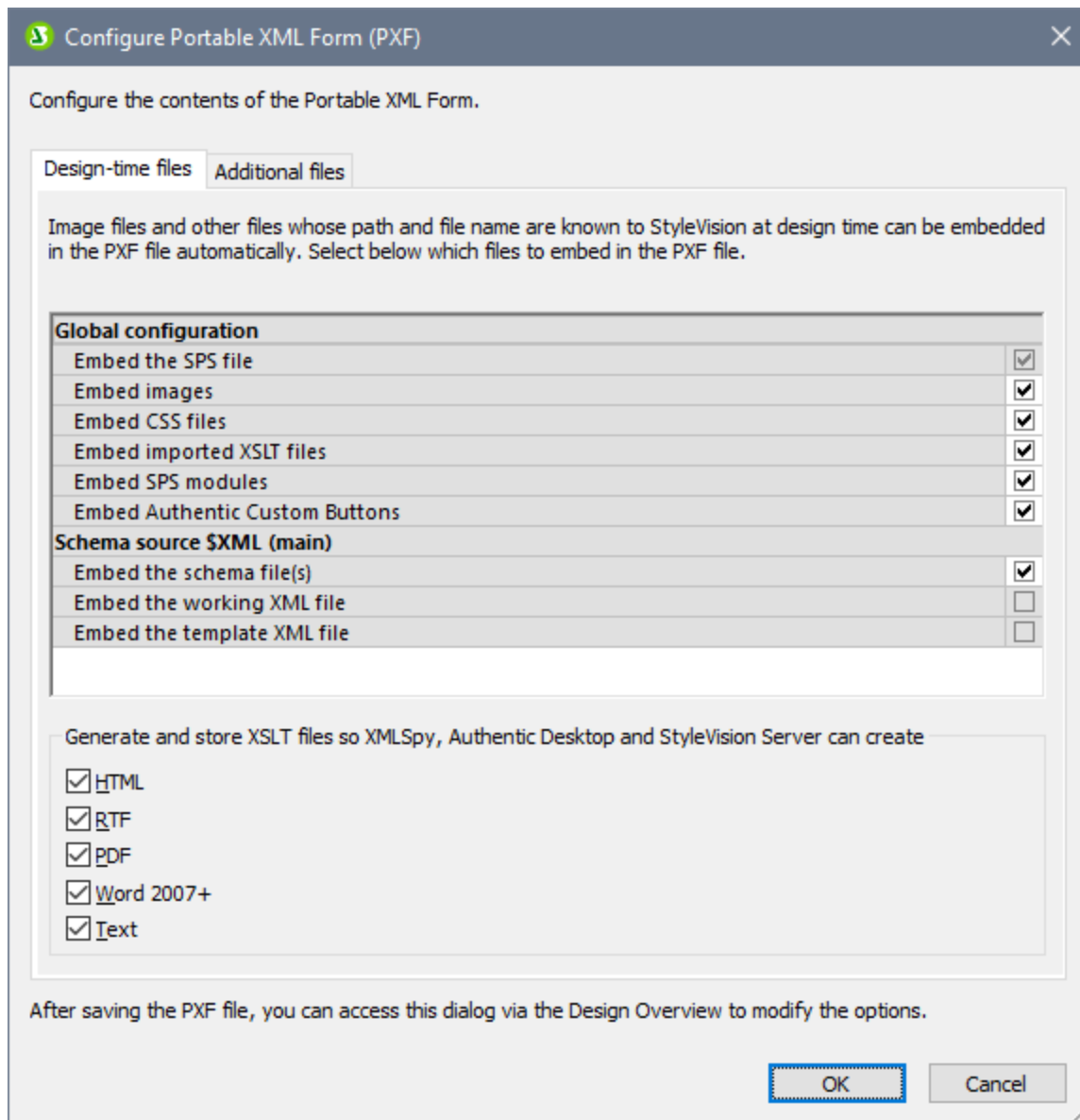
The **SPS format** is the standard Altova format for StyleVision designs. The **PXF format** is an Altova format that allows all files related to the design (schema files, XML files, images files, generated XSLT stylesheets, etc) to be embedded with the design. This format is very useful for transporting all the files required to open the design in Authentic View and/or to generate HTML, Text, and/or RTF 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](#)⁷²⁵ [command](#)⁷²⁵. 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](#) ⁷²⁵ [command](#) ⁷²⁵—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.

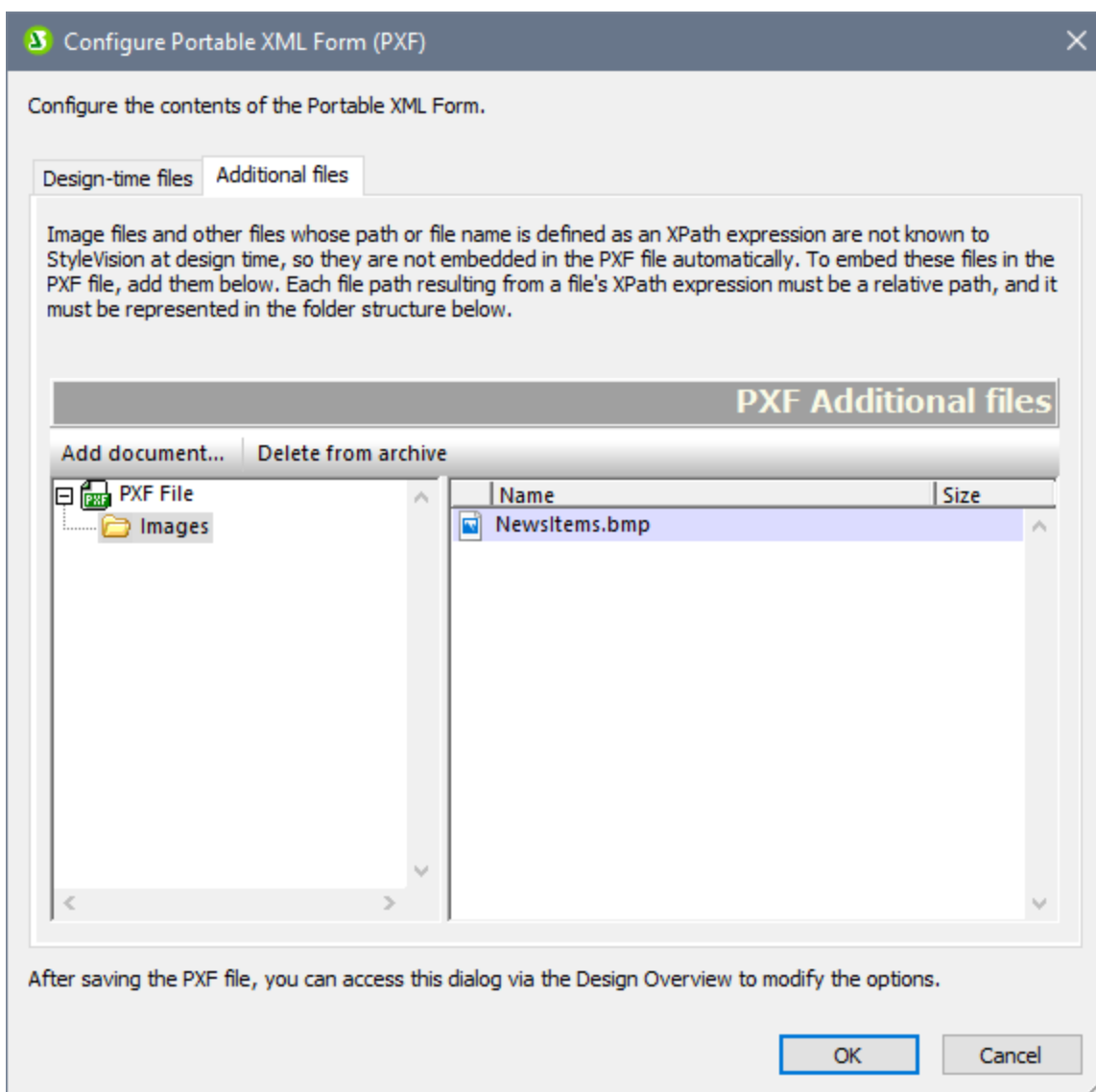


In the Global Configuration pane of the *Design-time Files* tab, you can select/deselect the design-related source files to be embedded/omitted. You can additionally choose to embed XSLT files generated from the design. In the XSLT files pane, select the output formats for which you wish to generate 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*).



Note: If XSLT files for outputs supported only in a higher edition of StyleVision (*high to low: Enterprise, Professional, Basic*) were created in a PXF file and if that PXF file is then opened in a lower edition, then on saving the PXF file the XSLT files for outputs not supported in the lower edition will not be saved. A prompt appears, asking whether you wish to continue saving the PXF file. You can then save without the unsupported formats, or abort the save and retain the unsupported formats.

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.

16.4.5 Export as MobileTogether Design File

This command generates an Altova MobileTogether design file from the active SPS design. A MobileTogether design file is used to execute solutions in the MobileTogether app for mobile devices. For example, a MobileTogether solution can be opened in a mobile device, such as a smartphone, to view and edit the contents of a database. A MobileTogether solution is designed in Altova MobileTogether Designer. This command enables you to convert an SPS design into a MobileTogether design that can be edited in MobileTogether Designer. For more information, see the [MobileTogether web page](#) and [MobileTogether Designer documentation](#). Conversions options are available in the MobileTogether Design tab of the Options dialog ([Tools | Options](#)⁸³⁹).

Note: Not all SPS design features have correspondences in MobileTogether designs. After running this command, you should therefore open the generated file in MobileTogether Designer to review it and, if necessary, correct it. The following design features are known not to be exported to MobileTogether designs: (i) Sources beyond the first one listed in the StyleVision design (an Enterprise feature); (ii) [global templates](#)²⁴⁴; (ii) [modules](#)²³⁰.

16.4.6 Save Authentic XML Data, Save As

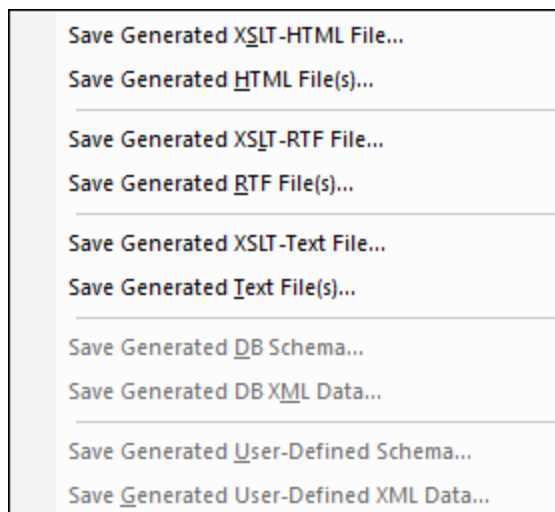


In Authentic View, you can edit the Working XML File or DB related to the SPS. The **Save Authentic XML Data** command saves these modifications to the Working XML File or DB. Alternatively to editing the XML file in StyleVision, you can edit an XML document or DB in the Authentic View of Altova XMLSpy or Altova Authentic Desktop.

The **Save Authentic XML Data As** command enables you to save the Authentic XML document as another file.

16.4.7 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](#)¹¹¹.



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](#)²⁶¹ 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](#)⁵⁸⁶); 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](#)²⁶¹ 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](#)⁵⁸⁶); 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-Text File

The Save Generated XSLT-Text File command generates an XSLT file for Text output from your SPS. You can use this XSLT file subsequently to transform an XML document to a text document.

Save Generated Text File(s)

The Save Generated Text File(s) command generates a text file. Multiple text files will be generated if [multiple document output](#)²⁶¹ 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](#)⁵⁸⁶); you do not assign a Working XML File.
- An XSLT-for-Text file, which is automatically generated from the currently active SPS file when this command is executed.

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.

16.4.8 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.
- 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*).

Enter the host name and port of a FlowForce Administration Interface to deploy the current transformation.

Server: Port:

User: Use SSL

Password:

Login:

Deploy As

Path:

The path must start with a slash character.

Save design changes before deploying

Open web browser to create new job

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.
- To deploy the mapping through a SSL-encrypted connection, select the **Use SSL** check box. This assumes that FlowForce Server is already configured to accept SSL connections. For more information, refer to FlowForce Server documentation.

On clicking **OK**, the `.transformation` file is deployed to the FlowForce Server at the location specified. If you have checked 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.

Multiple versions of StyleVision Server

If the server where you deploy the `.transformation` file has multiple versions of StyleVision Server running under FlowForce Server management (applicable to Windows servers only), then a *Select StyleVision Server* dialog appears, in which you are prompted to specify the version of StyleVision Server with which you want this mapping to be executed. You can select the version you want to use manually, or let the server select the most suitable version automatically.

This dialog appears when the FlowForce Server installation directory contains a `.tool` file for each StyleVision Server version which runs under FlowForce Server management. By default, a StyleVision Server `.tool` file is

added automatically to this directory when you install StyleVision Server as part of a FlowForce Server installation. The path where the `.tool` files are stored in FlowForce is: `C:\Program Files\Altova\FlowForceServer2024\tools`. If you have additional versions of StyleVision Server which you want to run under FlowForce Server management, their `.tool` files may need to be copied manually to the directory above. The `.tool` file of StyleVision Server can be found at: `C:\Program Files\Altova\StyleVisionServer2024\etc`.

Note: For information about how to work with FlowForce Server, see the [FlowForce documentation](#).

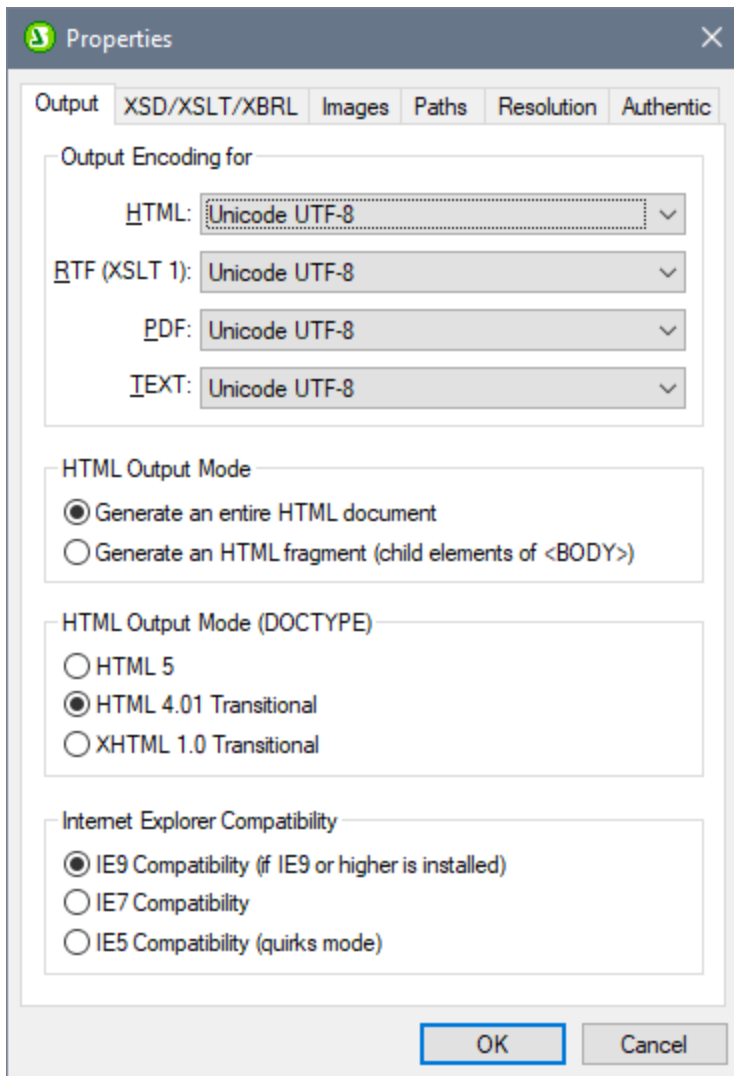
16.4.9 Web Design

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](#)⁴⁷⁹.

16.4.10 Properties

The **Properties** command pops up the Properties dialog (*screenshot below shows the dialog in the Enterprise Edition*), in which you can set various properties for the active SPS.



Output

The following properties can be set in the Output tab:

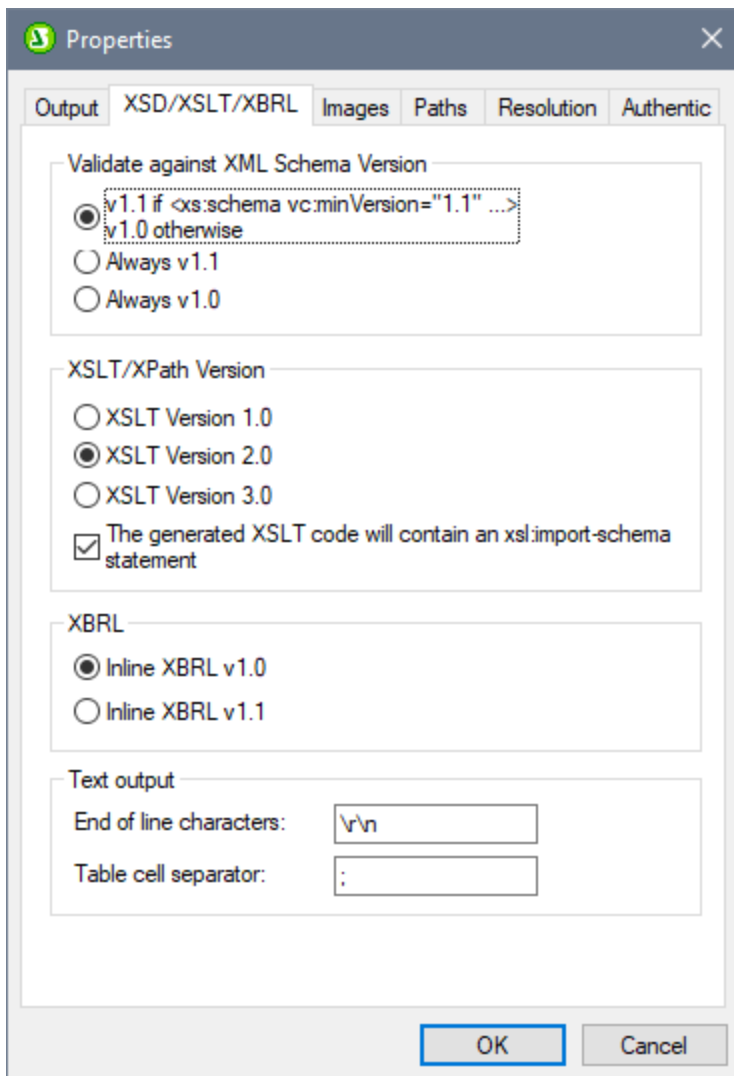
- **Output Encoding:** In the Output Encoding pane you can select the encoding of your output documents. Changing the encoding in this dialog changes the encoding for the currently active SPS. You can also specify the application-wide [default encoding](#) ⁶³⁹ for all SPS documents in the Encoding tab 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..
- **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](#) ⁷³⁸ dialog).

XSD/XSLT

In this tab (*screenshot below shows the dialog in the Enterprise Edition*), 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 `/xs:schema/@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.

Select the XSLT version for the active document in this tab. Checking the option about the *xsl:import-schema statement* 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.

For Text output, you can specify the following:

- The end-of-line characters to use. The default is `\r\n` (which indicates the carriage-return and linefeed characters on Windows systems).
- The character to indicate the end of a table cell. The default is the semicolon character.

Images

This tab provides settings for selecting the following:

- In the RTF output: Whether images are linked to or embedded.
- In HTML output: Whether images (currently chart, barcode and inline images) are embedded by means of the Data URI scheme.

These settings are made for each SPS individually, and they are available when the XSLT version of the SPS is either 2.0 or 3.0.

Embedding images in RTF output

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 or 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.

If the *Embed Images* check box is not checked, images will be linked according to the image file path specified in the image's properties (select the image and select URL in the [Properties entry helper](#)⁵⁵). For information about how paths are resolved, see the section [Image URIs](#)¹⁶⁸.

This setting must be selected in order to save [PXF files](#)⁴⁸².

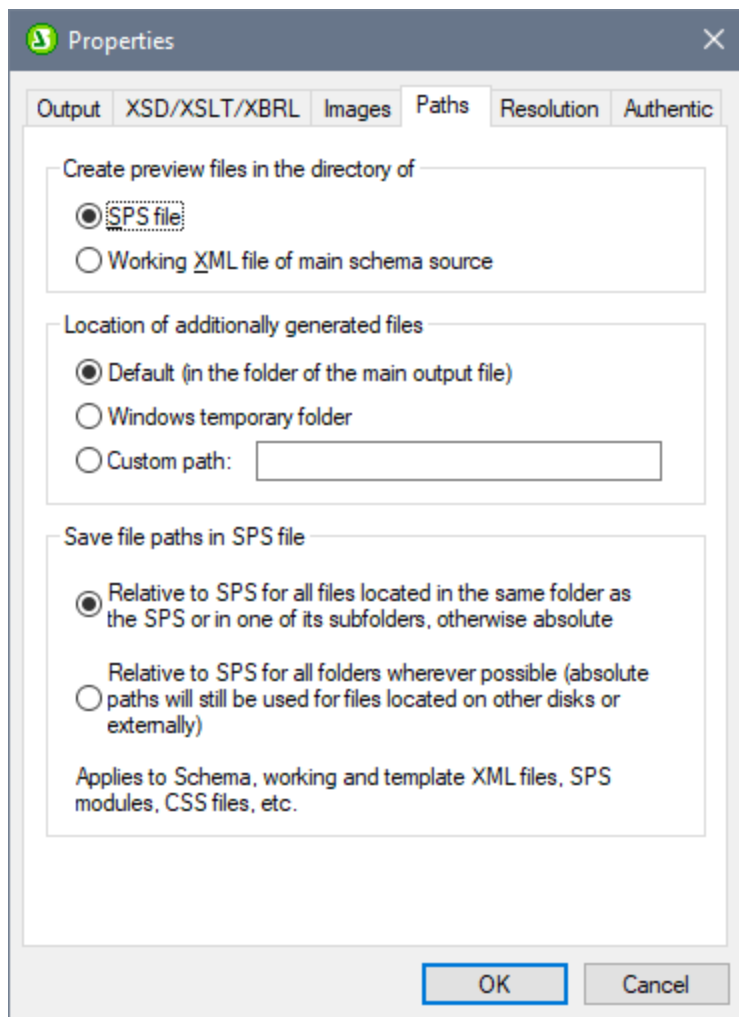
Note: The RTF format supports embedded images only for EMF, JPG, and PNG files.

Embedding images in HTML output

Chart, barcode, and inline images can be embedded in HTML output. If this checkbox is selected, the image data is converted to a Base64 string that is stored in the `src` attribute of the HTML `img` element using the Data URI scheme. This kind of embedding is available only when the XSLT version of the SPS is 2.0 or 3.0.

Paths

Default paths for various files created by the SPS file and for paths saved in the SPS file are specified in the settings of this tab (*screenshot below shows the dialog in the Enterprise Edition*).



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](#)⁴⁰⁹.

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.

16.4.11 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.

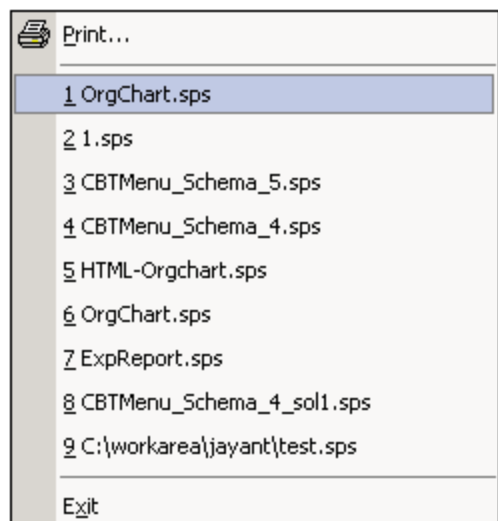
To navigate the pages of the preview, use the page navigation buttons at the bottom of the preview or by entering the page number in the Page text-box.

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**.

16.4.12 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.


16.5 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](#) ⁷⁴⁵, to undo or restore your previous actions, and to select all content of the SPS.
- [Find, Find Next, Replace](#) ⁷⁴⁵, to find text in the SPS, Authentic View, and XSLT stylesheet previews, and to replace text in Authentic View.
- [Stylesheet Parameters](#) ⁷⁵⁰, to edit parameters declared globally for the SPS.
- [Collapse/Expand Markup](#) ⁷⁵¹, 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.


16.5.1 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.

16.5.2 Find, Find Next, Replace

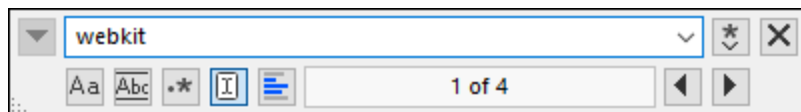
The **Find (Ctrl+F)** command  enables you to find text strings in Design View, JavaScript Editor, Authentic View, and XSLT stylesheets.

The Find & Replace dialog that is displayed depends on which view is currently active.

- When Design View is active, clicking the **Find** or **Replace** command will set the focus to the [Find & Replace sidebar](#) ⁶¹. The search and replace functionality of Design View is described in the topic [Find & Replace sidebar](#) ⁶¹.
- The search and replace functionality in JavaScript Editor, Authentic View, and XSLT stylesheets is described in this topic.

XSLT stylesheets and JavaScript Editor

Clicking the **Find** command in the XSLT-for-HTML, XSLT-for-RTF or JavaScript Editor tab displays the following dialog:



You can select from the following options:

- *Match case*: Case-sensitive search when toggled on (`Address` is not the same as `address`).
- *Match whole word*: Only the exact words in the text will be matched. For example, for the input string `fit`, with *Match whole word* toggled on, only the word `fit` will match the search string; the `fit` in `fitness`, for example, will not be matched.
- *Regular expression*: If toggled on, the search term will be read as a regular expression. See [Regular expressions](#)⁷⁴⁷ below for a description of how regular expressions are used.
- *Find anchor*: When a search term is entered, the matches in the document are highlighted and one of these matches will be marked as the current selection. The *Find anchor* toggle determines whether that first current selection is made relative to the cursor position or not. If *Find anchor* is toggled on, then the first currently selected match will be the next match from the current cursor location. If *Find anchor* is toggled off, then the first currently selected match will be the first match in the document, starting from the top.
- *Find in selection*: When toggled on, locks the current text selection and restricts the search to the selection. Otherwise, the entire document is searched. Before selecting a new range of text, unlock the currently selection by toggling off the *Find in Selection* option.
- *Replace*: In the JavaScript Editor tab, click the Down Arrow button, which is located at the top left of the dialog, to open the *Replace* field. Here you can enter the string that you want to substitute for the found string.


HTML Preview and Authentic View

Clicking the **Find** command in HTML Preview or Authentic View opens a simple [Find and Replace](#)⁶¹ dialog.

Note the following:

- 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`.
- In Authentic View, the dynamic data (XML data) is searched, not text from the static (XSLT) input.
- The **Replace** functionality in Authentic View is straightforward, enabling you to replace a chosen text string with another. Available options are: (i) the matching of whole words only (not words in the text that contain the searched-for string), and (ii) the matching of words that have the same casing as the searched-for string.

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](#)⁷⁴⁵ for a description of how to use the search function.

Using regular expressions

You can use regular expressions (regex) to find a text string. To do this, first, switch the *Regular expression* option on (see above). This specifies that the text in the search term field is to be evaluated as a regular expression. Next, enter the regular expression in the search term field. For help with building a regular expression, click the **Regular Expression Builder** button, which is located to the right of the search term field. Click an item in the Builder to enter the corresponding regex metacharacter/s in the search term field. See below for a brief description of metacharacters.

Regular expression metacharacters

Given below is a list of regular expression metacharacters.

.	Matches any character. This is a placeholder for a single character.
(Marks the start of a tagged expression.
)	Marks the end of a tagged expression.
(abc)	The (and) metacharacters mark the start and end of a tagged expression. Tagged expressions may be useful when you need to tag ("remember") a matched region for the purpose of referring to it later (back-reference). Up to nine expressions can be tagged (and then back-referenced later, either in the Find or Replace field). For example, (the) \1 matches the string the the. This expression can be literally explained as follows: match the string "the" (and remember it as a tagged region), followed by a space character, followed by a back-reference to the tagged region matched previously.
\n	Where n is a variable that can take integer values from 1 through 9. The expression refers to the first through ninth tagged region when replacing. For example, if the find string is Fred([1-9])XXX and the replace string is Sam\1YYY, this means that in the find string there is one tagged expression that is (implicitly) indexed with the number 1; in the replace string, the tagged expression is referenced with \1. If the find-replace command is applied to Fred2XXX, it would generate Sam2YYY.
\<	Matches the start of a word.
\>	Matches the end of a word.
\x	Allows you to use a character x, which would otherwise have a special meaning. For example, \[would be interpreted as [and not as the start of a character set.
[...]	Indicates a <i>set of characters</i> . For example, [abc] means any of the characters a, b or c. You can also use ranges: for example [a-z] for any lower case character.
[^...]	The complement of the characters in the set. For example, [^A-Za-z] means any character except an alphabetic character.
^	Matches the start of a line (unless used inside a set, see above).
\$	Matches the end of a line. Example: A+\$ to find one or more A's at end of line.
*	Matches 0 or more times. For example, Sa*m matches Sm, Sam, Saam, Saaam and so on.
+	Matches 1 or more times. For example, Sa+m matches Sam, Saam, Saaam and so on.

Representation of special characters

Note the following expressions.

<code>\r</code>	Carriage Return (CR). You can use either CR (<code>\r</code>) or LF (<code>\n</code>) to find or create a new line
<code>\n</code>	Line Feed (LF). You can use either CR (<code>\r</code>) or LF (<code>\n</code>) to find or create a new line
<code>\t</code>	Tab character
<code>\\</code>	Use this to escape characters that appear in regex expression, for example: <code>\\n</code>

Regular expression examples

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


Example 1: Renaming elements

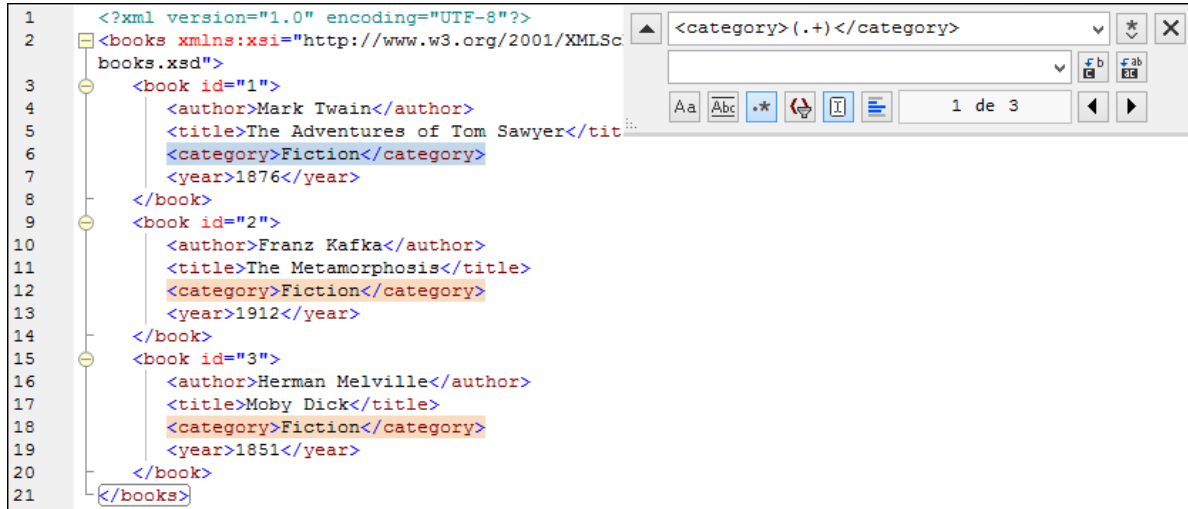
The sample XML code listing below contains a list of books. Let's suppose your goal is to replace the `<Category>` element of each book to `<Genre>`. One of the ways to achieve this goal is by using regular expressions.

```
<?xml version="1.0" encoding="UTF-8"?>
<books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="books.xsd">
  <book id="1">
    <author>Mark Twain</author>
    <title>The Adventures of Tom Sawyer</title>
    <category>Fiction</category>
    <year>1876</year>
  </book>
  <book id="2">
    <author>Franz Kafka</author>
    <title>The Metamorphosis</title>
    <category>Fiction</category>
    <year>1912</year>
  </book>
  <book id="3">
    <author>Herman Melville</author>
    <title>Moby Dick</title>
    <category>Fiction</category>
    <year>1851</year>
  </book>
</books>
```

To solve the requirement, follow the steps below:

1. Press **Ctrl+H** to open the Find and Replace dialog box.

2. Click **Use regular expressions** .
3. In the Find field, enter the following text: `<category>(.)</category>`. This regular expression matches all `category` elements, and they become highlighted.




```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <books xmlns:xsi="http://www.w3.org/2001/XMLSchema
books.xsd">
3      <book id="1">
4          <author>Mark Twain</author>
5          <title>The Adventures of Tom Sawyer</title>
6          <category>Fiction</category>
7          <year>1876</year>
8      </book>
9      <book id="2">
10         <author>Franz Kafka</author>
11         <title>The Metamorphosis</title>
12         <category>Fiction</category>
13         <year>1912</year>
14     </book>
15     <book id="3">
16         <author>Herman Melville</author>
17         <title>Moby Dick</title>
18         <category>Fiction</category>
19         <year>1851</year>
20     </book>
21 </books>

```

To match the inner text of each element (which is not known in advance), we used the tagged expression `(.)`. The tagged expression `(.)` means "match one or more occurrences of any character, that is `.`, and remember this match". As shown in the next step, we will need the reference to the tagged expression later.

4. In the Replace field, enter the following text: `<genre>\1</genre>`. This regular expression defines the replacement text. Notice it uses a back-reference `\1` to the previously tagged expression from the Find field. In other words, `\1` in this context means "the inner text of the currently matched `<category>` element".
5. Click **Replace All**  and observe the results. All `category` elements have now been renamed to `genre`, which was the intended goal.

Example 2: Changing the order of attributes


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

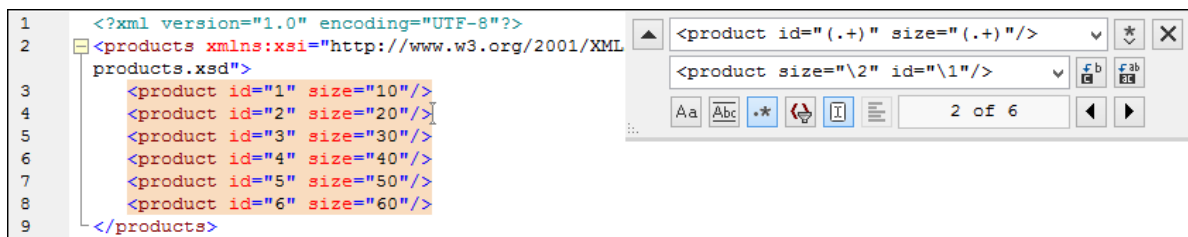
```


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

```


To solve the requirement, follow the steps below:

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



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

16.5.3 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.



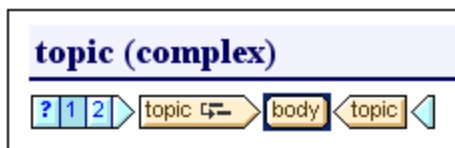
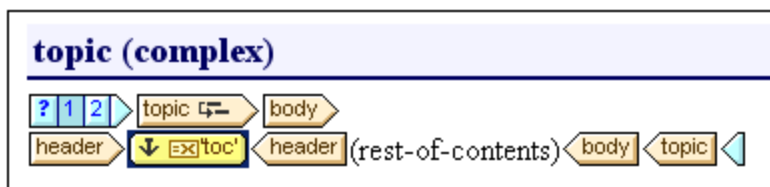
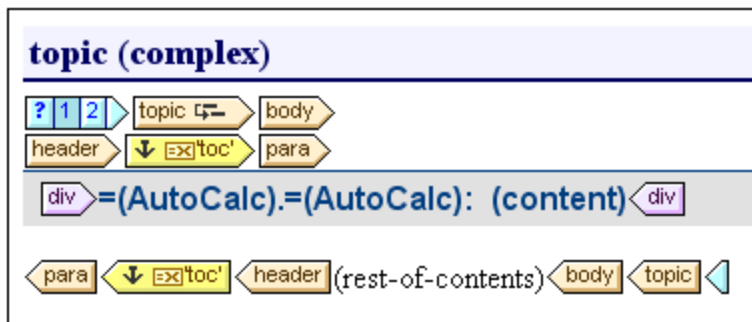
The following points should be noted:

- 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](#) ⁴².

16.5.4 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.

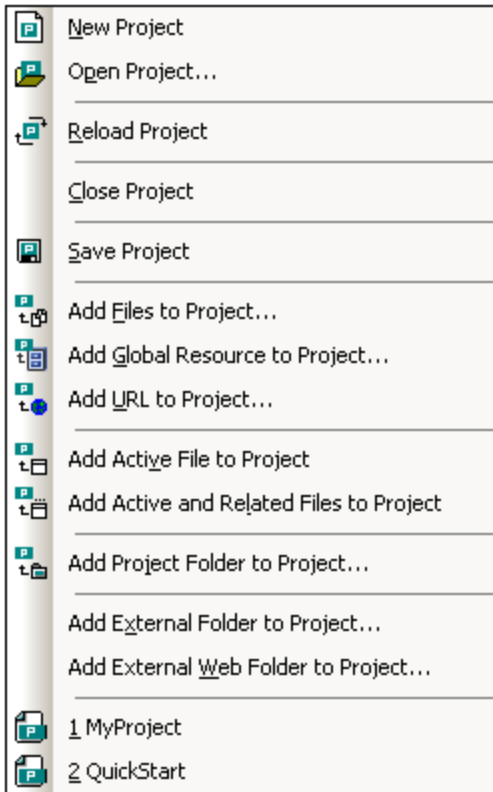
The screenshots below show how a series of tags are collapsed. Double-clicking a collapsed tag expands it.



Collapsing a tag can be useful for optimizing the display according to your editing needs.

16.6 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.



The **Project** menu contains the following commands, which are explained in detail in the sub-sections of this section:

- [New Project](#)⁷⁵⁴, for creating a new project
- [Open Project](#)⁷⁵⁴, for opening an existing project
- [Reload Project](#)⁷⁵⁴, for refreshing a project in the Projects sidebar
- [Close Project](#)⁷⁵⁴, for closing a project in the Project sidebar
- [Save Project](#)⁷⁵⁴, for saving and naming a project
- [Add Files to Project](#)⁷⁵⁵, for adding files to a project in the Project sidebar
- [Add Global Resource to Project](#)⁷⁵⁵, for adding Altova Global Resources to a project in the Project sidebar
- [Add URL to Project](#)⁷⁵⁵, for adding a file via a URL to a project in the Project sidebar
- [Add Active File to Project](#)⁷⁵⁶, for adding the currently active SPS file to a project
- [Add Active and Related Files to Project](#)⁷⁵⁶, for adding the currently active SPS file and its related files
- [Add Project Folder to Project](#)⁷⁵⁶, for adding folders to a project in the Project sidebar
- [Add External Folder to Project](#)⁷⁵⁶, for adding local folders to a project in the Project sidebar
- [Add External Web Folder to Project](#)⁷⁵⁶, for adding folders via URL to a project in the Project sidebar


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.

16.6.1 New Project, Open Project, Reload Project

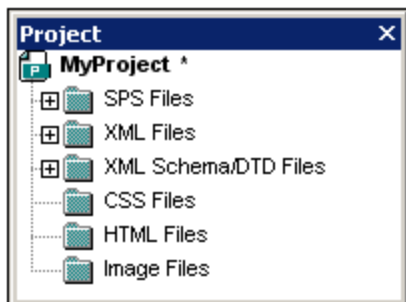
The **New Project** command  creates a new project. The new project replaces the previous project (if any) in the Projects sidebar. If the project you have been working on has unsaved changes, a prompt appears asking whether you wish to save changes to the project. Note that the **New Project** command only creates a new project without saving it; you have to explicitly save the project using the [Save Project](#) ⁷⁵⁴ 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.

16.6.2 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*).



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.

16.6.3 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.

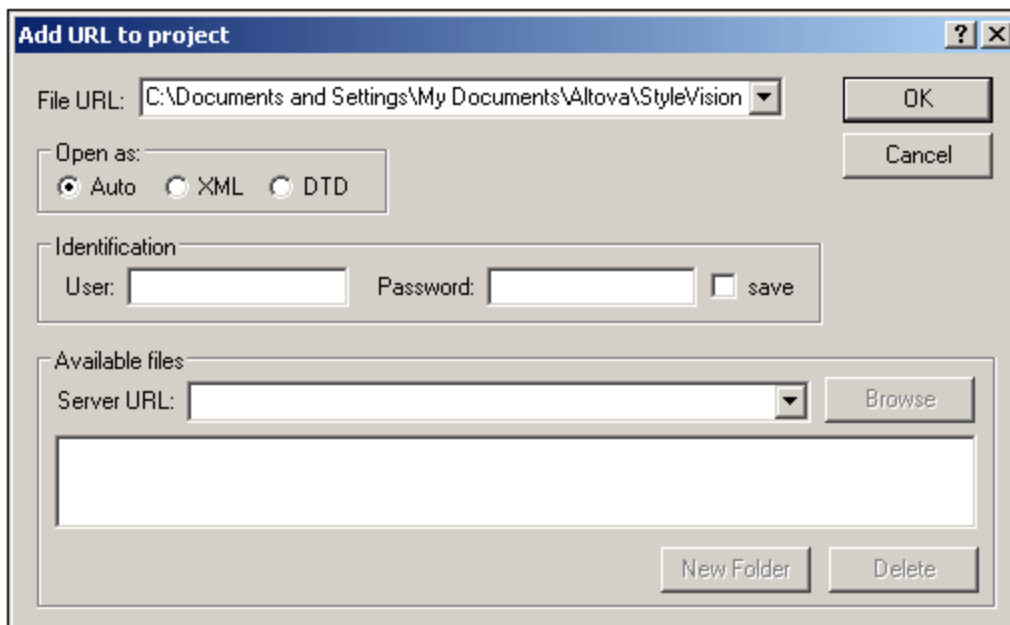
To add a file to a folder or sub-folder within the main project folder, right-click that folder, select the command **Add Files**, and then browse for the required file/s.

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**.

16.6.4 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

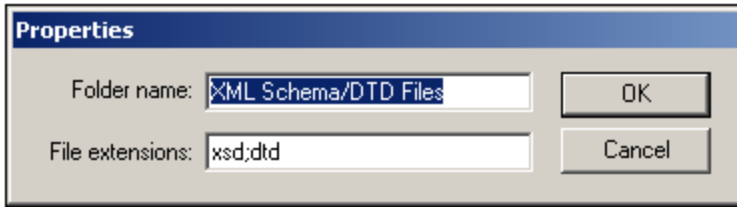
To delete a file, select the file in the Project sidebar, right-click, and, from the context menu, select **Delete**.

16.6.5 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.



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

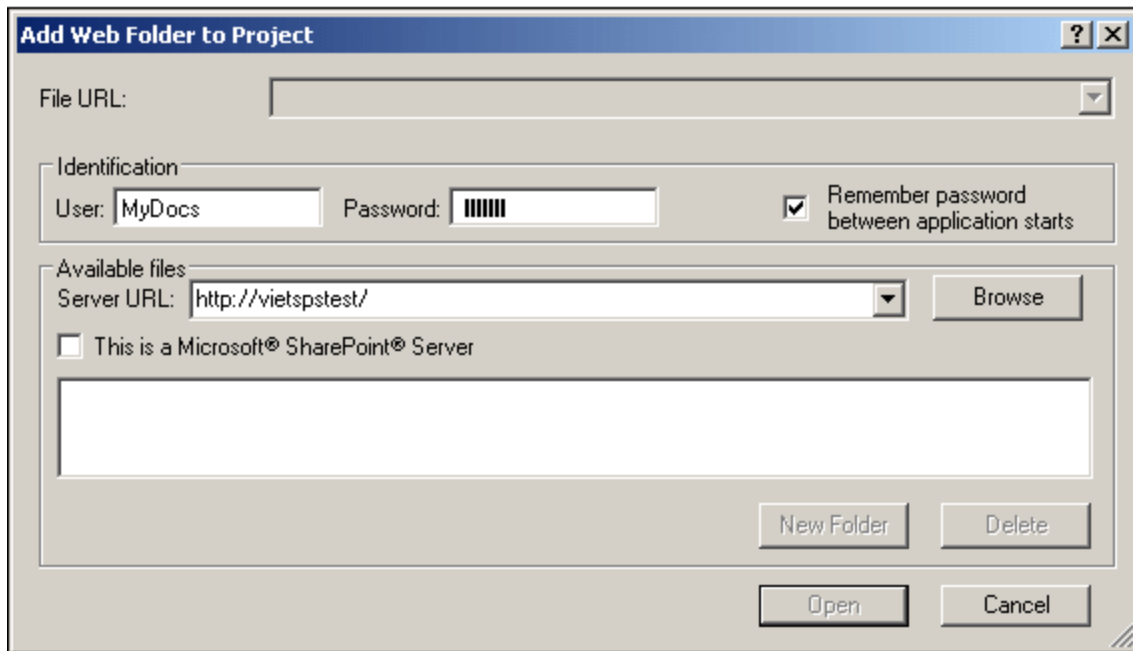
The **Add External Folder to Project** command adds a new external folder to the current project. The command adds a local or network folder, with all its contents, to the current project. The added external folder can be expanded and collapsed. To add an external folder to a project folder as a sub-folder, right-click the project folder and, from the context menu, select the command **Add External Folder**.

Add External Web Folder to Project

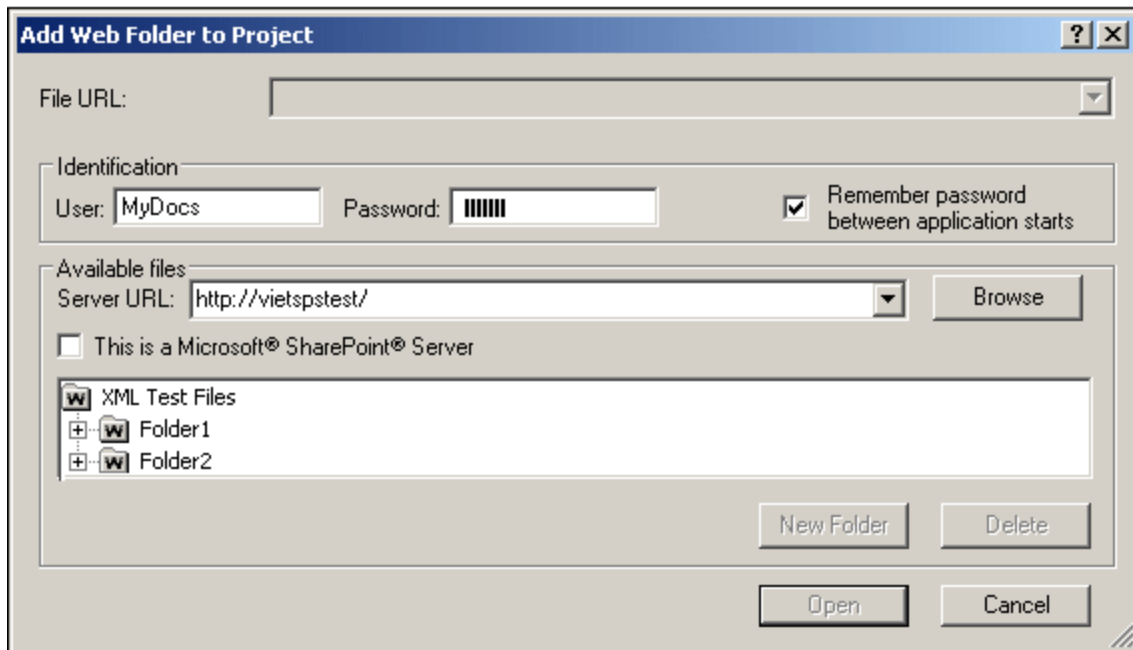
The **Add External Web Folder to Project** command adds a new external folder via a URL to the current project. The added external folder can be expanded and collapsed. To add an external web folder to a project folder as a sub-folder, right-click the project folder and, from the context menu, select 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

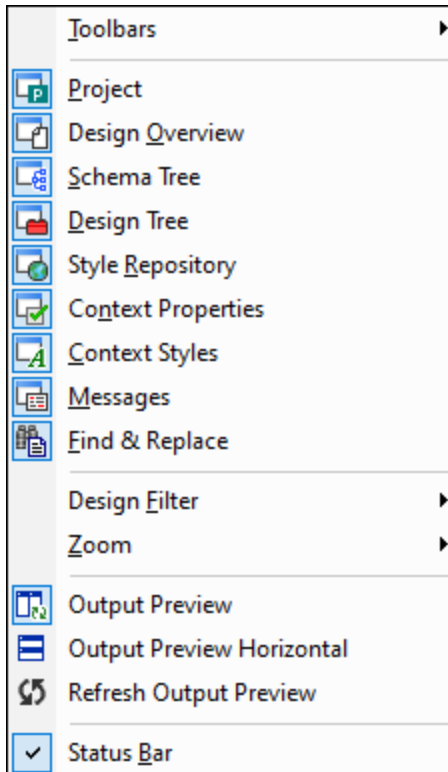
In the Project window, a folder can be dragged to another folder or to another location within the same folder. Additionally, folders can be dragged from Windows File Explorer to the Project window.

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**.

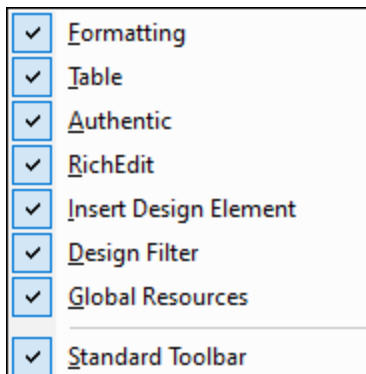
16.7 View Menu

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.



16.7.1 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.



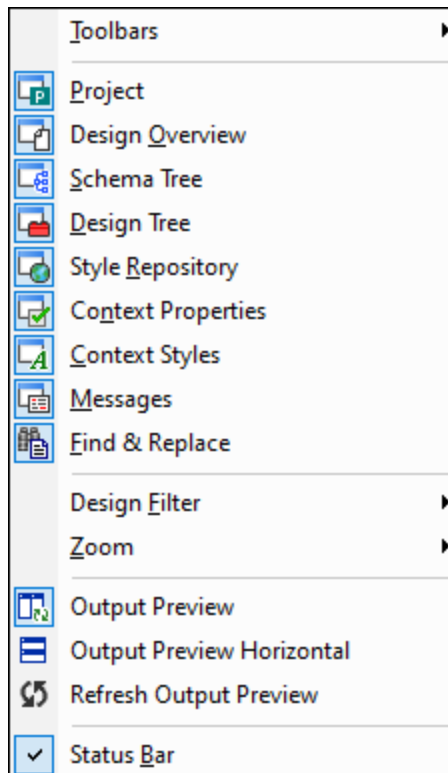
When a toolbar is checked, it is displayed. In the screenshot above all the toolbars are displayed. To toggle on or off the display of a toolbar, click the appropriate toolbar. For a complete description of toolbars, see the section [Reference | Toolbars](#)⁷⁰².

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.

16.7.2 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.

16.7.3 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:

Icon	Command	Description
	Show only one template	Shows the selected template only. Place the cursor in a template and click to show that template only.
	Show all template types	Shows all templates in the SPS (main, global, named, and layout) .
	Show imported templates	Toggles the display of imported templates on and off.
	Show/Hide main template	Toggles the display of the main template on and off.
	Show/Hide global templates	Toggles the display of global templates on and off.
	Show/Hide Design Fragments	Toggles the display of Design Fragments on and off.

Note that these commands are also available as toolbar icons in the [Design Filters](#) ⁷¹¹ 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.

16.7.4 Output Previews

There are three Output Preview commands:

- **Output Preview**, which switches Output Preview on/off
- **Output Preview Horizontal**, which is enabled when Output Preview has been switched on. The command switches from a vertical split view to a horizontal split view. When the output preview becomes available in a horizontal split, the command automatically changes to **Output Preview Vertical**, which switches the split view back to a vertical split view.
- **Refresh Output Preview**, which immediately refresh the output preview with any changes in the design or Authentic View.

16.8 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](#)⁷⁰². 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.

16.8.1 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](#)⁷⁰⁸, 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.

16.8.2 Rest of Contents

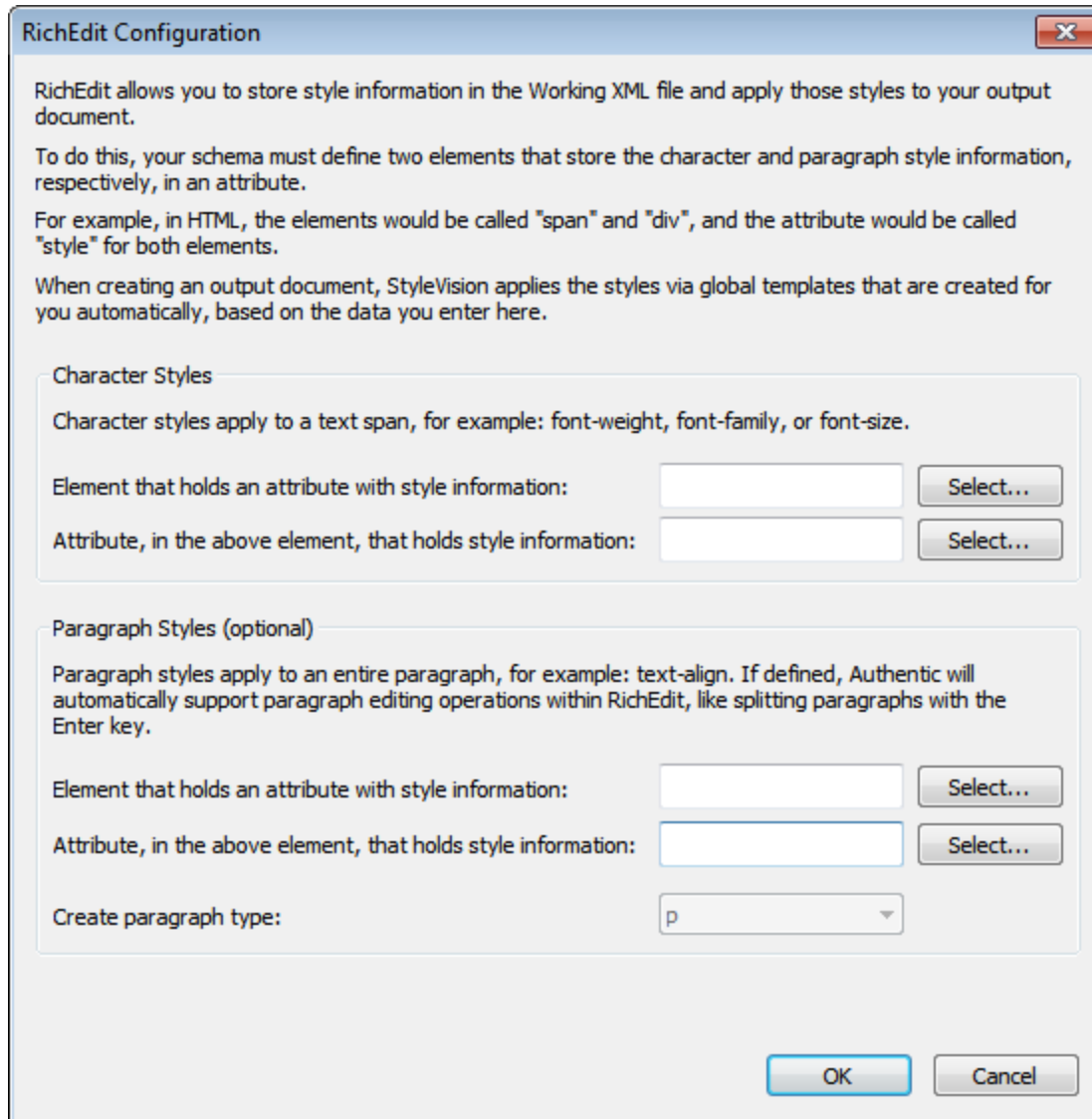
The **Rest of Contents** command inserts the `(rest-of-contents)` placeholder for that node. This placeholder represents the content of **unused child nodes** of the current node; it corresponds to the `xsl:apply-templates` rule of XSLT applied to the unused elements and text nodes of the current element. 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.

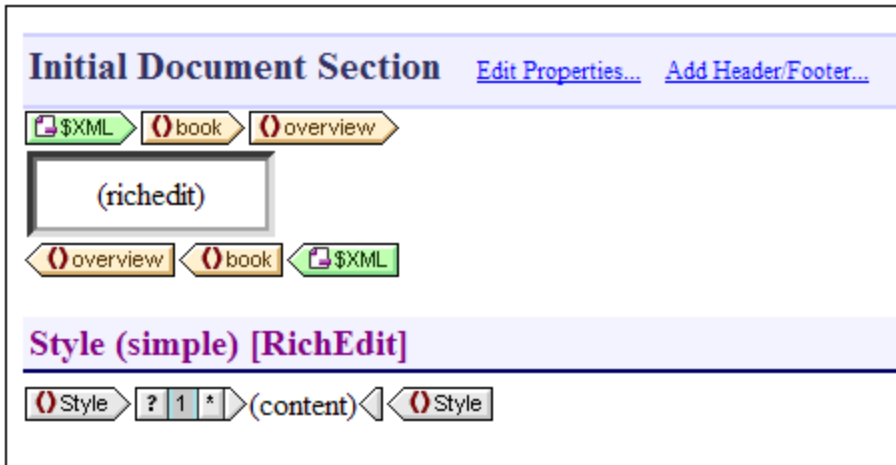
The `(rest-of-contents)` placeholder can be deleted by selecting it and pressing the **Delete** key on the keyboard.

16.8.3 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.



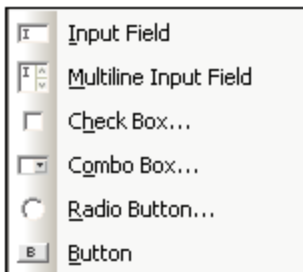
In the RichEdit Configuration dialog, enter the name of the styling element and its attribute that is to contain the RichEdit styling properties. You can also select the required element and attribute from the schema tree. Click the respective **Select** buttons to open the schema tree. When done, click **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](#) ³⁸¹.

16.8.4 Form Controls

Mousing over the **Form Controls** command rolls out a submenu (*screenshot below*) containing commands to insert various form controls ([data-entry devices](#) ¹⁷⁴).



How to create each of these form controls is described in the section [Using Data-Entry Devices](#) ¹⁷⁴. 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](#) ⁵⁵.

Form controls can also be inserted in the design by right-clicking at the insertion point and selecting **Insert | Form Controls**, or by clicking the respective Form Control icon in the [Insert Design Elements toolbar](#) ⁷⁰⁸, 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.

16.8.5 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.

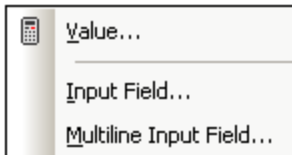
The list of commands is as follows.

- 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](#) ⁵⁹⁴.

16.8.6 Auto-Calculation

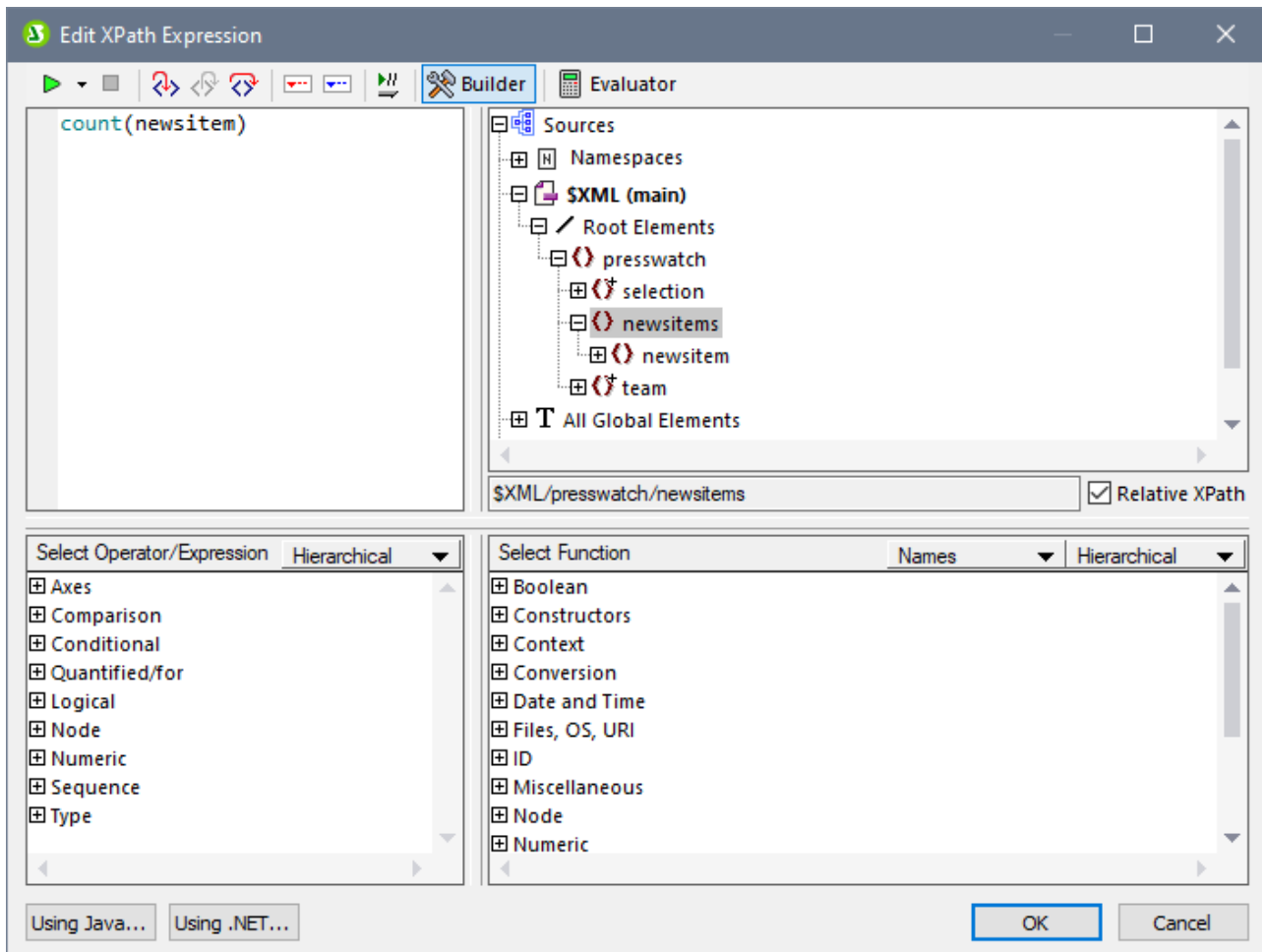
An **Auto-Calculation** uses an XPath expression to calculate a value. This value is displayed at the point where the Auto-Calculation is inserted. An Auto-Calculation can be inserted in the SPS as a text value, input field, or multiline input field. Place the cursor at the location where the Auto-Calculation 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](#) ⁷⁰⁸.



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](#) ⁶⁸⁵ (*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 *Relative XPath* check box is checked) or as an absolute expression starting from the document node (if the *Relative XPath* check box is unchecked).

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](#) ²⁷².

16.8.7 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.

16.8.8 Paragraph, Special Paragraph

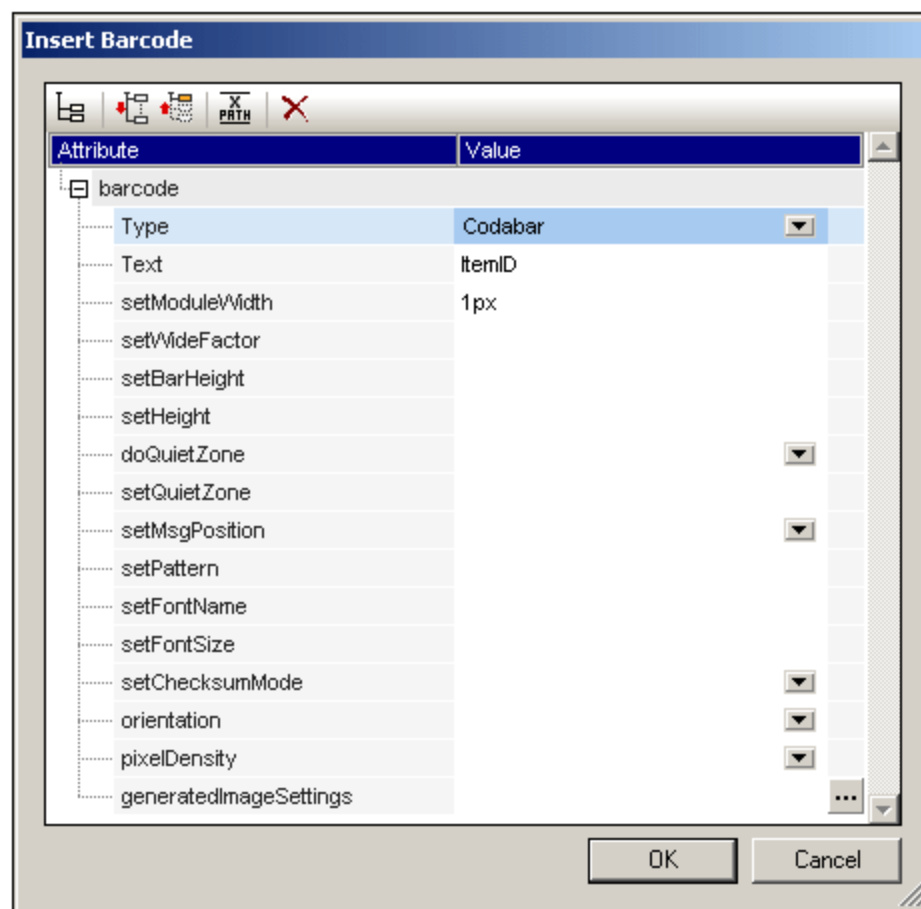
The **Paragraph** command  inserts an HTML paragraph `<p>` element around the selected component. A component is considered selected for this purpose when the entire node is selected (by clicking either of its tags) or when static text is selected. If the cursor is placed within static text, the paragraph 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](#) ⁷⁰⁸.

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.

16.8.9 Barcode

The **Insert Barcode** command pops up the Insert Barcode dialog (*screenshot below*).



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](#)¹⁸³.



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.

Important: For barcodes to be generated in the output, you **must use Altova's XSLT processor** to generate the output. This is because the barcodes in an SPS are generated by calling special Java extension functions that are not part of the XSLT standard. Altova's XSLT processors support these specific extension functions, whereas other XSLT processors very probably do not. As a result, barcodes will not be generated if processed with a non-Altova XSLT processor. The Altova XSLT processor is packaged with StyleVision, and is automatically called when you generate output via the **Generate** commands in the **File** menu. Alternatively, you can use [RaptorXML Server](#), which is a Altova's standalone XSLT processor.

16.8.10 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](#)⁷⁰⁸ also pops up the Insert Image dialog.

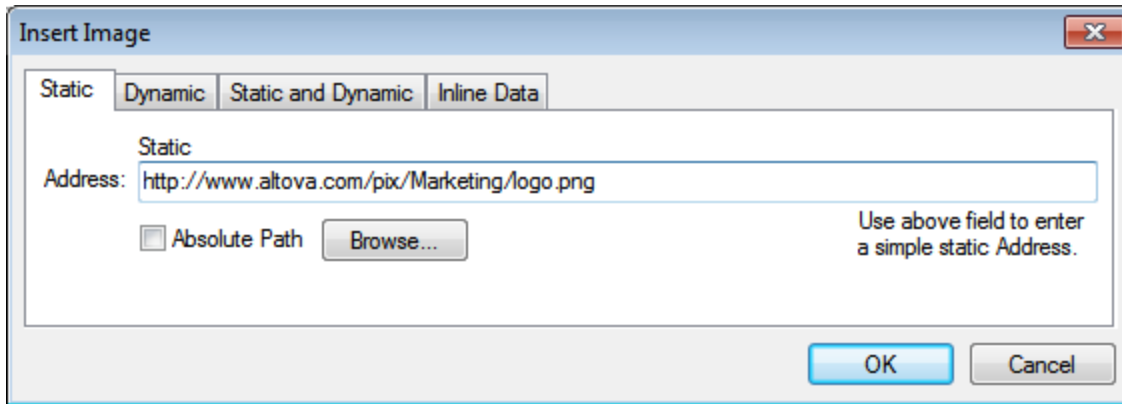
The Insert Image dialog has four tabs, each of which provides a different way to specify the image location. These are:

- *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

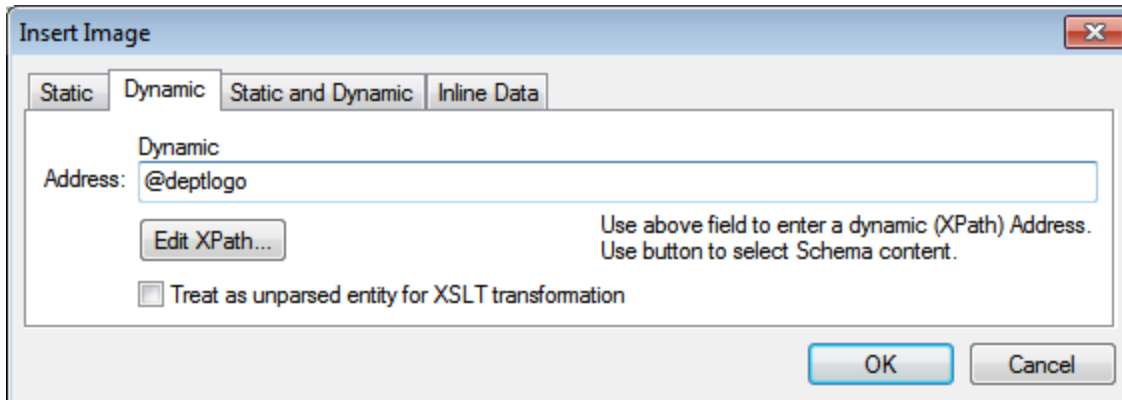
The image URI is entered directly in the *Address* field (see *screenshot below*). In the screenshot below the image URI is: `http://www.altova.com/pix/Marketing/logo.png`.



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.

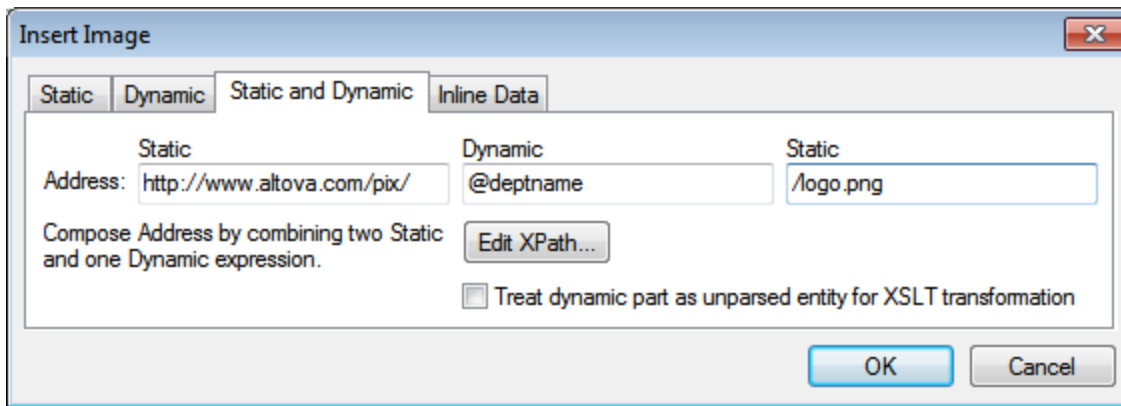


Click the **Edit XPath** button to pop up the [XPath Expression Builder](#)⁶⁹⁷. In the schema tree of the XPath Expression Builder, the context node will be highlighted.

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](#)⁴⁴⁰.

Static and Dynamic

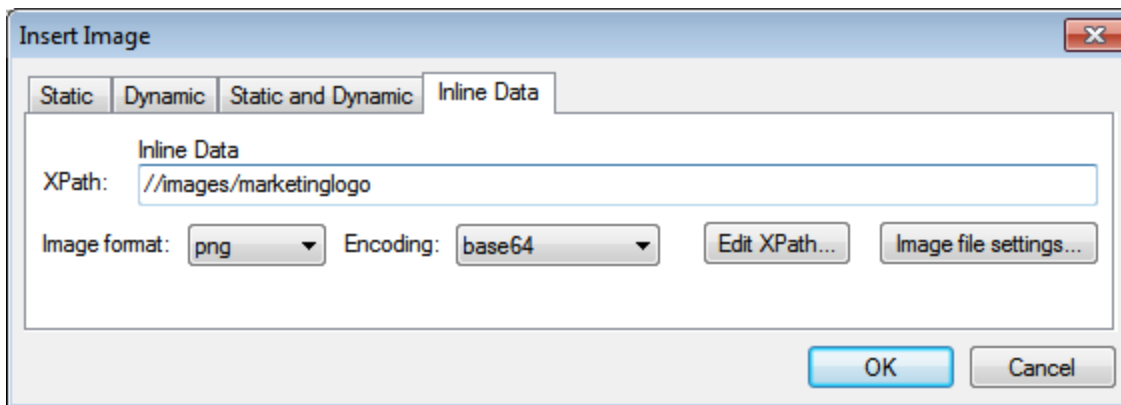
Use both the static and dynamic mechanisms together to generate the URI.



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](#)⁶⁹⁷ for the dynamic part.

Inline data

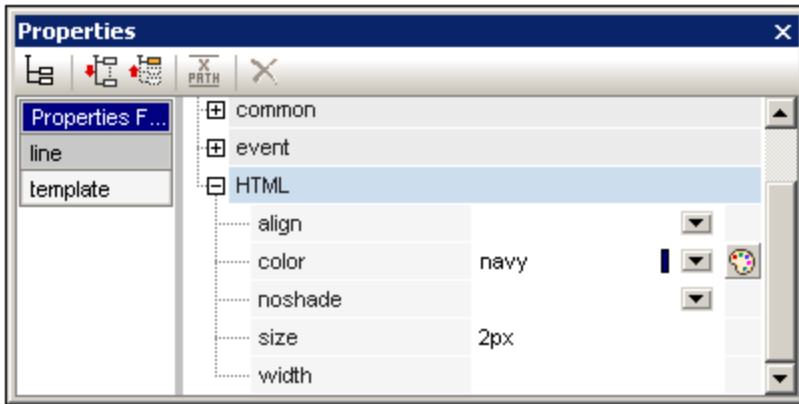
An image can be stored in an XML file as Base-16 or Base-64 encoded text. The XPath expression in the Insert Image dialog (see *screenshot below*) selects the node containing the encoded text. The Encoding combo box specifies the encoding used in the source XML so that StyleVision can correctly read 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.

16.8.11 Horizontal Line

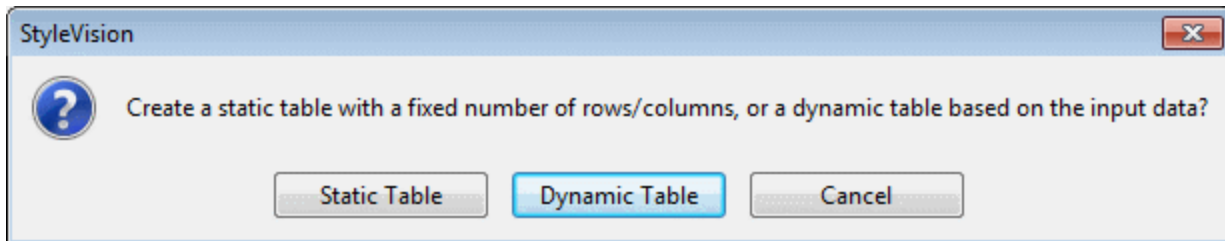
The **Horizontal Line** command inserts a horizontal line at the cursor insertion point. This command is not available when an SPS component is selected. To set properties for the horizontal line, select the line in the design, and in the Properties sidebar, select *line*, and specify values for properties in the *HTML* group (see *screenshot below*).



You can specify the following properties for the line: its `color`, `size` (thickness), `width` (in the design), `align`, and the `noshade` property.

16.8.12 Table

The **Insert Table** command pops up the Create Table dialog (*screenshot below*).



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](#)¹⁴⁵ and [Dynamic SPS Tables](#)¹⁴⁶.

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.

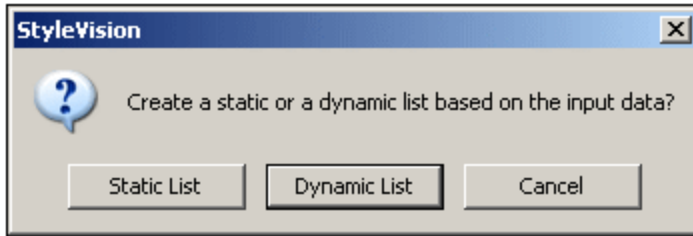
16.8.13 Bullets and Numbering



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.

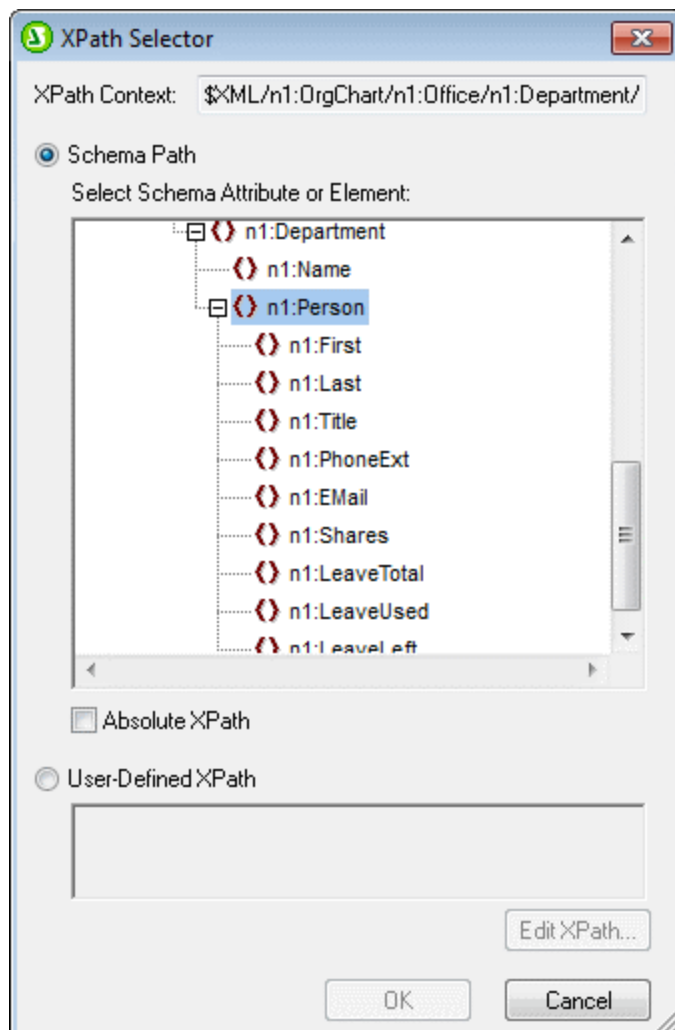
To create a list do the following:

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*).



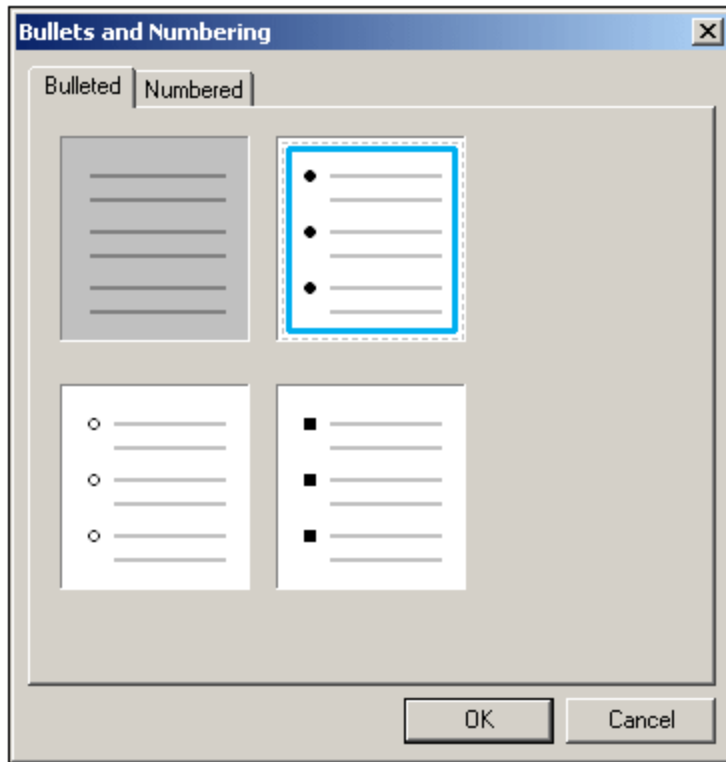
If you click **Static List**, the Bullets and Numbering dialog described in Step 3 pops up. If you click **Dynamic List**, the XPath Selector dialog pops up (*screenshot below*).

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 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.



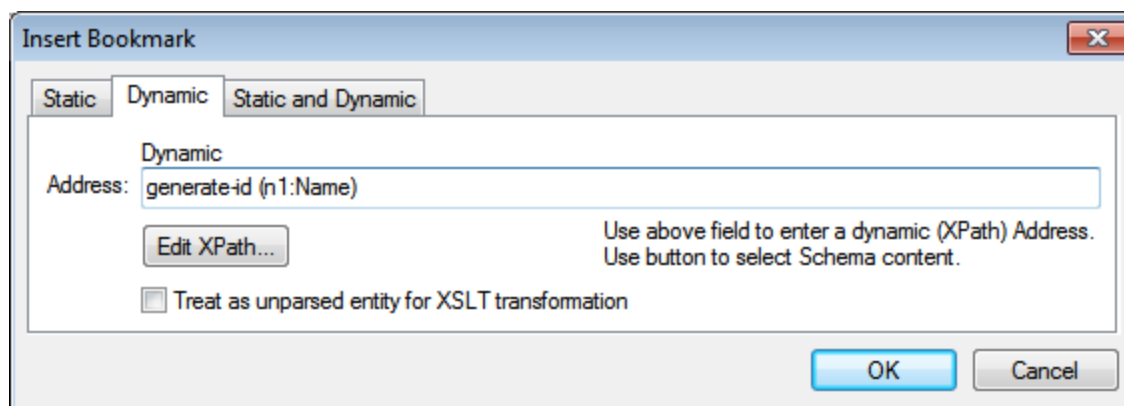
Note: A static list can also be created by placing the cursor at the location where the list is to be created and then clicking the Bulleted List icon or Numbered List icon in the [Insert Design Elements toolbar](#)⁷⁰⁸ as required. A dynamic list can also be created by dragging a node from the Schema Tree into the design.

16.8.14 Bookmark

The **Bookmark** command allows you to insert a bookmark (or anchor) anywhere in the SPS. A bookmark can be referenced by a [Hyperlink](#)⁷⁷⁶.

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](#) ³³⁹, 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](#) ³⁴⁴. 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.

16.8.15 Hyperlink



The **Hyperlink** command enables you to insert a link from any part of the output document (HTML or RTF) 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](#)⁷⁰⁸.
3. In the [Insert Hyperlink dialog](#)³⁴¹ 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](#)³⁴⁴.
 - 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](#)⁴⁴⁰.

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.

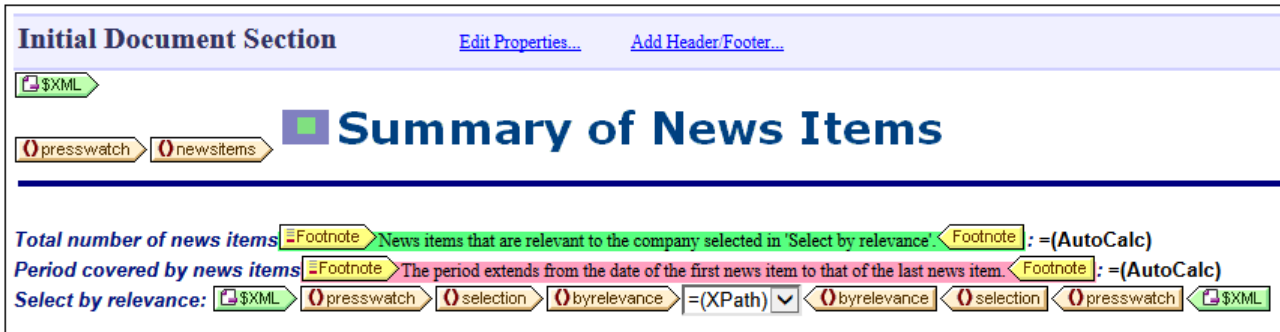
16.8.16 Footnote

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*).

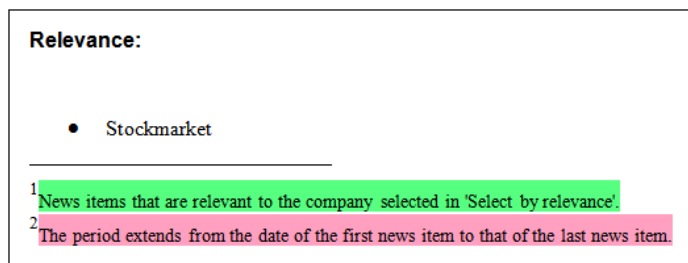
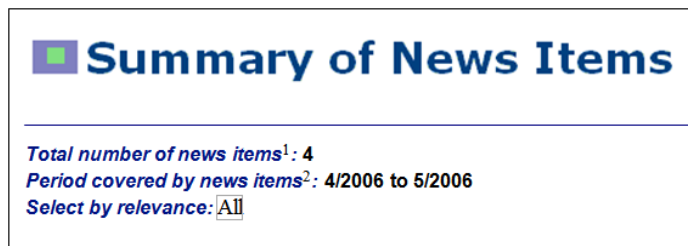
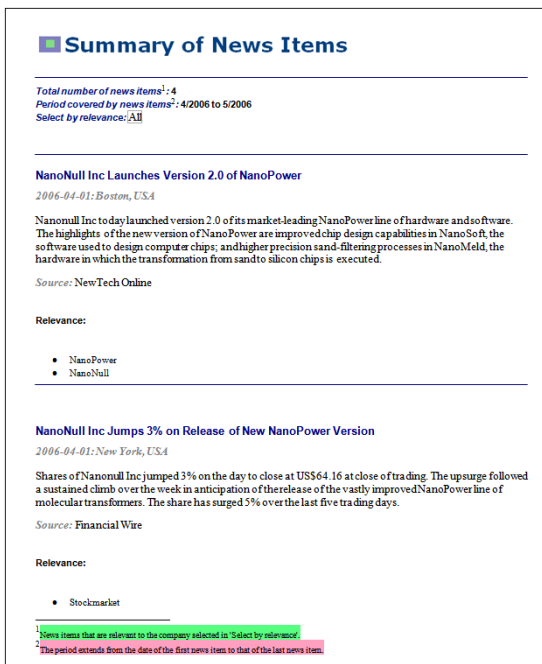
Note the following points:

- 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*).



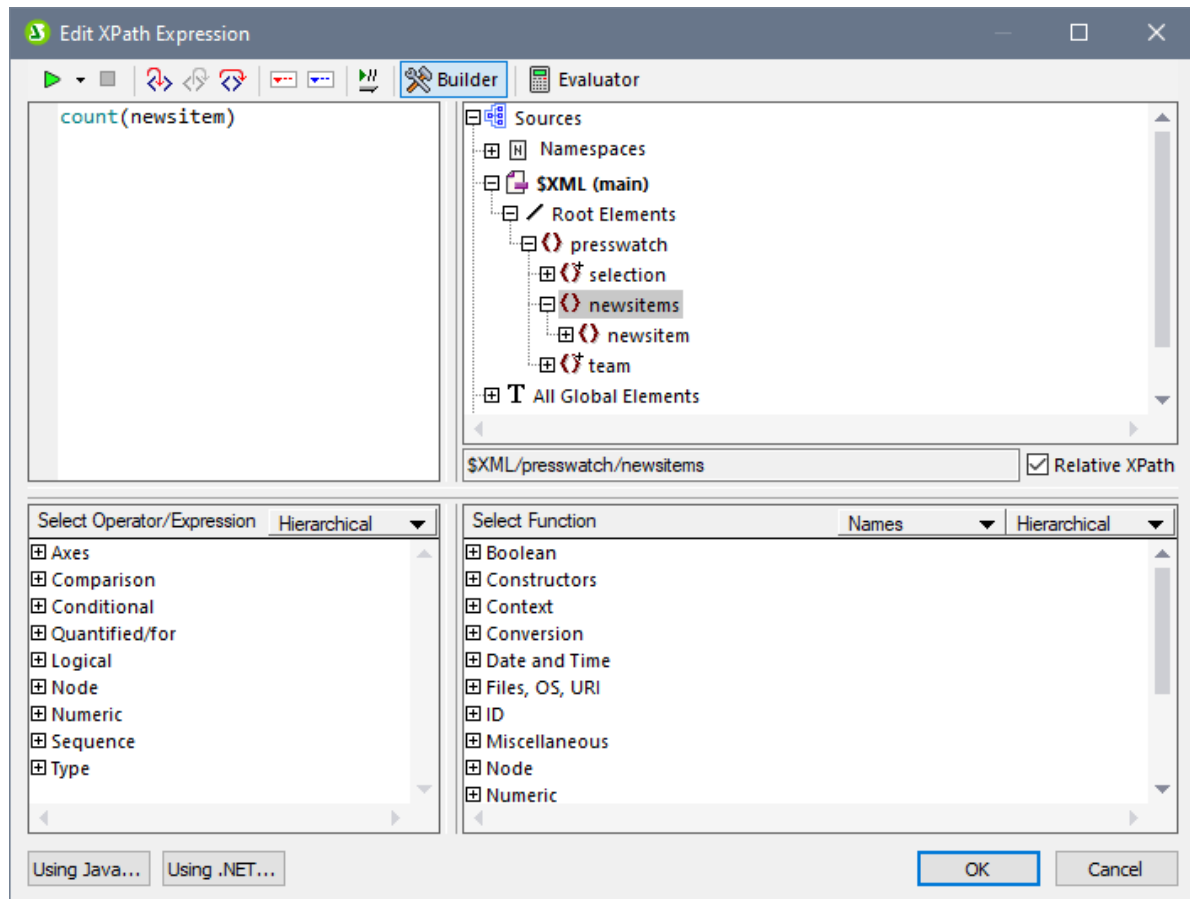
Note: Formatting of footnote numbers is not supported.

16.8.17 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](#)⁶⁸⁵ that pops up (*screenshot below*), enter the XPath expression.



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 *Relative XPath* check box is checked) or as an absolute expression starting from the document node (if the *Relative XPath* check box is unchecked).


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, Text 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](#)²⁸³). 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](#)²⁸⁴. 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,

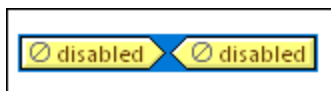
`select condition branch | when`. Click the **Edit** button  for the XPath item. This pops up the Edit XPath Expression dialog (*screenshot above*), in which you can edit the expression. Click **OK** when done.

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.

16.8.18 Disabled

The **Disabled** command inserts a Disabled component at the cursor location (*screenshot below*). (To put the Disabled component around selected content, use the **Enclose With | Disabled**  command.)



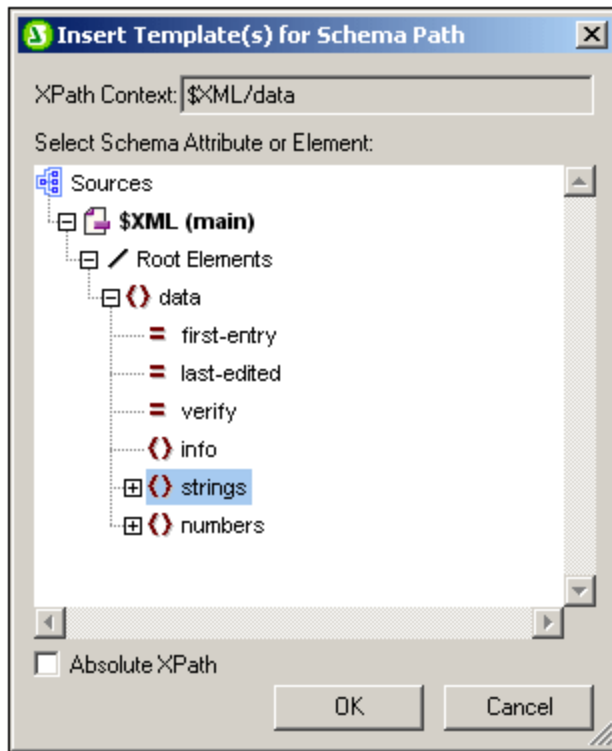
Content inside a Disabled component is ignored in the output. So you can add content that you want to ignore to the Disabled component. The Disabled component thus serves as a way to comment out content.

When you want to reinstate disabled content, simply remove the Disabled tags from around the content. To do this, select the Disabled component, right-click, and select **Remove Tag Only**.

16.8.19 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*).



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).



16.8.20 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.

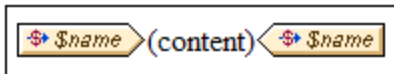
2. Click the **Insert | User-Defined Template** command. This pops up the [Edit XPath Expression dialog](#) ⁶⁸⁵.
3. Enter the XPath expression to select the node you want. There are a few points to note in this connection: (i) The XPath expression will be evaluated in the context of the node within which 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](#) ¹³⁷.

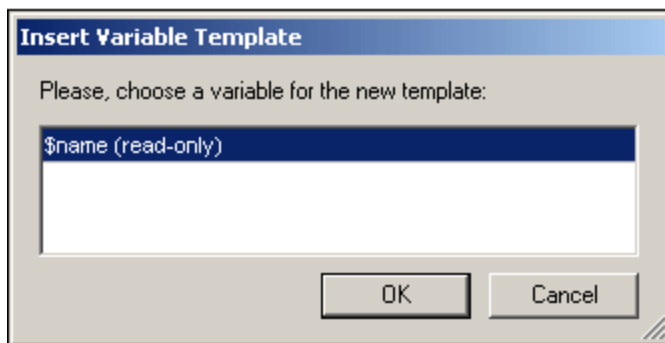
16.8.21 Variable Template

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.



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](#) ³⁰² defined in the SPS. Select the variable for which you wish to add a variable template.
4. Click **OK** to finish.

16.8.22 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.

16.8.23 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](#)⁷⁰⁸. 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](#)¹⁸⁷.

16.8.24 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](#)³²²
- [TOC Bookmark](#)³¹⁹
- [TOC Bookmark \(Wizard\)](#)³¹⁵
- [TOC Reference](#)³²⁵
- [TOC Reference | Entry Text / Leader / Page Reference](#)³²⁵
- [Hierarchical Numbering](#)³²⁵
- [Sequential Numbering](#)³²⁵
- [Level](#)³¹⁶
- [Level Reference](#)³²⁴
- [Template Serves as Level](#)³¹⁶

Note: These commands are also available as commands in a context menu, depending on where you right click in the design.

16.8.25 New Document

The **Insert New Document** command inserts a New Document template (*screenshot below*) at the cursor insertion point.



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](#) ²⁶¹.

16.8.26 Page / Column / Document Section

With the **Page / Column / Document Section** command you can insert, for paged media output, a page break (HTML printouts and RTF outputs) and page number (RTF output). 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 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 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](#)⁴⁰¹ 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](#)³⁹³ 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**.

16.8.27 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](#)¹⁴⁰ or a [User-Defined XML Text Block](#)¹⁴¹. How to use these two components is described in the section [SPS File: Content | User-Defined Elements, XML Text Blocks](#)¹⁴⁰.

16.9 Enclose With Menu

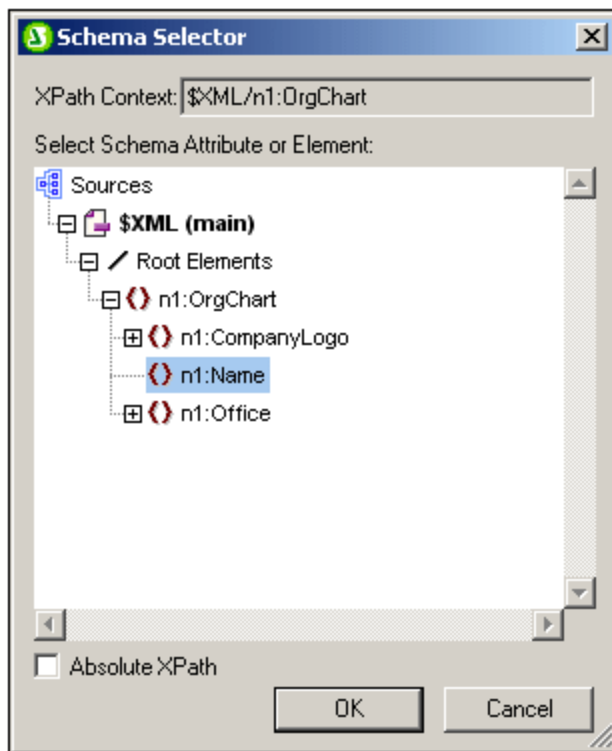
The **Enclose with** menu provides commands enabling you to enclose a selection in the design with a variety of design components. Some of these commands are available as [toolbar icons](#)⁷⁰² that enable you to insert the component in the design (equivalent commands are available in the [Insert menu](#)⁷⁶³). 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.

16.9.1 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.

16.9.2 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](#)⁶⁸⁵ 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 | User-Defined Templates](#)²⁴⁸.

16.9.3 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](#)²⁵¹.
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](#)²⁵¹.

16.9.4 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](#)⁷⁰⁸.

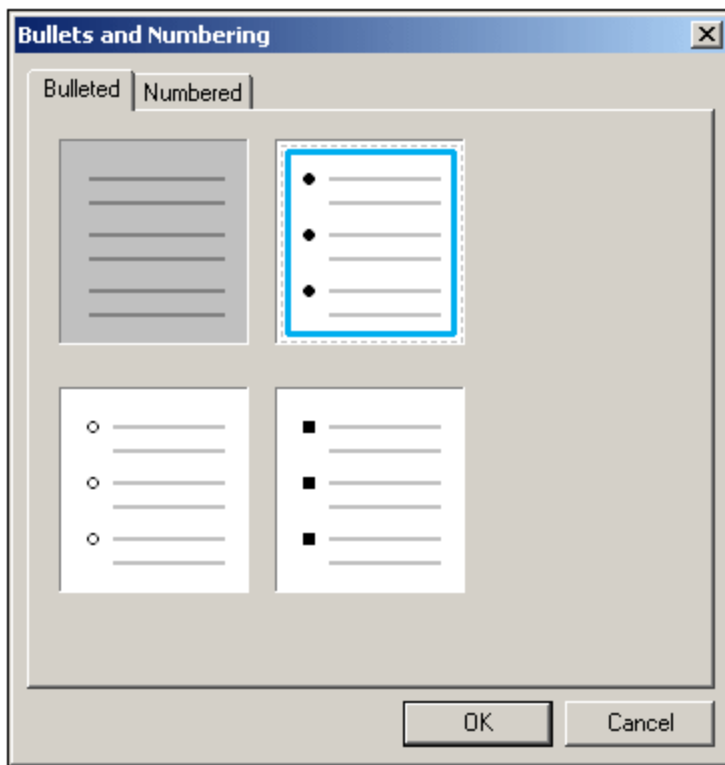
The **Special Paragraph** command allows you to assign a predefined format to the selected node. The available predefined formats can also be selected from the combo box in the toolbar.

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.

16.9.5 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](#)⁷⁰⁸.

16.9.6 Bookmarks and Hyperlinks

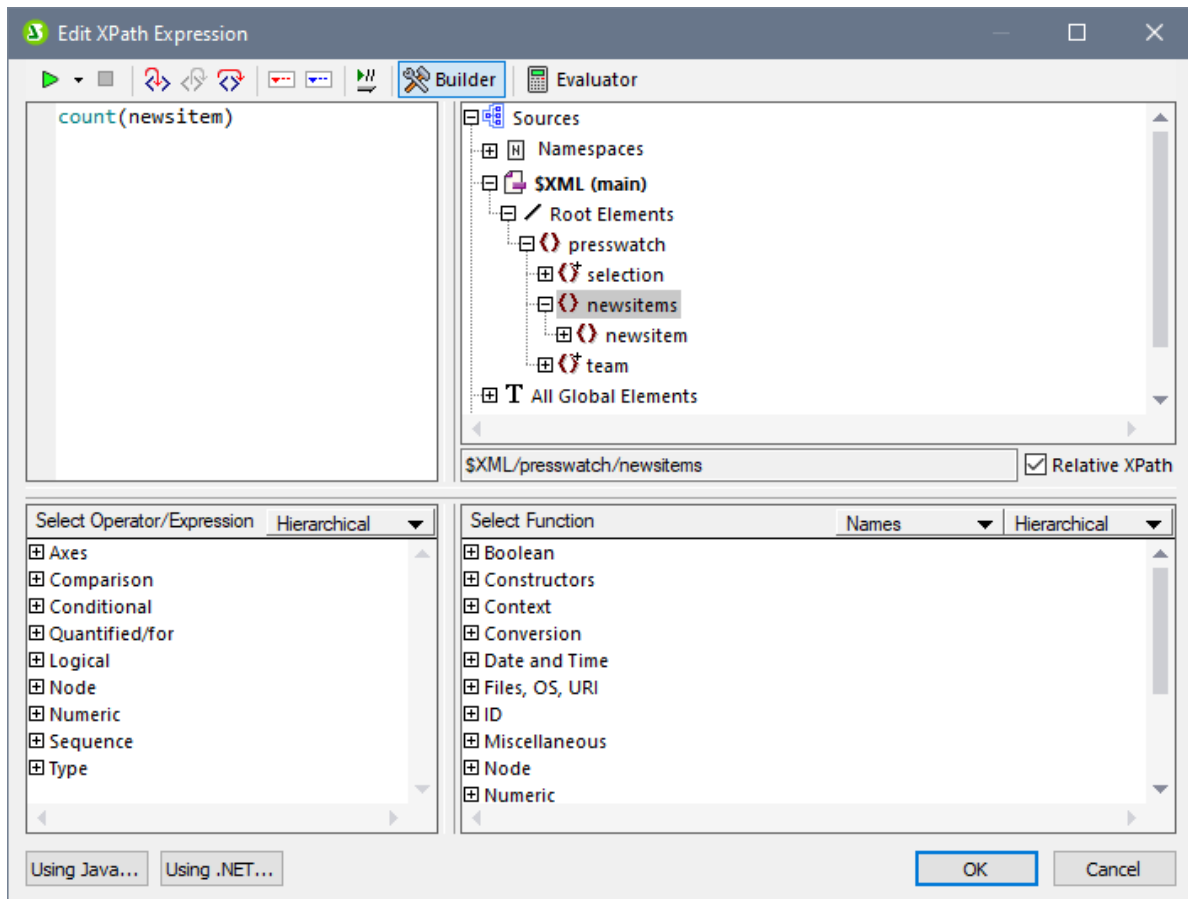
The **Enclose with | Bookmark** and **Enclose With | Hyperlink** commands are enabled when some text or component in the SPS design is selected. These commands enable a bookmark and hyperlink, respectively, to be created around the selection. For more information about how bookmarks and hyperlinks work and how to create them, see the section [Advanced Features | Table of Contents, Referencing, Bookmarks](#)³³⁹.

16.9.7 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](#)⁶⁸⁵ that pops up (*screenshot below*), enter the XPath expression.



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 *Relative XPath* check box is checked) or as an absolute expression starting from the document node (if the *Relative XPath* check box is unchecked).


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, Text 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](#)²⁸³). 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](#)²⁸⁴. 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,

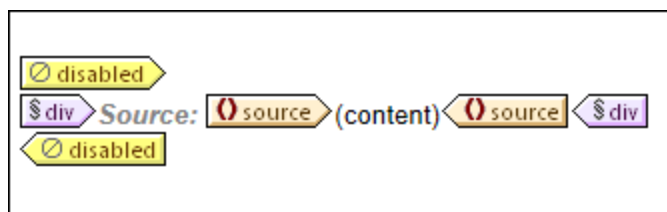
select `condition branch | when`. Click the **Edit** button  for the XPath item. This pops up the Edit XPath Expression dialog (*screenshot above*), in which you can edit the expression. Click **OK** when done.

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.

16.9.8 Disabled

The **Disabled** command encloses selected content (including design components) with a Disabled component (see *screenshot below*).



Content inside a Disabled component is ignored in the output. So you can enclose content that you want to ignore with a Disabled component.

When you want to reinstate disabled content, simply remove the Disabled tags from around the content. To do this, select the Disabled component, right-click, and select **Remove Tag Only**.

16.9.9 TOC Bookmarks and TOC Levels

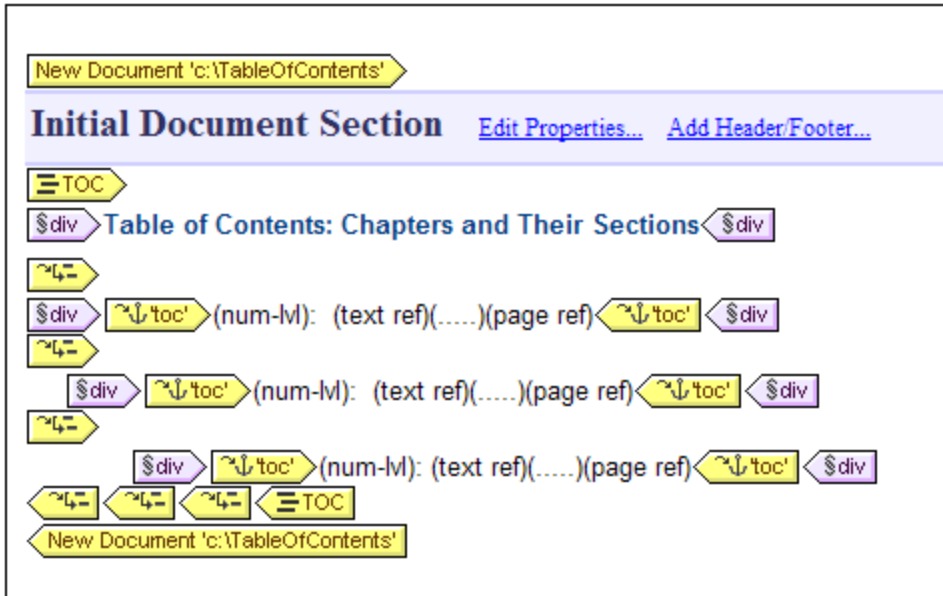
When a component in the design is selected, it can be enclosed with one or more relevant Table of Contents (TOC) components. The list of TOC commands is as follows. For the details of how to use them click on the respective links, which will take you to the section on how to use that particular TOC component.

- [TOC Bookmark](#) ³¹⁹
- [TOC Bookmark \(Wizard\)](#) ³¹⁵
- [Level](#) ³¹⁶
- [Level Reference](#) ³²⁴

Note: These commands are also available as commands in a context menu, depending on where you right click in the design.

16.9.10 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](#) ²⁶¹.

16.9.11 User-Defined Element

The **Enclose with | User-Defined Element** command creates a [User-Defined Element](#) ¹⁴⁰ around the selection in the design. How to use user-defined elements is described in the section [SPS File: Content | User-Defined Elements](#) ¹⁴⁰.

16.10 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](#)¹⁵³ and [HTML properties for tables and its sub-components](#)¹⁵³.

The Table commands are available in the **Table** menu (see *list below*) and as icons in the [Table toolbar](#)⁷⁰⁵. 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](#)⁷⁹⁴ 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](#)¹⁵³ and/or [Properties sidebar](#)¹⁵³, define the required property for that table component.

The following commands are available in the Table menu:

- [Insert Table, Delete Table](#)⁷⁹⁴
- [Add Table Headers, Footers](#)⁷⁹⁵
- [Append/Insert Row/Column](#)⁷⁹⁵
- [Delete Row, Column](#)⁷⁹⁶
- [Join Cell Left, Right, Below, Above](#)⁷⁹⁶
- [Split Cell Horizontally, Vertically](#)⁷⁹⁶
- [View Cell Bounds, Table Markup](#)⁷⁹⁷
- [Table Properties](#)⁷⁹⁷
- [Vertical Alignment of Cell Content](#)⁷⁹⁸

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.


16.10.1 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](#) ¹⁴⁶.

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.


16.10.2 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 `th` elements, they appear bold (because the bold formatting is inherent in the `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.

16.10.3 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.


16.10.4 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.

16.10.5 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.

16.10.6 Split Cell Horizontally, Vertically

The **Split Cell Horizontally** command  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  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.

16.10.7 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.

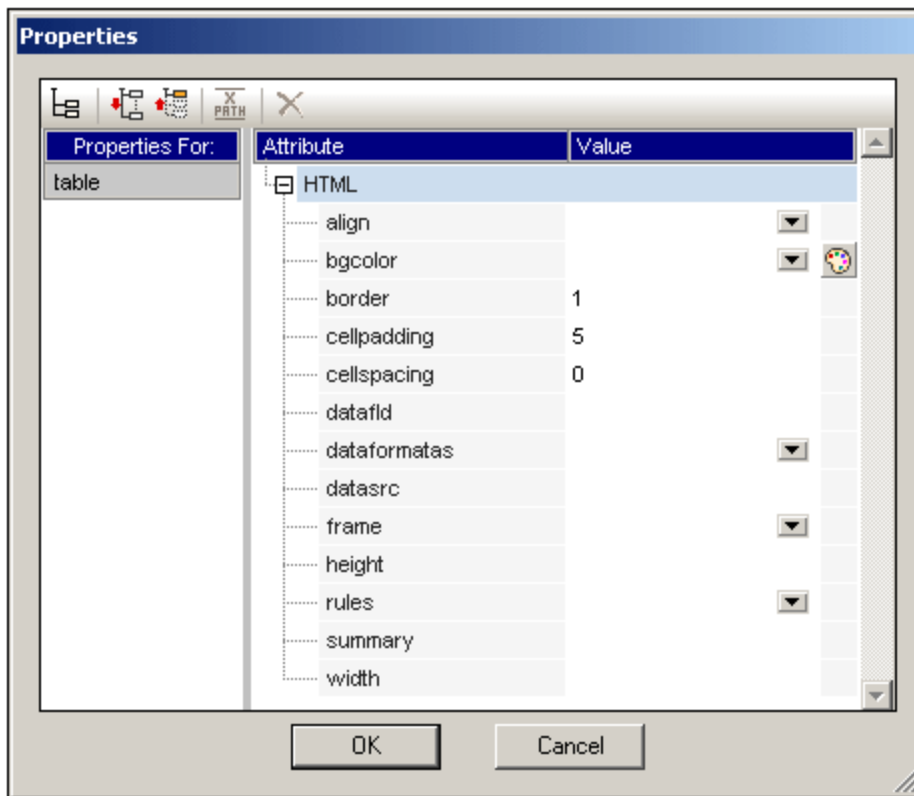


The **View Table Markup** command toggles the display of the blue column and row markers on and off.

16.10.8 Table Properties



The **Table Properties** command is enabled when the cursor is placed inside a [static or dynamic table](#)¹⁴³. Clicking the command, pops up the Properties sidebar, with the *Table* component selected (*screenshot below*).

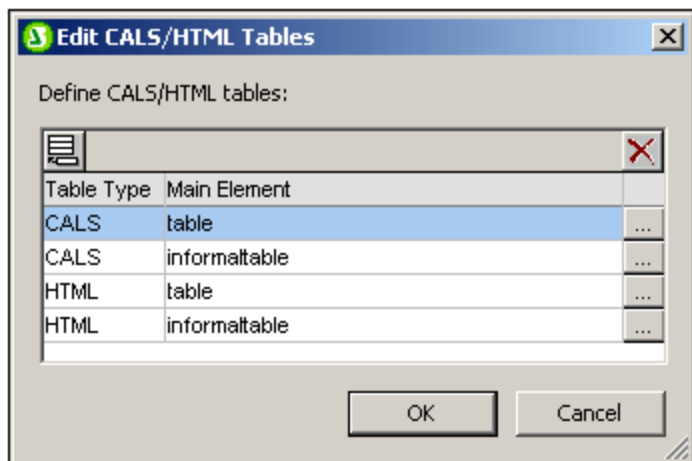


You can now edit the properties of the table. Click **OK** when done.

16.10.9 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](#)¹⁴³.

16.10.10 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.

16.11 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.

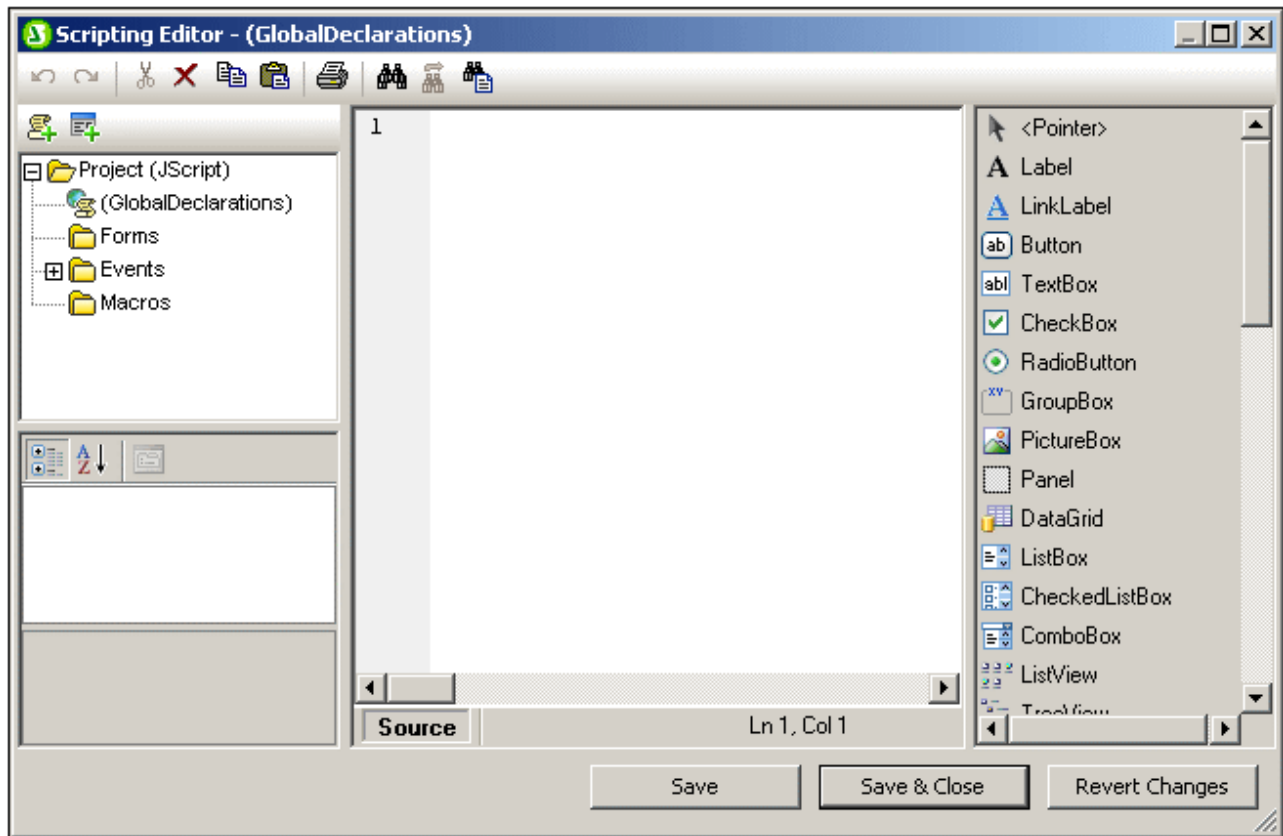
The commands in the Authentic menu are listed below:

- [Edit Authentic Scripts](#) ⁷⁹⁹
- [Custom Toolbar Buttons](#) ⁸⁰⁰
- [Check Macro References](#) ⁸⁰³
- [Auto-Add Date Picker](#) ⁸⁰⁴
- [Auto-Add DB Controls](#) ⁸⁰⁴
- [Reload, Validate XML](#) ⁸⁰⁴
- [Select New Row with XML Data for Editing](#) ⁸⁰⁵
- [Define XML Entities](#) ⁸⁰⁶
- [Markup Commands](#) ⁸⁰⁶
- [\(Dynamic Table\) Row Commands](#) ⁸⁰⁷

Each of these commands is described in detail in the sub-sections of this section.

16.11.1 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.

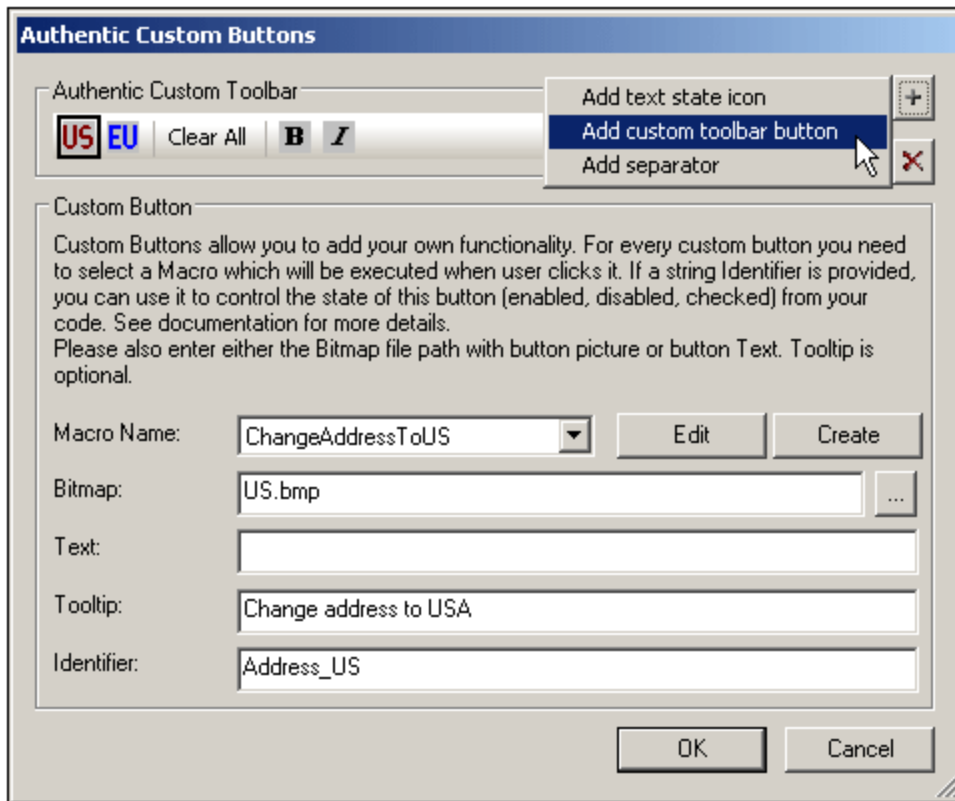


For an overview of how scripts can be used in Authentic View, see the section, [Scripting for Authentic](#)⁶⁵⁰. For a description of how the Scripting Editor works, see the section, [Scripting](#)⁶⁵⁴ in the [Programmers' Reference](#)⁸⁵³.

16.11.2 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.



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](#)⁸⁰¹ and [custom buttons](#)⁸⁰² are described in detail below.

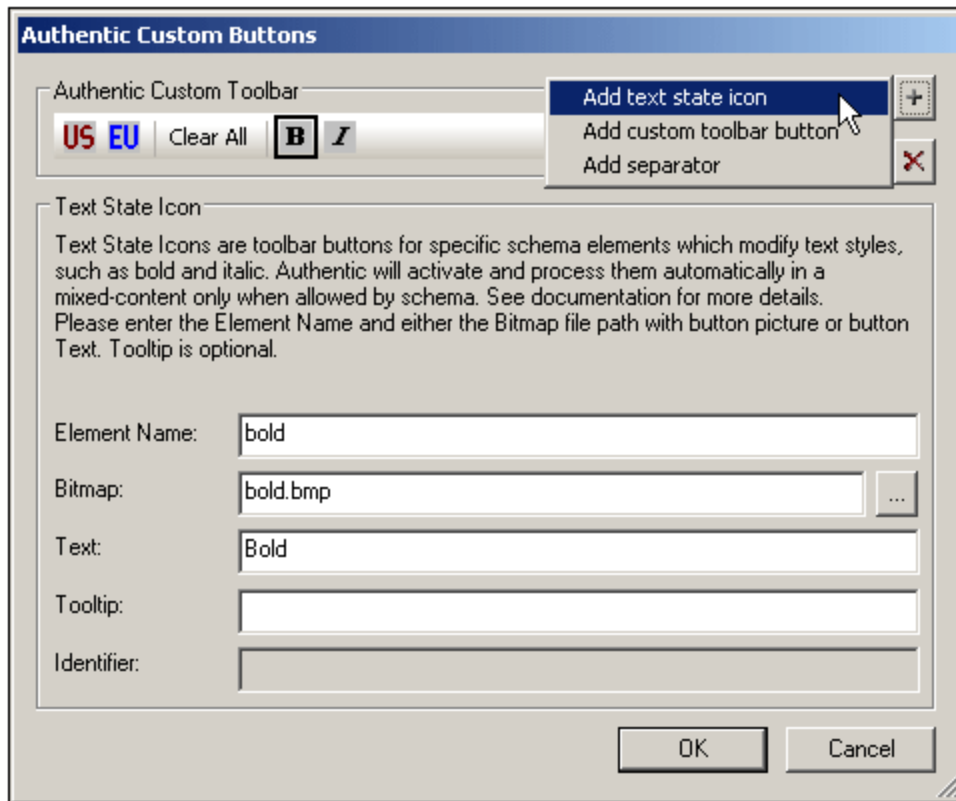
Moving a button and deleting a button

To move a button or separator to another location 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.

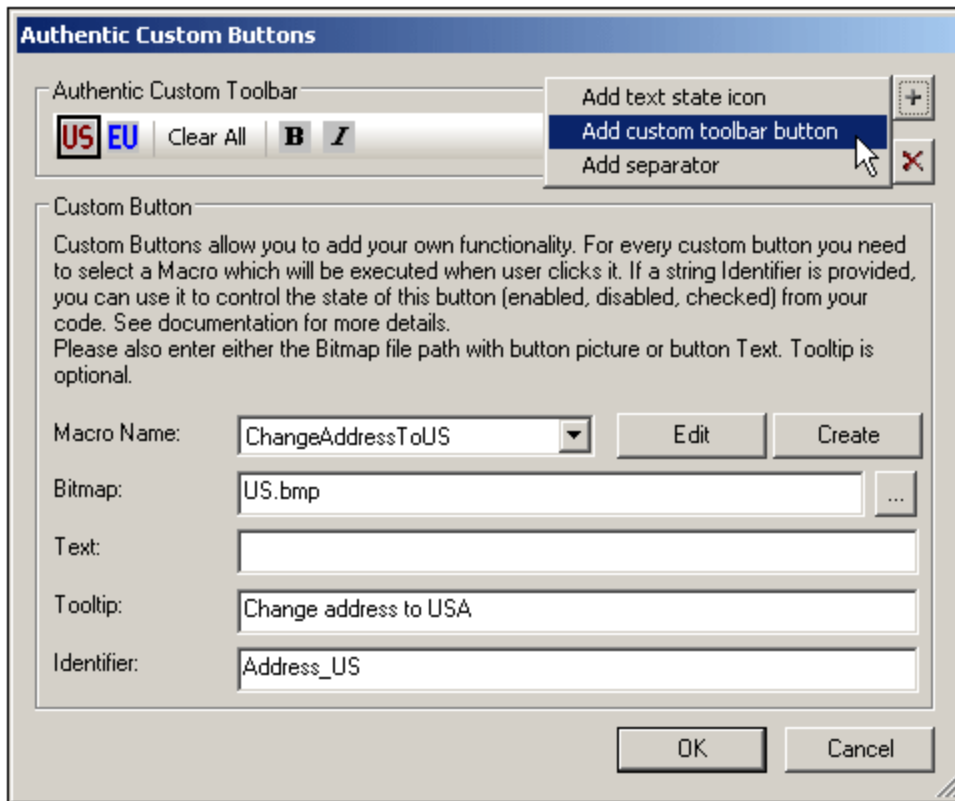


The screenshot above shows the text state icon for the `bold` element selected. This text state icon uses an image named `bold.bmp`. Text for the button can be entered as a fallback. For the text state icon defined in the screenshot above, if the image cannot be found, then the text `Bold` will be used 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](#)⁸⁵⁴ 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](#)⁸⁵⁴.
- 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` in the `Authentic\Scripting` folder of the `Examples` project in the Project window.

16.11.3 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.

16.11.4 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 `xs:date` or `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 `contents` placeholder or input field.

16.11.5 Auto-Add DB Controls

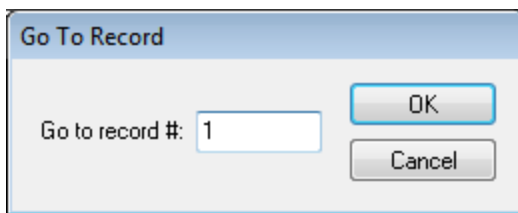


This is a toggle command that switches the Auto-Add DB Controls feature on and off.

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.

16.11.6 Reload Authentic View, Validate XML

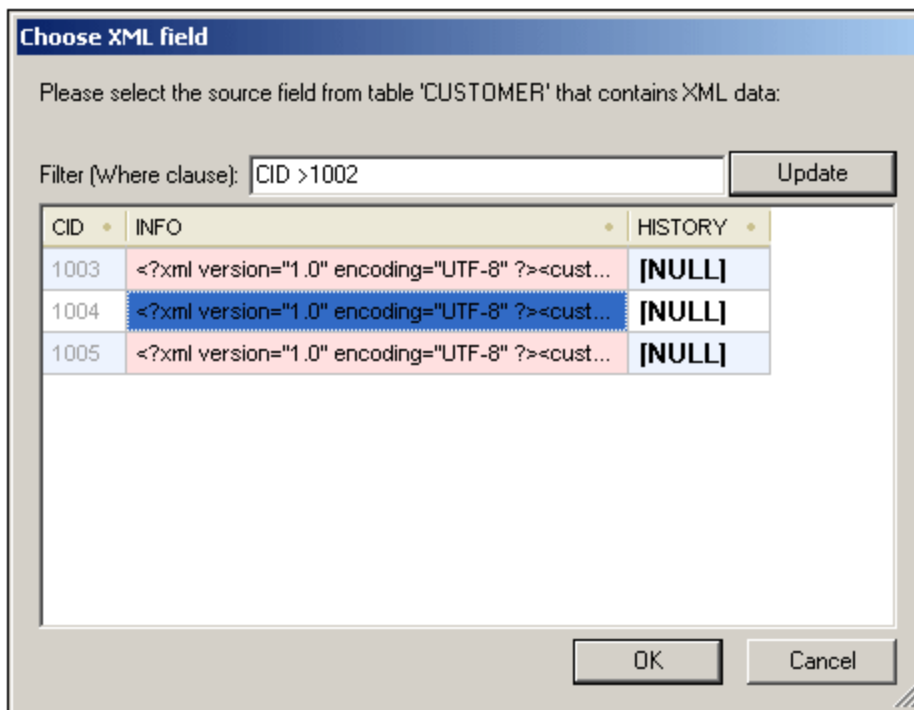
The **Reload** command reloads the Authentic XML data file. This can be useful if the file has been modified outside StyleVision, especially by another user working from another machine.

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](#)⁷³⁸. 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.

16.11.7 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.

16.11.8 Define XML Entities

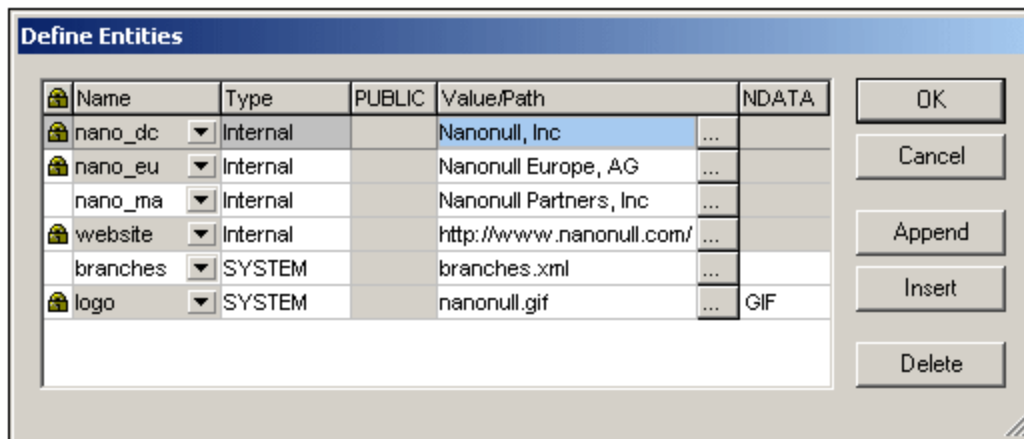


The **Define XML Entities** command is available only in Authentic View. With the **Define Entities** command in Authentic View, you can define entities that you want to add to your **XML document**. After an entity has been defined, it can be inserted in the XML document by right-clicking at the location 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.

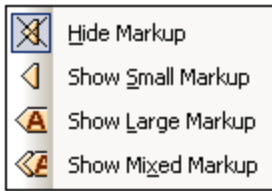
Clicking the command, pops up the Define Entities dialog (*screenshot below*).



For a description of how to use this dialog, see [Define Entities](#)⁶⁴⁶ in the Authentic View documentation.

16.11.9 View Markup

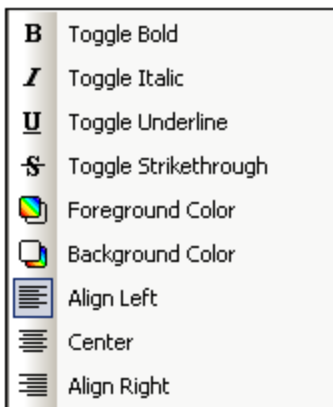
The **View Markup** command has a submenu with options to control markup in the Authentic XML document. With the four markup commands (*screenshot below*), you can select between the Hide Markup and the various Show Markup modes.



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](#)⁴³³ 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](#)⁴³³) small or large markup for some node/s.

16.11.10 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.



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](#)³⁸¹.

16.11.11 (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.

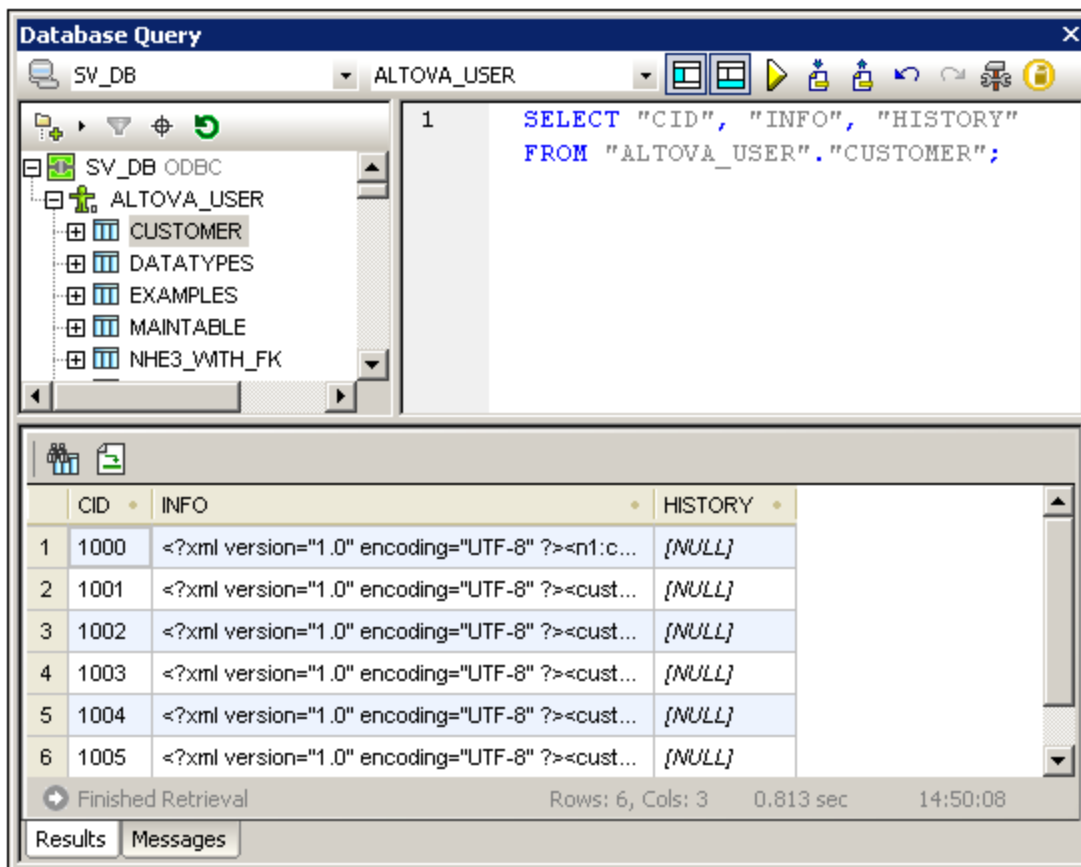
16.12 Database Menu

The **Database** menu contains commands to query the connected database and to edit and clear the filters applied to the connected database.

- [Query Database](#)⁸⁰⁹ starts up the [Connect to Database](#)⁴⁹² process and opens the Database Query window.
- [Edit DB Filter. Clear DB Filter](#)⁸¹⁰, to access the Edit DB Filters dialog and clear DB Filters, respectively.


16.12.1 Query Database

The **Query Database**  command pops up the [Database Query window](#)⁵⁹⁹ (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](#)⁵⁹⁹.



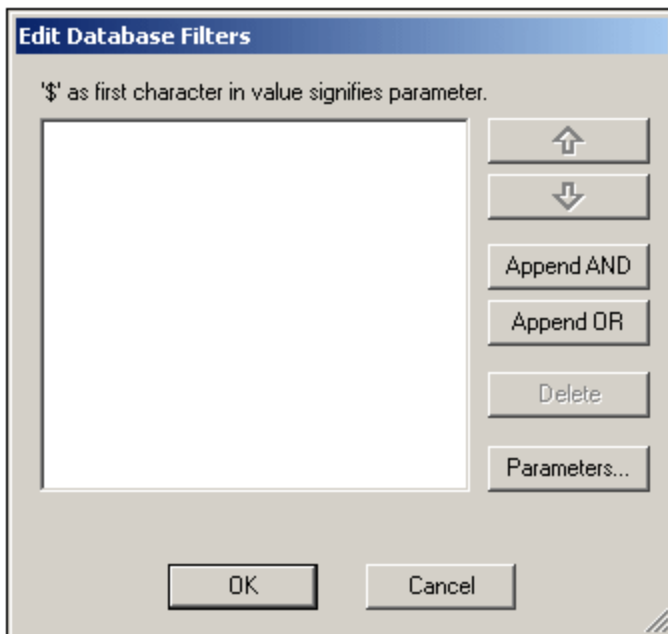
The Database Query window is toggled on and off by clicking the **Query Database** command.

16.12.2 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.

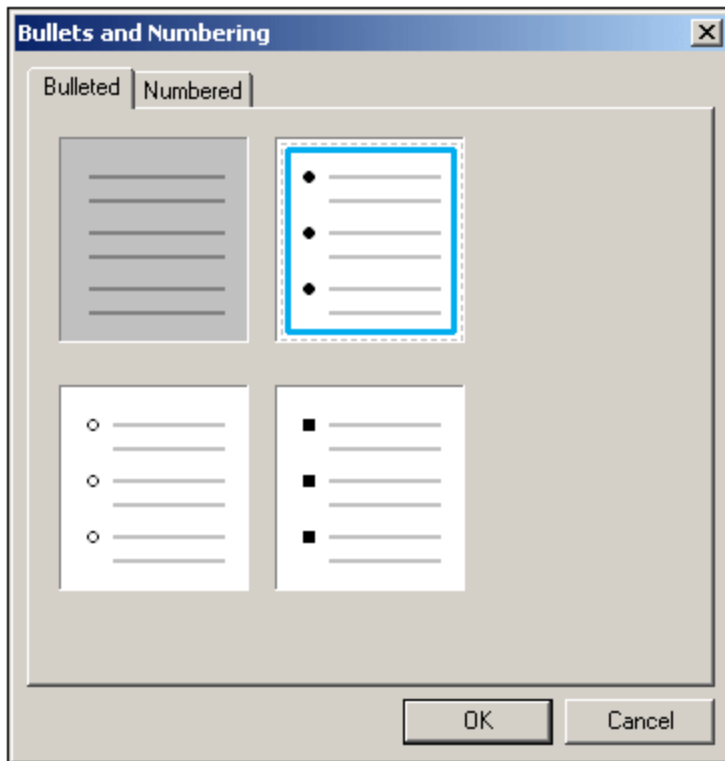
16.13 Properties Menu

The **Properties** menu contains commands that enable you to insert lists and define datatype formats for the [input formatting](#)³⁵⁴ feature. The description of the commands is organized into the following sub-sections:

- [Bullets and Numbering](#)⁸¹¹ command, to insert lists.
- [Predefined Format Strings](#)⁸¹¹ command, to define numeric datatype formats for a given SPS.

16.13.1 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.



16.13.2 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](#)³⁵⁴ dialog. In the Value Formatting dialog, you can either create a format directly or select from a drop-down list of predefined formats.

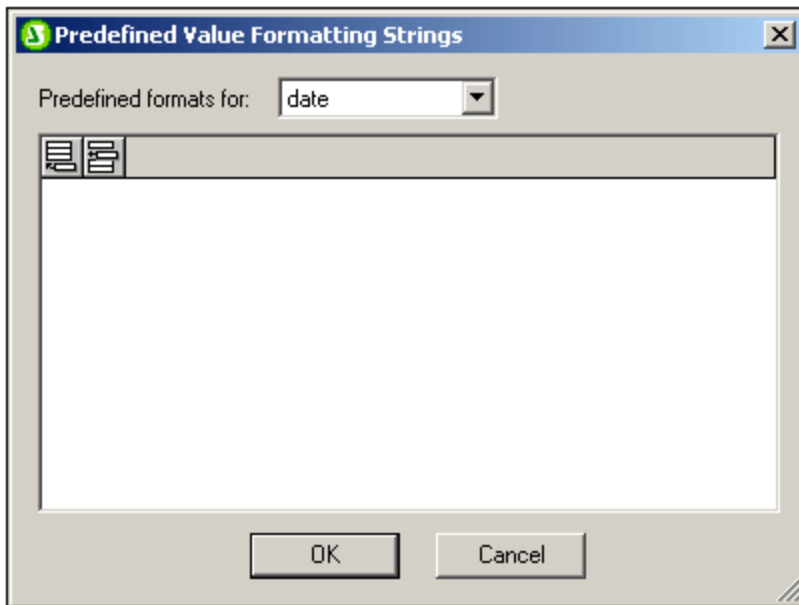
The predefined formats that are available in the dropdown list are of two types:

- 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.

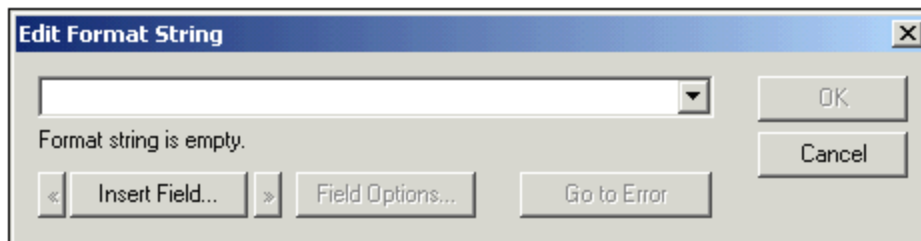
Creating a predefined value formatting string

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:



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:



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](#)³⁵⁴. 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 the following points:

- 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.

16.14 Tools Menu

The **Tools** menu contains the spell-check command and commands that enable you to customize StyleVision.

The description of the Tools menu commands is organized into the following sub-sections:

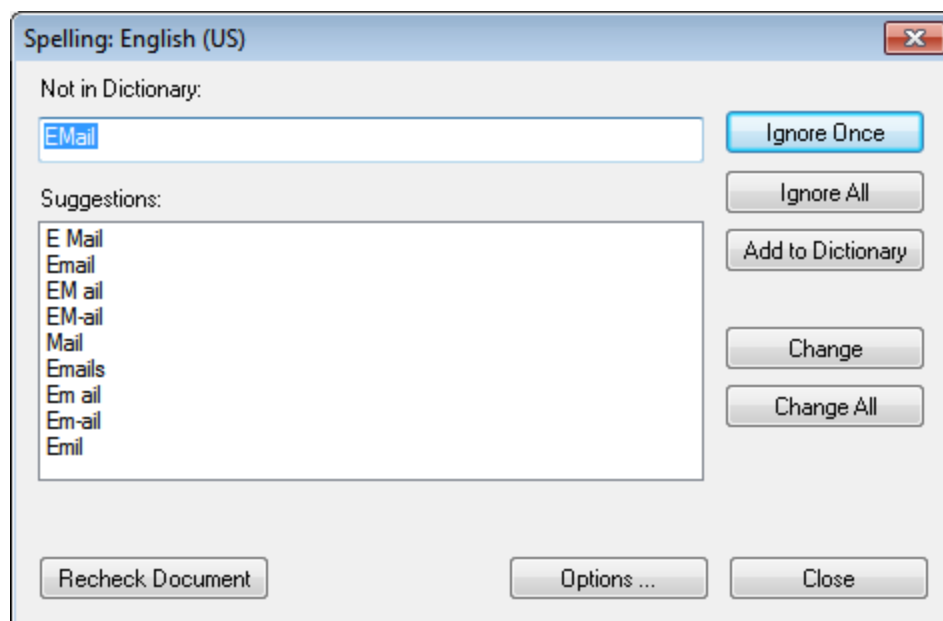
- [Spelling](#) ⁸¹⁴
- [Spelling Options](#) ⁸¹⁵
- [Global Resources](#) ⁸¹⁸
- [Active Configuration](#) ⁸¹⁹
- [XML Schema Manager](#) ²¹¹
- [Customize](#) ⁸³⁵
- [Options](#) ⁸³⁹

16.14.1 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>. 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.

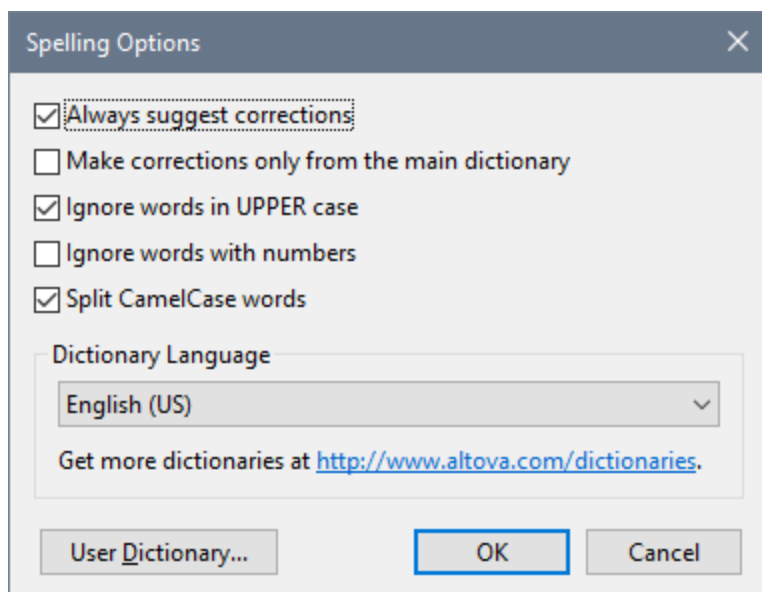


You can then select an entry from the list in the Suggestions pane and click **Change** or **Change All** to change the highlighted instance of this spelling or all its instances, respectively. (Double-clicking a word in the Suggestions list causes it to replace the unknown word.) Alternatively, you 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](#)⁸¹⁵ dialog, in which you can specify options for the spelling check.

16.14.2 Spelling Options

The **Spelling options** command opens a dialog box (shown below) in which you specify options for the spelling check.



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](http://www.altova.com/dictionaries).

Adding dictionaries for the spellchecker

For each dictionary language there are two Hunspell dictionary files that work together: a .aff file and .dic file. All language dictionaries are installed in a `Lexicons` folder at the following location: `C:`


```
\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons.
```

Within the `Lexicons` folder, different language dictionaries are each stored in a different folder: `<language name>\<dictionary files>`. For example, files for the two English-language dictionaries (English (British) and English (US)) will be stored as below:

```
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (British)
\en_GB.aff
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (British)
\en_GB.dic
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (US)\en_US.aff
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (US)\en_US.dic
```

In the Spelling Options dialog, the dropdown list of the *Dictionary Language* combo box displays the language dictionaries. These dictionaries are those available in the `Lexicons` folder and have the same names as the language subfolders in the `Lexicons` folder. For example, in the case of the English-language dictionaries shown above, the dictionaries would appear in the Dictionary Language combo box as: *English (British)* and *English (US)*.

All installed dictionaries are shared by the different users of the machine and the different major versions of Altova products (whether 32-bit or 64-bit).

You can add dictionaries for the spellchecker in two ways, neither of which require that the files be registered with the system:

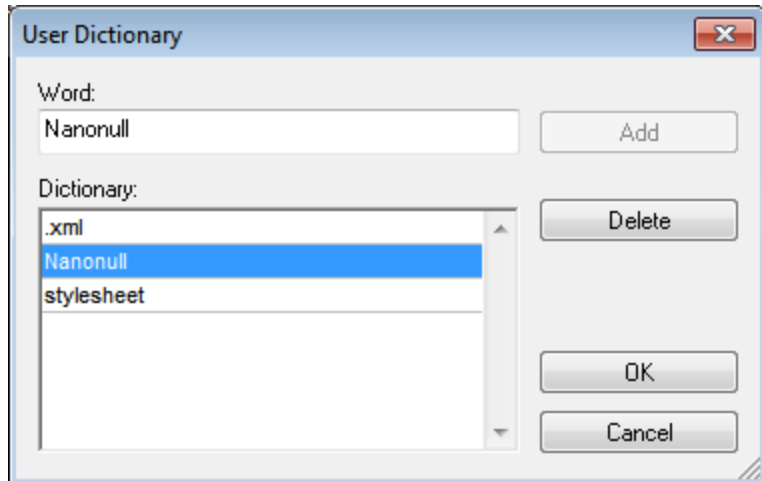
- By adding Hunspell dictionaries into a new subfolder of the `Lexicons` folder. Hunspell dictionaries can be downloaded, for example, from <https://wiki.openoffice.org/wiki/Dictionaryes> or <http://extensions.services.openoffice.org/en/dictionaries>. (Note that OpenOffice uses the zipped `OXT` format. 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.

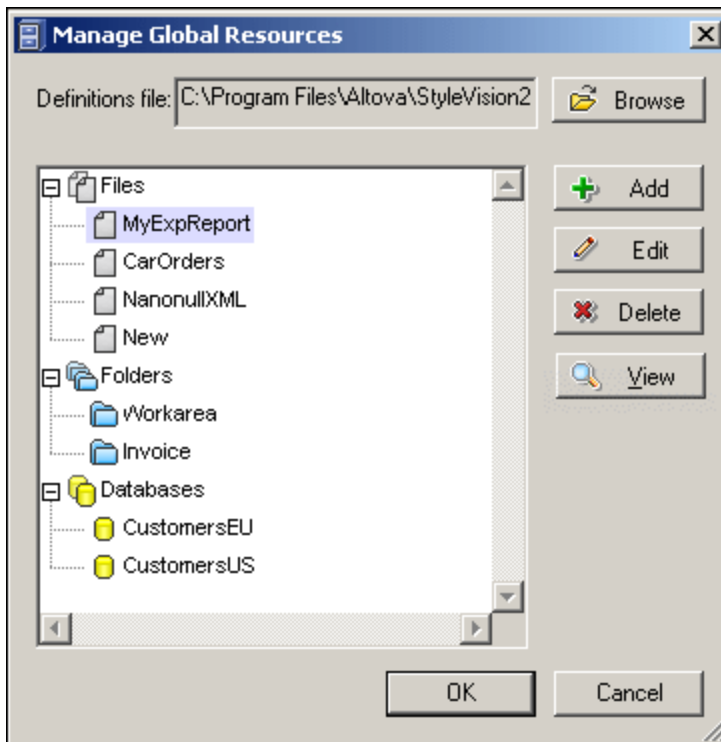
Words may also be added to the User Dictionary during a spelling check. If an unknown word is encountered during a spelling check, then the [Spelling dialog](#)⁸¹⁴ pops up prompting you for the action you wish to take. If you click the **Add to Dictionary** button, then the unknown word is added to the user dictionary.

The user dictionary is located at: `C:\Users\\Documents\Altova\SpellChecker\Lexicons\user.dic`

16.14.3 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.

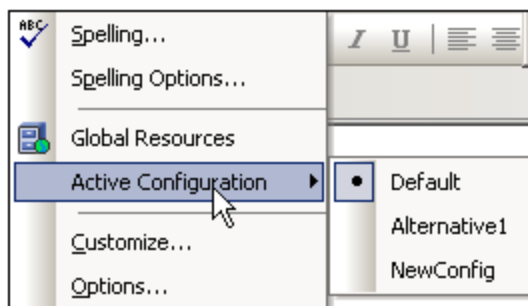


How to define global resources is described in detail in the section, [Defining Global Resources](#)⁴¹⁶.

Note: The Altova Global Resources dialog can also be accessed via the [Global Resources toolbar](#)⁷⁶⁰ (**View | Toolbars | Global Resources**).

16.14.4 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](#)⁴¹⁶ (screenshot below).



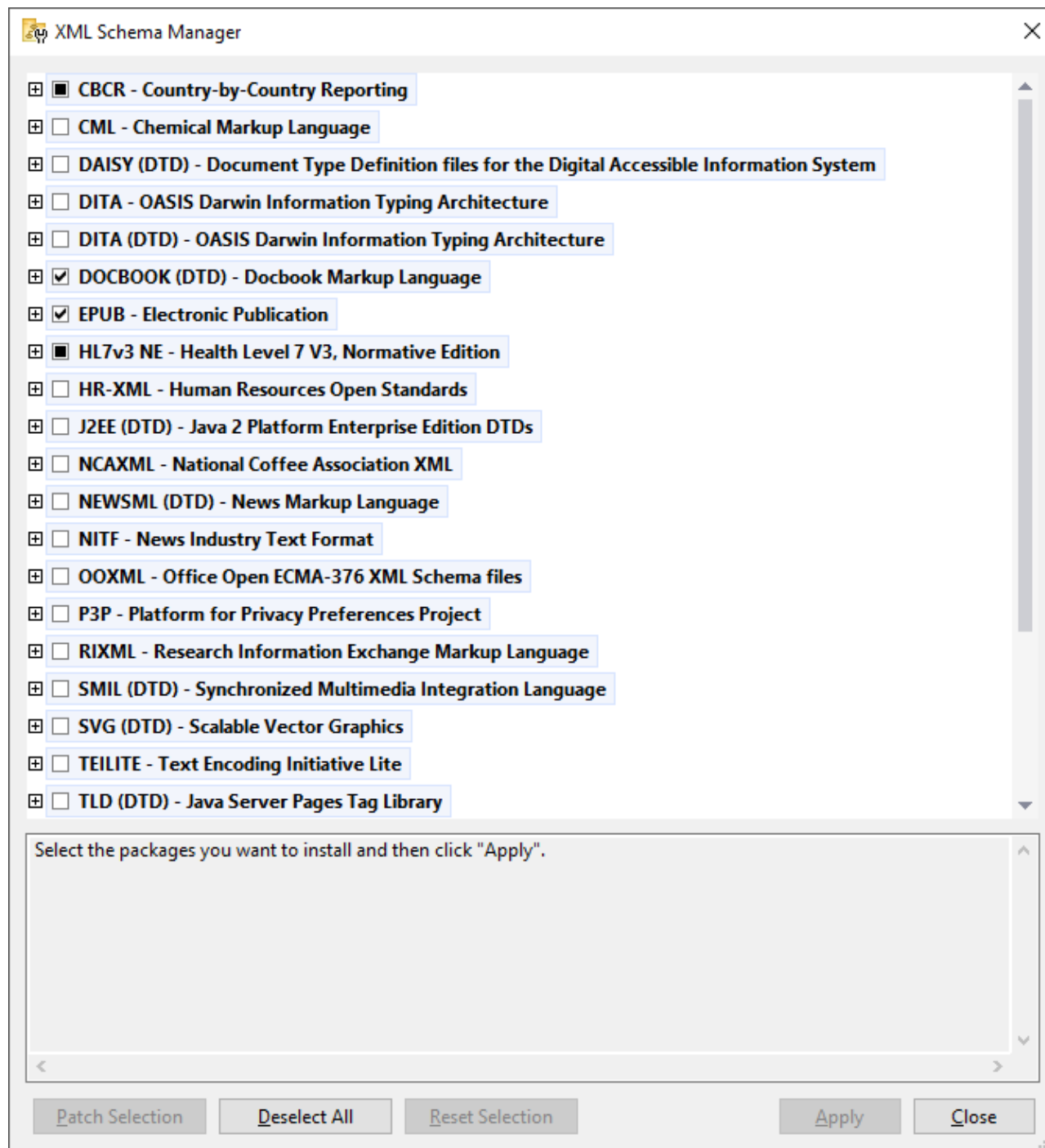
The currently active configuration is indicated with a bullet. In the screenshot above the currently active configuration is *Default*. To change the active configuration, select the configuration you wish to make active.

Note: The active configuration can also be selected via the [Global Resources toolbar](#)⁷⁶⁰ (**View | Toolbars | Global Resources**).

16.14.5 Schema Manager

XML Schema Manager is an Altova tool that provides a centralized way to install and manage XML schemas (DTDs for XML and XML Schemas) for use across all Altova's XML-Schema-aware applications, including StyleVision.

- On Windows, Schema Manager has a graphical user interface (*screenshot below*) and is also available at the command line. (Altova's desktop applications are available on Windows only; *see list below*.)
- On Linux and macOS, Schema Manager is available at the command line only. (Altova's server applications are available on Windows, Linux, and macOS; *see list below*.)



Altova applications that operate with Schema Manager

Desktop applications (Windows only)	Server applications (Windows, Linux, macOS)
XMLSpy (all editions)	RaptorXML Server, RaptorXML+XBRL Server

MapForce (all editions)	StyleVision Server
StyleVision (all editions)	
Authentic Desktop Enterprise Edition	

Installation and de-installation of Schema Manager

Schema Manager is installed automatically when you first install a new version of Altova Mission Kit or of any of Altova's XML-schema-aware applications (see *table above*).

Likewise, it is removed automatically when you uninstall the last Altova XML-schema-aware application from your computer.

Schema Manager features

Schema Manager provides the following features:

- Shows XML schemas installed on your computer and checks whether new versions are available for download.
- Downloads newer versions of XML schemas independently of the Altova product release cycle. (Altova stores schemas online, and you can download them via Schema Manager.)
- Install or uninstall any of the multiple versions of a given schema (or all versions if necessary).
- An XML schema may have dependencies on other schemas. When you install or uninstall a particular schema, Schema Manager informs you about dependent schemas and will automatically install or remove them as well.
- Schema Manager uses the [XML catalog](#) mechanism to map schema references to local files. In the case of large XML schemas, processing will therefore be faster than if the schemas were at a remote location.
- All major schemas are available via Schema Manager and are regularly updated for the latest versions. This provides you with a convenient single resource for managing all your schemas and making them readily available to all of Altova's XML-schema-aware applications.
- Changes made in Schema Manager take effect for all Altova products installed on that machine.
- In an Altova product, if you attempt to validate on a schema that is not installed but which is available via Schema Manager, then installation is triggered automatically. However, if the schema package contains namespace mappings, then there will be no automatic installation; in this case, you must start Schema Manager, select the package/s you want to install, and run the installation. If, after installation, your open Altova application does not restart automatically, then you must restart it manually.

How it works

Altova stores all XML schemas used in Altova products online. This repository is updated when new versions of the schemas are released. Schema Manager displays information about the latest available schemas when invoked in both its GUI form as well as on the CLI. You can then install, upgrade or uninstall schemas via Schema Manager.

Schema Manager also installs schemas in one other way. At the Altova website (<https://www.altova.com/schema-manager>) you can select a schema and its dependent schemas that you want to install. The website will prepare a file of type `.altova_xmlschemas` for download that contains information about your schema selection. When you double-click this file or pass it to Schema Manager via the CLI as an argument of the `install` ²²² command, Schema Manager will install the schemas you selected.

Local cache: tracking your schemas

All information about installed schemas is tracked in a centralized cache directory on your computer, located here:

<i>Windows</i>	C:\ProgramData\Altova\pkgs\.cache
<i>Linux</i>	/var/opt/Altova/pkgs\.cache
<i>macOS</i>	/var/Altova/pkgs

This cache directory is updated regularly with the latest status of schemas at Altova's online storage. These updates are carried out at the following times:

- Every time you start Schema Manager.
- When you start StyleVision for the first time on a given calendar day.
- If StyleVision is open for more than 24 hours, the cache is updated every 24 hours.
- You can also update the cache by running the [update](#)²²⁵ command at the command line interface.

The cache therefore enables Schema Manager to continuously track your installed schemas against the schemas available online at the Altova website.

Do not modify the cache manually!

The local cache directory is maintained automatically based on the schemas you install and uninstall. It should not be altered or deleted manually. If you ever need to reset Schema Manager to its original "pristine" state, then, on the command line interface (CLI): (i) run the [reset](#)²²³ command, and (ii) run the [initialize](#)²²¹ command. (Alternatively, run the `reset` command with the `--i` option.)

16.14.5.1 Run Schema Manager

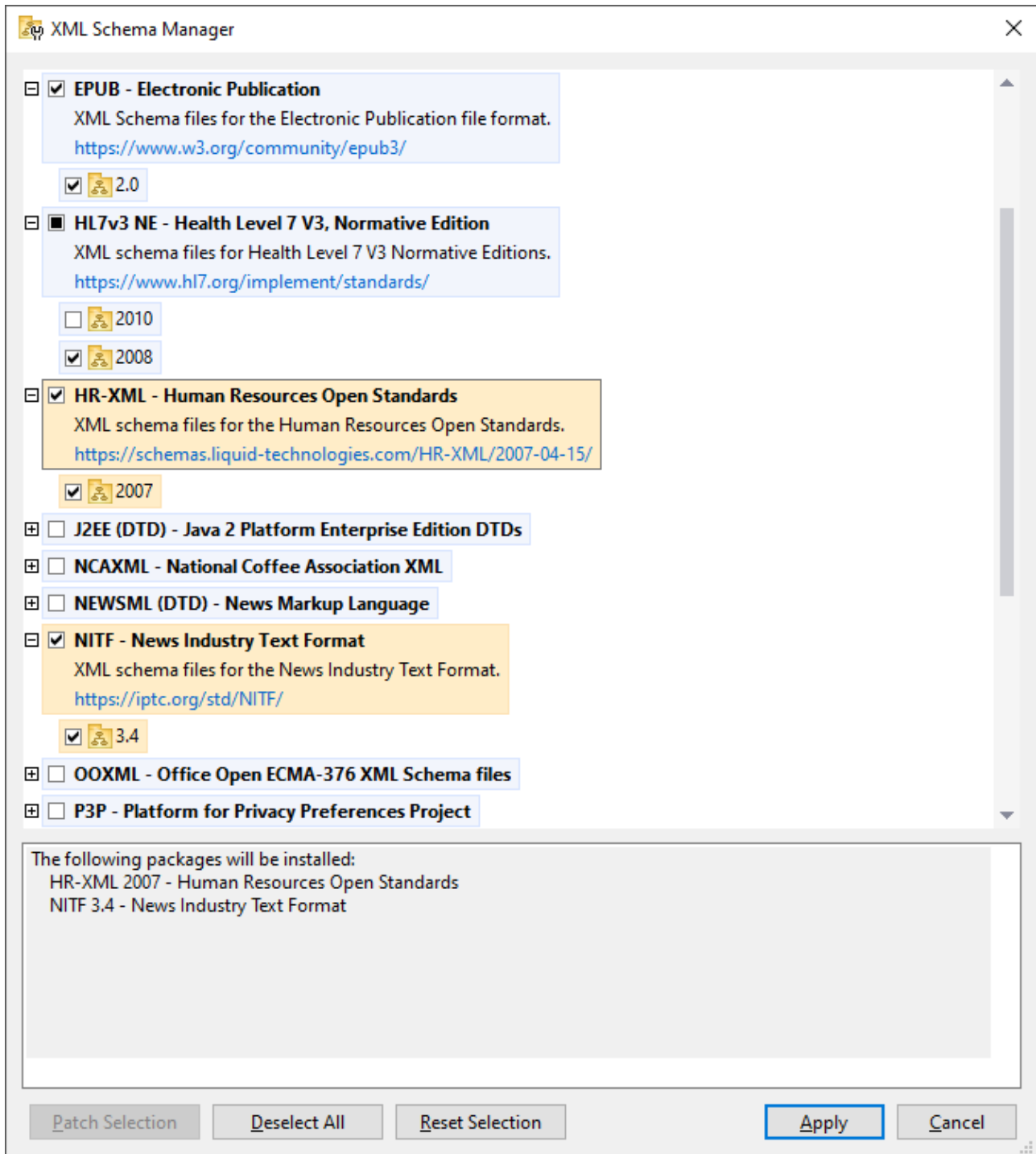
Graphical User Interface

You can access the GUI of Schema Manager in any of the following ways:

- *During the installation of StyleVision:* Towards the end of the installation procedure, select the check box *Invoke Altova XML-Schema Manager* to access the Schema Manager GUI straight away. This will enable you to install schemas during the installation process of your Altova application.
- *After the installation of StyleVision:* After your application has been installed, you can access the Schema Manager GUI at any time, via the menu command **Tools | XML Schema Manager**.
- Via the `.altova_xmlschemas` file downloaded from the [Altova website](#): Double-click the downloaded file to run the Schema Manager GUI, which will be set up to install the schemas you selected (at the website) for installation.

After the Schema Manager GUI (*screenshot below*) has been opened, already installed schemas will be shown selected. If you want to install an additional schema, select it. If you want to uninstall an already installed schema, deselect it. After you have made your selections and/or deselections, you are ready to apply your changes. The schemas that will be installed or uninstalled will be highlighted and a message about the

upcoming changes will be posted to the Messages pane at the bottom of the Schema Manager window (see *screenshot*).



Command line interface

You can run Schema Manager from a command line interface by sending commands to its executable file, `xmlschemamanager.exe`.

The `xmlschemamanager.exe` file is located in the following folder:

- *On Windows:* C:\ProgramData\Altova\SharedBetweenVersions
- *On Linux or macOS (server applications only):* %INSTALLDIR%/bin, where %INSTALLDIR% is the program's installation directory.

You can then use any of the commands listed in the [CLI command reference section](#)²²⁰.

To display help for the commands, run the following:

- *On Windows:* `xmlschemamanager.exe --help`
- *On Linux or macOS (server applications only):* `sudo ./xmlschemamanager --help`

16.14.5.2 Status Categories

Schema Manager categorizes the schemas under its management as follows:

- *Installed schemas.* These are shown in the GUI with their check boxes selected (*in the screenshot below the checked and blue versions of the EPUB and HL7v3 NE schemas are installed schemas*). If all the versions of a schema are selected, then the selection mark is a tick. If at least one version is unselected, then the selection mark is a solid colored square. You can deselect an installed schema to **uninstall** it; (*in the screenshot below, the DocBook DTD is installed and has been deselected, thereby preparing it for de-installation*).
- *Uninstalled available schemas.* These are shown in the GUI with their check boxes unselected. You can select the schemas you want to **install**.



- *Upgradeable schemas* are those which have been revised by their issuers since they were installed. They are indicated in the GUI by a 📦 icon. You can **patch** an installed schema with an available revision.

Points to note

- In the screenshot above, both CBCR schemas are checked. The one with the blue background is already installed. The one with the yellow background is uninstalled and has been selected for installation. Note that the HL7v3 NE 2010 schema is not installed and has not been selected for installation.
- A yellow background means that the schema will be modified in some way when the **Apply** button is clicked. If a schema is unchecked and has a yellow background, it means that it will be uninstalled when the **Apply** button is clicked. In the screenshot above the DocBook DTD has such a status.
- When running Schema Manager from the command line, the `list` command is used with different options to list different categories of schemas:

<code>xmlschemamanager.exe list</code>	Lists all installed and available schemas; upgradeables are also indicated
<code>xmlschemamanager.exe list -i</code>	Lists installed schemas only; upgradeables are also indicated
<code>xmlschemamanager.exe list -u</code>	Lists upgradeable schemas




Note: On Linux and macOS, use `sudo ./xmlschemamanager list`

16.14.5.3 Patch or Install a Schema

Patch an installed schema

Occasionally, XML schemas may receive patches (upgrades or revisions) from their issuers. When Schema Manager detects that patches are available, these are indicated in the schema listings of Schema Manager and you can install the patches quickly.

In the GUI

Patches are indicated by the  icon. (Also see the previous topic about [status categories](#)²¹⁶.) If patches are available, the **Patch Selection** button will be enabled. Click it to select and prepare all patches for installation. In the GUI, the icon of each schema that will be patched changes from  to , and the Messages pane at the bottom of the dialog lists the patches that will be applied. When you are ready to install the selected patches, click **Apply**. All patches will be applied together. Note that if you deselect a schema marked for patching, you will actually be uninstalling that schema.

On the CLI

To apply a patch at the command line interface:

1. Run the `list -u`²²² command. This lists any schemas for which upgrades are available.
2. Run the `upgrade`²²⁵ command to install all the patches.

Install an available schema

You can install schemas using either the Schema Manager GUI or by sending Schema Manager the install instructions via the command line.

Note: If the current schema references other schemas, the referenced schemas are also installed.

In the GUI

To install schemas using the Schema Manager GUI, select the schemas you want to install and click **Apply**.

You can also select the schemas you want to install at the [Altova website](#) and generate a downloadable `.altova_xmlschemas` file. When you double-click this file, it will open Schema Manager with the schemas you wanted pre-selected. All you will now have to do is click **Apply**.

On the CLI

To install schemas via the command line, run the `install`²²² command:

```
xmlschemamanager.exe install [options] Schema+
```

where `Schema` is the schema (or schemas) you want to install or a `.altova_xmlschemas` file. A schema is referenced by an identifier of format `<name>-<version>`. (The identifiers of schemas are displayed when you run the `list`²²² command.) You can enter as many schemas as you like. For details, see the description of the `install`²²² command.

Note: On Linux or macOS, use the `sudo ./xmlschemamanager` command.

Installing a required schema

When you run an XML-schema-related command in StyleVision and StyleVision discovers that a schema it needs for executing the command is not present or is incomplete, Schema Manager will display information about the missing schema/s. You can then directly install any missing schema via Schema Manager.

In the Schema Manager GUI, you can view all previously installed schemas at any time by running Schema Manager from **Tools | Schema Manager**.

16.14.5.4 Uninstall a Schema, Reset

Uninstall a schema

You can uninstall schemas using either the Schema Manager GUI or by sending Schema Manager the uninstall instructions via the command line.

Note: If the schema you want to uninstall references other schemas, then the referenced schemas are also uninstalled.

In the GUI

To uninstall schemas in the Schema Manager GUI, clear their check boxes and click **Apply**. The selected schemas and their referenced schemas will be uninstalled.

To uninstall all schemas, click **Deselect All** and click **Apply**.

On the CLI

To uninstall schemas via the command line, run the `uninstall`²²⁴ command:

```
xmlschemamanager.exe uninstall [options] Schema+
```

where each `schema` argument is a schema you want to uninstall or a `.altova_xmlschemas` file. A schema is specified by an identifier that has a format of `<name>-<version>`. (The identifiers of schemas are displayed when you run the `list`²²² command.) You can enter as many schemas as you like. For details, see the description of the `uninstall`²²⁴ command.

Note: On Linux or macOS, use the `sudo ./xmlschemamanager` command.

Reset Schema Manager

You can reset Schema Manager. This removes all installed schemas and the cache directory.

- In the GUI, click **Reset Selection**.
- On the CLI, run the `reset`²²³ command.

After running this command, make sure to run the [initialize](#)²²¹ command in order to recreate the cache directory. Alternatively, run the [reset](#)²²³ command with the `-i` option.

Note that [reset -i](#)²²³ restores the original installation of the product, so it is recommended to run the [update](#)²²⁵ command after performing a reset. Alternatively, run the [reset](#)²²³ command with the `-i` and `-u` options.

16.14.5.5 Command Line Interface (CLI)

To call Schema Manager at the command line, you need to know the path of the executable. By default, the Schema Manager executable is installed here:

```
C:\ProgramData\Altova\SharedBetweenVersions\XMLSchemaManager.exe
```

Note: On Linux and macOS systems, once you have changed the directory to that containing the executable, you can call the executable with `sudo ./xmlschemamanager`. The prefix `./` indicates that the executable is in the current directory. The prefix `sudo` indicates that the command must be run with root privileges.

Command line syntax

The general syntax for using the command line is as follows:

```
<exec> -h | --help | --version | <command> [options] [arguments]
```

In the listing above, the vertical bar `|` separates a set of mutually exclusive items. The square brackets `[]` indicate optional items. Essentially, you can type the executable path followed by either `--h`, `--help`, or `--version` options, or by a command. Each command may have options and arguments. The list of commands is described in the following sections.

16.14.5.5.1 help

This command provides contextual help about commands pertaining to Schema Manager executable.

Syntax

```
<exec> help [command]
```

Where `[command]` is an optional argument which specifies any valid command name.

Note the following:

- You can invoke help for a command by typing the command followed by `-h` or `--help`, for example:
`<exec> list -h`
- If you type `-h` or `--help` directly after the executable and before a command, you will get general help (not help for the command), for example: `<exec> -h list`

Example

The following command displays help about the `list` command:

```
xmlschemamanager help list
```

16.14.5.5.2 info

This command displays detailed information for each of the schemas supplied as a `Schema` argument. This information for each submitted schema includes the title, version, description, publisher, and any referenced schemas, as well as whether the schema has been installed or not.

Syntax

```
<exec> info [options] Schema+
```

- The `Schema` argument is the name of a schema or a part of a schema's name. (To display a schema's package ID and detailed information about its installation status, you should use the [list](#)²²² command.)
- Use `<exec> info -h` to display help for the command.

Example

The following command displays information about the latest `DocBook-DTD` and `NITF` schemas:

```
xmlschemamanager info doc nitf
```

16.14.5.5.3 initialize

This command initializes the Schema Manager environment. It creates a cache directory where information about all schemas is stored. Initialization is performed automatically the first time a schema-cognizant Altova application is installed. You would not need to run this command under normal circumstances, but you would typically need to run it after executing the `reset` command.

Syntax

```
<exec> initialize | init [options]
```

Options

The `initialize` command takes the following options:

<code>--silent, --s</code>	Display only error messages. The default is <code>false</code> .
<code>--verbose, --v</code>	Display detailed information during execution. The default is <code>false</code> .
<code>--help, --h</code>	Display help for the command.

Example

The following command initializes Schema Manager:

```
xmlschemamanager initialize
```

16.14.5.5.4 install

This command installs one or more schemas.

Syntax

```
<exec> install [options] Schema+
```

To install multiple schemas, add the `schema` argument multiple times.

The `schema` argument is one of the following:

- A schema identifier (having a format of `<name>-<version>`, for example: `cocr-2.0`). To find out the schema identifiers of the schemas you want, run the [list](#)²²² command. You can also use an abbreviated identifier if it is unique, for example `docbook`. If you use an abbreviated identifier, then the latest version of that schema will be installed.
- The path to a `.altova_xmlschemas` file downloaded from the Altova website. For information about these files, see [Introduction to SchemaManager: How It Works](#)²¹¹.

Options

The `install` command takes the following options:

<code>--silent, --s</code>	Display only error messages. The default is <code>false</code> .
<code>--verbose, --v</code>	Display detailed information during execution. The default is <code>false</code> .
<code>--help, --h</code>	Display help for the command.

Example

The following command installs the COCR 2.0 (Country-By-Country Reporting) schema and the latest DocBook DTD:

```
xmlschemamanager install cocr-2.0 docbook
```

16.14.5.5.5 list

This command lists schemas under the management of Schema Manager. The list displays one of the following

- All available schemas
- Schemas containing in their name the string submitted as a `Schema` argument
- Only installed schemas
- Only schemas that can be upgraded

Syntax

```
<exec> list | ls [options] Schema?
```

If no `Schema` argument is submitted, then all available schemas are listed. Otherwise, schemas are listed as specified by the submitted options (see *example below*). Note that you can submit the `Schema` argument multiple times.

Options

The `list` command takes the following options:

<code>--installed, --i</code>	List only installed schemas. The default is <code>false</code> .
<code>--upgradeable, --u</code>	List only schemas where upgrades (patches) are available. The default is <code>false</code> .
<code>--help, --h</code>	Display help for the command.

Examples

- To list all available schemas, run: `xmlschemamanager list`
- To list installed schemas only, run: `xmlschemamanager list -i`
- To list schemas that contain either "doc" or "nitf" in their name, run: `xmlschemamanager list doc nitf`

16.14.5.5.6 reset

This command removes all installed schemas and the cache directory. You will be completely resetting your schema environment. After running this command, be sure to run the [initialize](#)²²¹ command to recreate the cache directory. Alternatively, run the `reset` command with the `-i` option. Since `reset -i` restores the original installation of the product, we recommend that you run the [update](#)²²⁵ command after performing a reset and initialization. Alternatively, run the `reset` command with both the `-i` and `-u` options.

Syntax

```
<exec> reset [options]
```

Options

The `reset` command takes the following options:

<code>--init, --i</code>	Initialize Schema Manager after reset. The default is <code>false</code> .
<code>--update, --u</code>	Updates the list of available schemas in the cache. The default is <code>false</code> .

<code>--silent, --s</code>	Display only error messages. The default is <code>false</code> .
<code>--verbose, --v</code>	Display detailed information during execution. The default is <code>false</code> .
<code>--help, --h</code>	Display help for the command.

Examples

- To reset Schema Manager, run: `xmlschemamanager reset`
- To reset Schema Manager and initialize it, run: `xmlschemamanager reset -i`
- To reset Schema Manager, initialize it, and update its schema list, run: `xmlschemamanager reset -i -u`

16.14.5.5.7 uninstall

This command uninstalls one or more schemas. By default, any schemas referenced by the current one are uninstalled as well. To uninstall just the current schema and keep the referenced schemas, set the option `--k`.

Syntax

```
<exec> uninstall [options] Schema+
```

To uninstall multiple schemas, add the `schema` argument multiple times.

The `schema` argument is one of the following:

- A schema identifier (having a format of `<name>-<version>`, for example: `cbcr-2.0`). To find out the schema identifiers of the schemas that are installed, run the `list -i` ²²² command. You can also use an abbreviated schema name if it is unique, for example `docbook`. If you use an abbreviated name, then all schemas that contain the abbreviation in its name will be uninstalled.
- The path to a `.altova_xmlschemas` file downloaded from the Altova website. For information about these files, see [Introduction to SchemaManager: How It Works](#) ²¹¹.

Options

The `uninstall` command takes the following options:

<code>--keep-references, --k</code>	Set this option to keep referenced schemas. The default is <code>false</code> .
<code>--silent, --s</code>	Display only error messages. The default is <code>false</code> .
<code>--verbose, --v</code>	Display detailed information during execution. The default is <code>false</code> .
<code>--help, --h</code>	Display help for the command.

Example

The following command uninstalls the CBCR 2.0 and EPUB 2.0 schemas and their dependencies:

```
xmlschemamanager uninstall cbcr-2.0 epub-2.0
```

The following command uninstalls the `eba-2.10` schema but not the schemas it references:

```
xmlschemamanager uninstall --k cbc-2.0
```

16.14.5.5.8 update

This command queries the list of schemas available from the online storage and updates the local cache directory. You should not need to run this command unless you have performed a [reset](#)²²³ and [initialize](#)²²¹.

Syntax

```
<exec> update [options]
```

Options

The `update` command takes the following options:

<code>--silent, --s</code>	Display only error messages. The default is <code>false</code> .
<code>--verbose, --v</code>	Display detailed information during execution. The default is <code>false</code> .
<code>--help, --h</code>	Display help for the command.

Example

The following command updates the local cache with the list of latest schemas:

```
xmlschemamanager update
```

16.14.5.5.9 upgrade

This command upgrades all installed schemas that can be upgraded to the latest available *patched* version. You can identify upgradeable schemas by running the [list -u](#)²²² command.

Note: The `upgrade` command removes a deprecated schema if no newer version is available.

Syntax

```
<exec> upgrade [options]
```

Options

The `upgrade` command takes the following options:

<code>--silent, --s</code>	Display only error messages. The default is <code>false</code> .
<code>--verbose, --v</code>	Display detailed information during execution. The default is <code>false</code> .

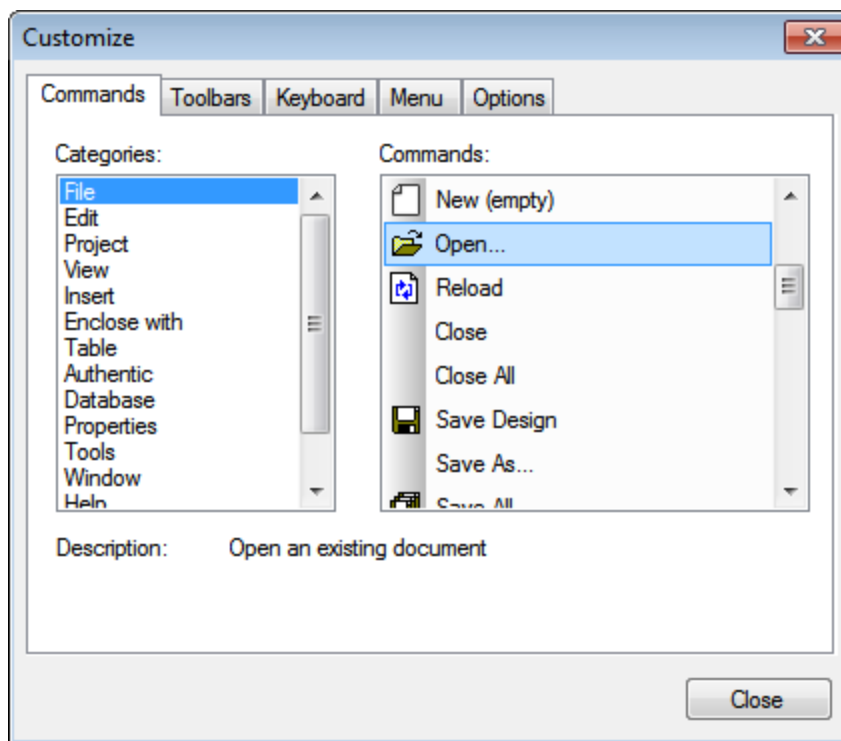
--help, --h Display help for the command.

16.14.6 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**.

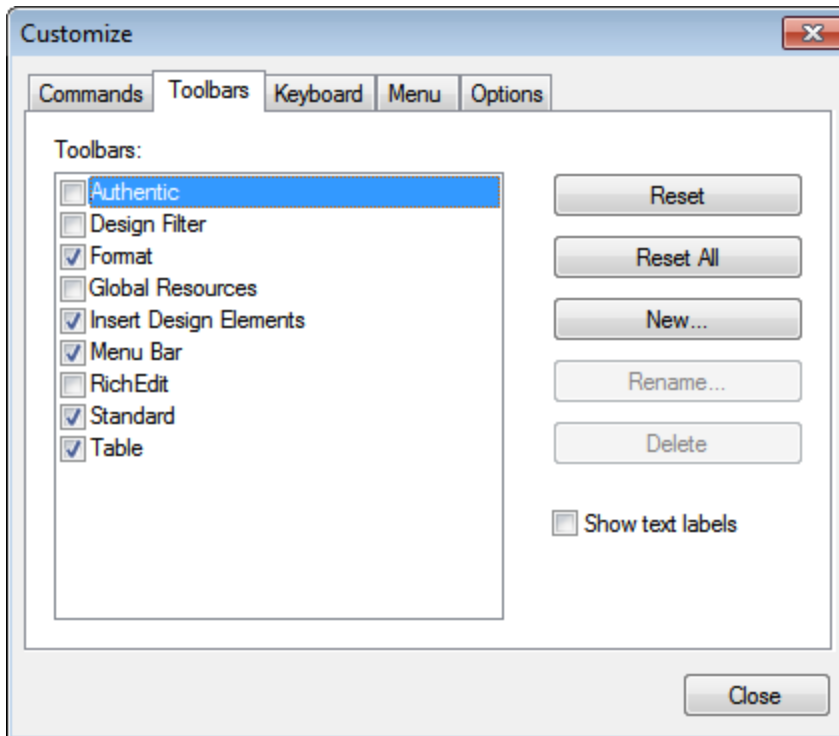
Note:

- 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.

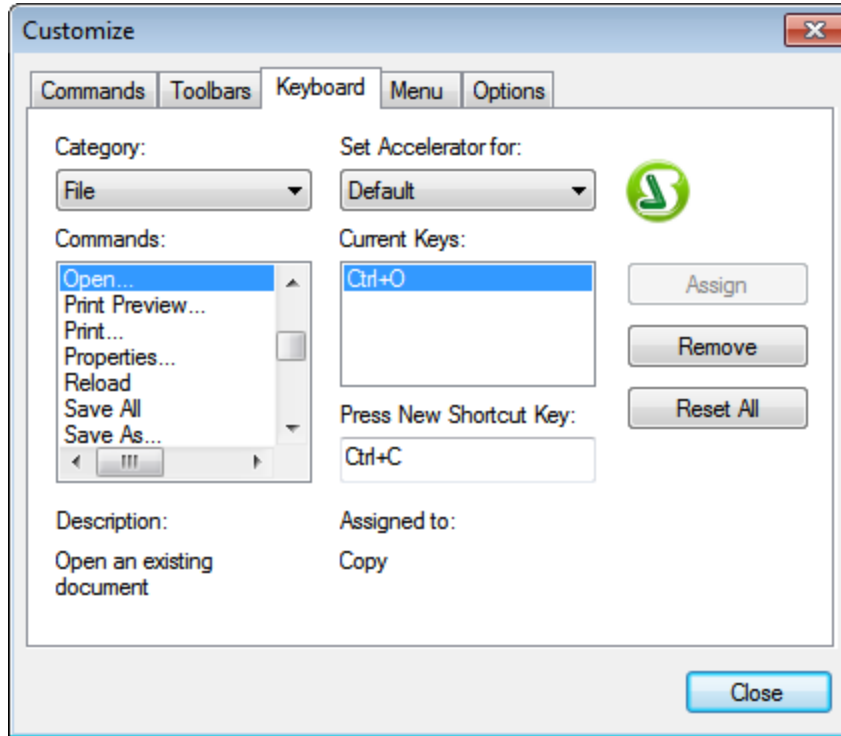
In the Toolbars tab of the Customize dialog, you can toggle a toolbar on and off by clicking in its checkbox. When a toolbar is selected (in the Toolbars tab), you can cause the text labels of that toolbar's items to be displayed by clicking the **Show text labels** check box. You can also reset a selected 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

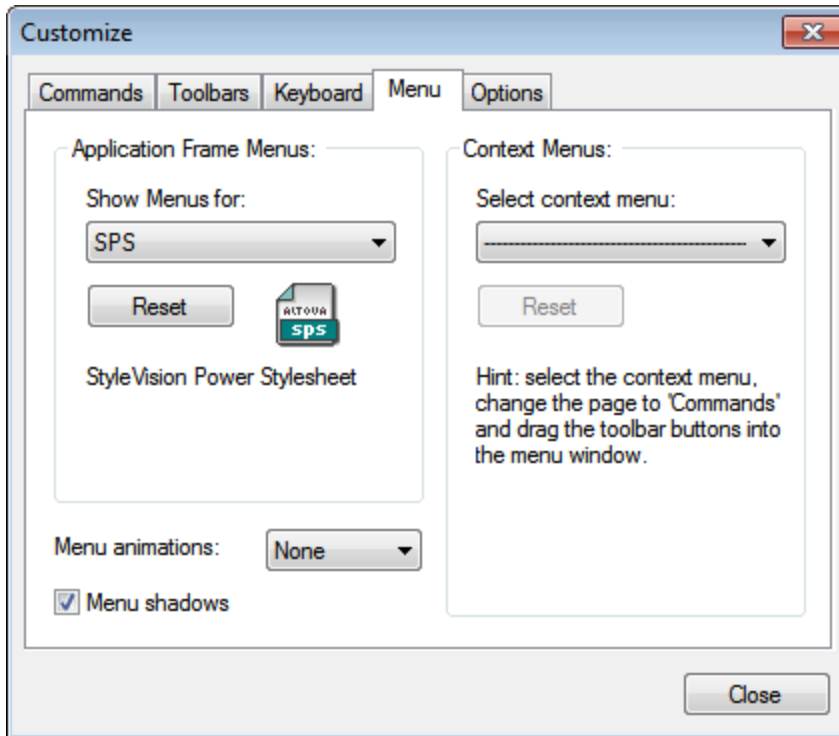
1. Click the **Reset All** button to go back to the original, installation-time shortcuts. A dialog box appears prompting you to confirm whether you want to reset all keyboard assignments.
2. Click **Yes** if you want to reset all keyboard assignments.

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).



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. In the Application Frame Menus pane, select either *Default* (which shows available menus when no document is open) or *SPS* (which shows available menus when one or more documents are open).
2. With the Customize dialog open, select (i) the menu you want to delete from the application's menu bar, or (ii) the command you want to delete from one of these menus.
3. Either (i) drag the menu from the menu bar or the menu command from the menu, or (ii) right-click the menu or menu command and select **Delete**.

To reset either of the menu bars

1. Select the menu entry you want to reset in the combo box of the Application Frame Menus pane.
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.
2. Select the **Delete** option from the popup menu or 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.

16.14.7 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.

16.14.8 Options

The **Options** command opens a dialog (*screenshot below*) in which you can specify the encoding of the HTML output file.

Design options

In the Design tab (*screenshot below*), you can set the application-wide general options for designs.

Design View Options

Maximum width of markups: px

Default Unit:

Layout Container Options

Grid Size X:

Grid Size Y:

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:

Default Additional Height:

Create templates at locations with unknown context node

Create single template

Create templates with absolute schema path

Always ask

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*.

Previews

In the Previews tab (*screenshot below*), you can select options for the previews.

- **HTML Output:** Select whether to use Edge/WebView2 or Internet Explorer as the browser for HTML previews.
- **PDF Output:** Select which PDF reader to use for the PDF preview: (i) PDF.js (an open-source PDF viewer for browsers) in Edge; (ii) Edge native (the built-in PDF reader of Microsoft Edge); or (iii) Adobe Acrobat Reader.
- **Split Preview settings:** You can set a delay for the output previews that are generated on automatic refreshes. Automatic refreshes occur when the design is modified or data is revised in Authentic View. If you set, for example, a zero delay, then the output preview will be regenerated with a zero delay each time the design or Authentic data is modified. If the delay is a short period, then the output generation will be triggered at short intervals and therefore occur at high frequency. This might result not only in slow output-preview generation, but also to intermediate outputs, such as error messages that occur if a modification in progress leads to a temporarily invalid design or invalid data. On the other hand, a long delay could lead to a long wait for the output preview. Select an optimal value according to the size of your design and data. Note that the delay specified in this setting does not apply when you first switch to the split preview or when you run a manual refresh by clicking the **Refresh** button of the Output preview pane. In both these latter cases, the output is generated immediately.

Schema 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 Options tab. By default, attributes are listed alphabetically and elements are listed in an order corresponding to the schema structure, as far as this is possible.

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: (i) Edit the file in XMLSpy; (ii) Create a new design based on the selected file; (iii) Ask the user which action to execute.

Default encoding

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" />
```

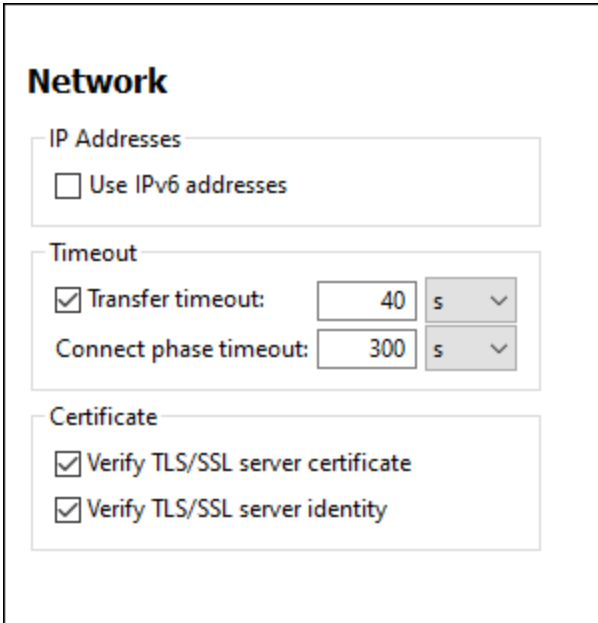
Note: These settings are the default encodings, and will be used for new SPSs. You cannot change the encoding of the currently open SPS using this dialog. To change the encoding of the currently open SPS, use the [File | Properties](#)⁷³⁸ command.

XSL options

In the meta information of HTML 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.

Network options

The **Network** section (*screenshot below*) enables you to configure important network settings.



Network

IP Addresses

Use IPv6 addresses

Timeout

Transfer timeout: 40 s

Connect phase timeout: 300 s

Certificate

Verify TLS/SSL server certificate

Verify TLS/SSL server identity

IP addresses

When host names resolve to more than one address in mixed IPv4/IPv6 networks, selecting this option causes the IPv6 addresses to be used. If the option is not selected in such environments and IPv4 addresses are available, then IPv4 addresses are used.

Timeout

- **Transfer timeout:** If this limit is reached for the transfer of any two consecutive data packages of a transfer (sent or received), then the entire transfer is aborted. Values can be specified in seconds [s] or milliseconds [ms], with the default being 40 seconds. If the option is not selected, then there is no time limit for aborting a transfer.
- **Connection phase timeout:** This is the time limit within which the connection has to be established, including the time taken for security handshakes. Values can be specified in seconds [s] or milliseconds [ms], with the default being 300 seconds. This timeout cannot be disabled.

Certificate

- **Verify TLS/SSL server certificate:** If selected, then the authenticity of the server's certificate is checked by verifying the chain of digital signatures until a trusted root certificate is reached. This option is enabled by default. If this option is not selected, then the communication is insecure, and attacks (for example, a man-in-the-middle attack) would not be detected. Note that this option does not verify that

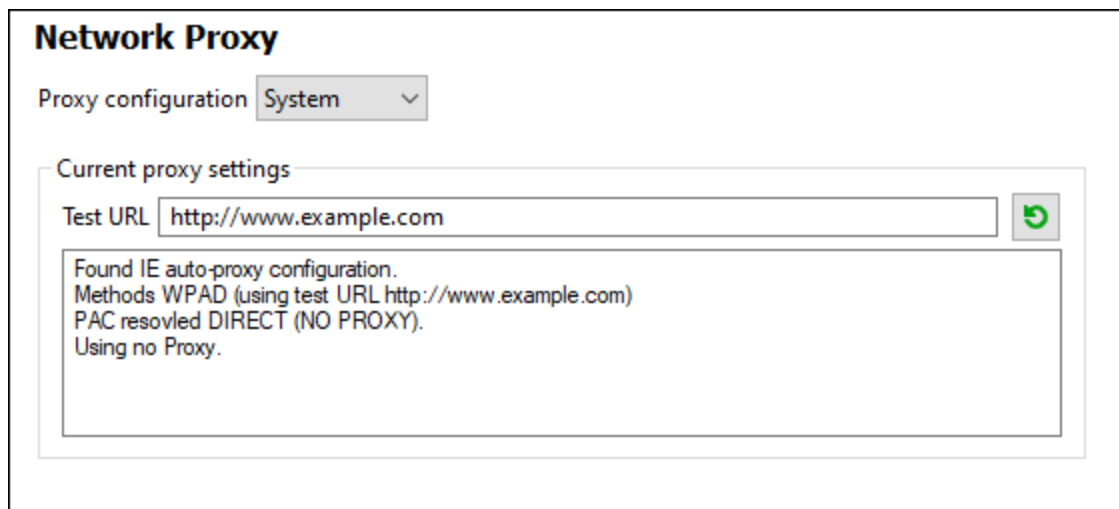
the certificate is actually for the server that is communicated with. To enable full security, both the certificate and the identity must be checked (*see next option*).

- *Verify TLS/SSL server identity*: If selected, then the server's certificate is verified to belong to the server we intend to communicate with. This is done by checking that the server name in the URL is the same as the name in the certificate. This option is enabled by default. If this option is not selected, then the server's identify is not checked. Note that this option does not enable verification of the server's certificate. To enable full security, both the certificate as well as the identity must be checked (*see previous option*).

Network Proxy options

The *Network Proxy* section enables you to configure custom proxy settings. These settings affect how the application connects to the Internet (for XML validation purposes, for example). By default, the application uses the system's proxy settings, so you should not need to change the proxy settings in most cases. If necessary, however, you can set an alternative network proxy by selecting, in the *Proxy Configuration* combo box, either *Automatic* or *Manual* to configure the settings accordingly.

Note: The network proxy settings are shared among all Altova MissionKit applications. So, if you change the settings in one application, all MissionKit applications will be affected.



Use system proxy settings

Uses the Internet Explorer (IE) settings configurable via the system proxy settings. It also queries the settings configured with `netsh.exe winhttp`.

Automatic proxy configuration

The following options are provided:

- *Auto-detect settings*: Looks up a WPAD script (`http://wpad.LOCALDOMAIN/wpad.dat`) via DHCP or DNS, and uses this script for proxy setup.
- *Script URL*: Specify an HTTP URL to a proxy-auto-configuration (`.pac`) script that is to be used for proxy setup.
- *Reload*: Resets and reloads the current auto-proxy-configuration. This action requires Windows 8 or newer, and may need up to 30s to take effect.

Manual proxy configuration

Manually specify the fully qualified host name and port for the proxies of the respective protocols. A supported scheme may be included in the host name (for example: `http://hostname`). It is not required that the scheme is the same as the respective protocol if the proxy supports the scheme.

Network Proxy

Proxy configuration **Manual** ▾

HTTP Proxy Port

Use this proxy server for all protocols

SSL Proxy Port

No Proxy for

Do not use the proxy server for local addresses

Current proxy settings

Test URL

(using test URL `http://www.example.com`)
Using no Proxy.

The following options are provided:

- *HTTP Proxy*: Uses the specified host name and port for the HTTP protocol. If *Use this proxy server for all protocols* is selected, then the specified HTTP proxy is used for all protocols.
- *SSL Proxy*: Uses the specified host name and port for the SSL protocol.
- *No Proxy for*: A semi-colon (;) separated list of fully qualified host names, domain names, or IP addresses for hosts that should be used without a proxy. IP addresses may not be truncated and IPv6 addresses have to be enclosed by square brackets (for example: `[2606:2800:220:1:248:1893:25c8:1946]`). Domain names must start with a leading dot (for example: `.example.com`).
- *Do not use the proxy server for local addresses*: If checked, adds `<local>` to the *No Proxy for* list. If this option is selected, then the following will not use the proxy: (i) `127.0.0.1`, (ii) `::1`, (iii) all host names not containing a dot character (.).

Note: If a proxy server has been set and you want to deploy a transformation to [Altova FlowForce Server](#), you must select the option *Do not use the proxy server for local addresses*.

Current proxy settings

Provides a verbose log of the proxy detection. It can be refreshed with the **Refresh** button to the right of the *Test URL* field (for example, when changing the test URL, or when the proxy settings have been changed).

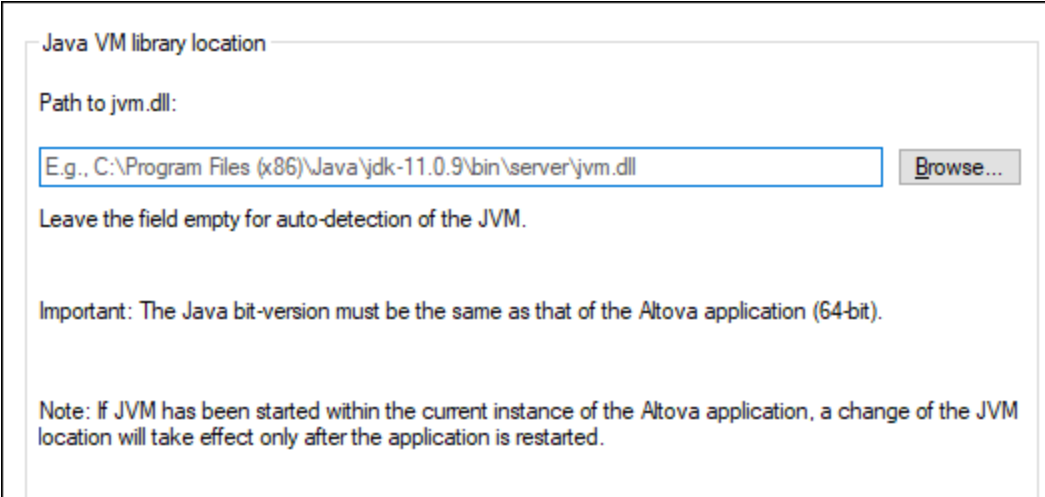
- *Test URL*: A test URL can be used to see which proxy is used for that specific URL. No I/O is done with this URL. This field must not be empty if proxy-auto-configuration is used (either through *Use*

system proxy settings or Automatic proxy configuration).

Java options

In the *Java* section (see *screenshot below*), you can optionally enter the path to a Java VM (Virtual Machine) on your file system. Note that adding a custom Java VM path is not always necessary. By default, StyleVision attempts to detect the Java VM path automatically by reading (in this order) the Windows registry and the `JAVA_HOME` environment variable. The custom path added in this dialog box will take priority over any other Java VM path detected automatically.

You may need to add a custom Java VM path, for example, if you are using a Java virtual machine which does not have an installer and does not create registry entries (e.g., Oracle's OpenJDK). You might also want to set this path if you need to override, for whatever reason, any Java VM path detected automatically by StyleVision.



Note the following:

- The Java VM path is shared between Altova desktop (not server) applications. Consequently, if you change it in one application, it will automatically apply to all other Altova applications.
- The path must point to the `jvm.dll` file from the `\bin\server` or `\bin\client` directory, relative to the directory where the JDK was installed.
- The StyleVision platform (32-bit, 64-bit) must be the same as that of the JDK.
- After changing the Java VM path, you may need to restart StyleVision for the new settings to take effect.

Changing the Java VM path affects the following areas:

- JDBC connectivity
- Java extension functions for XSLT/XPath
- Barcode support

Help

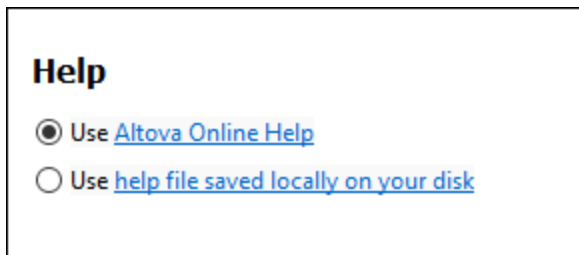
StyleVision provides Help (the user manual) in two formats:

- Online Help, in HTML format, which is available at the Altova website. In order to access the Online

Help you will need Internet access.

- A Help file in PDF format, which is installed on your machine when you install StyleVision. It is named **StyleVision.pdf** and is located in the application folder (in the Program Files folder). If you do not have Internet access, you can always open this locally saved Help file.

The Help option (*screenshot below*) enables you to select which of the two formats is opened when you click the **Help (F1)** command in the **Help** menu.



You can change this option at any time for the new selection to take effect. The links in this section (see *screenshot above*) open the respective Help format.

16.15 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.

16.16 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 web site. The Help menu also contains the [Registration dialog](#)⁸⁴⁸, 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:

- [Help](#)⁸⁴⁸
- [Activation, Order Form, Registration, Updates](#)⁸⁴⁸
- [Other Commands](#)⁸⁵¹

16.16.1 Help

The **Help (F1)** command opens the application's Help documentation (its user manual). By default, the Online Help in HTML format at the Altova website will be opened.

If you do not have Internet access or do not want, for some other reason, to access the Online Help, you can use the locally stored version of the user manual. The local version is a PDF file named `StyleVision.pdf` that is stored in the application folder (in the Program Files folder).

If you want to change the default format to open (Online Help or local PDF), do this in the Help section of the Options dialog (menu command **Tools | Options**).

16.16.2 Activation, Order Form, Registration, Updates

☐ Software Activation

License your product

After you download your Altova product software, you can license—or activate—it using either a free evaluation key or a purchased permanent license key.

- **Free evaluation license.** 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 license. Click it to get your license. When you click this button, your machine-ID will be hashed and sent to Altova via HTTPS. The license information will be sent back to the machine via an HTTP response. If the license is created successfully, a dialog to this effect will appear in your Altova application. On clicking **OK** in this dialog, the software will be activated for a period of 30 days **on this particular machine**.
- **Permanent license key.** The **Software Activation** dialog allows you 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. Your license will be sent to you by e-mail in the form of a license file, which contains your license-data.

There are three types of permanent license: *installed*, *concurrent user*, and *named user*. An

installed license unlocks the software on a single computer. If you buy an installed license for N computers, then the license allows use of the software on up to N computers. A concurrent-user license for N concurrent users allows N users to run the software concurrently. (The software may be installed on $10N$ computers.) A named-user license authorizes a specific user to use the software on up to 5 different computers. To activate your software, click **Upload a New License**, and, in the dialog that appears, enter the path to the license file, and click **OK**.

Note: For multi-user licenses, each user will be prompted to enter his or her own name.

Your license email and the different ways to license (activate) your Altova product

The license email that you receive from Altova will contain your license file as an attachment. The license file has a `.altova_licenses` file extension.

To activate your Altova product, you can do one of the following:

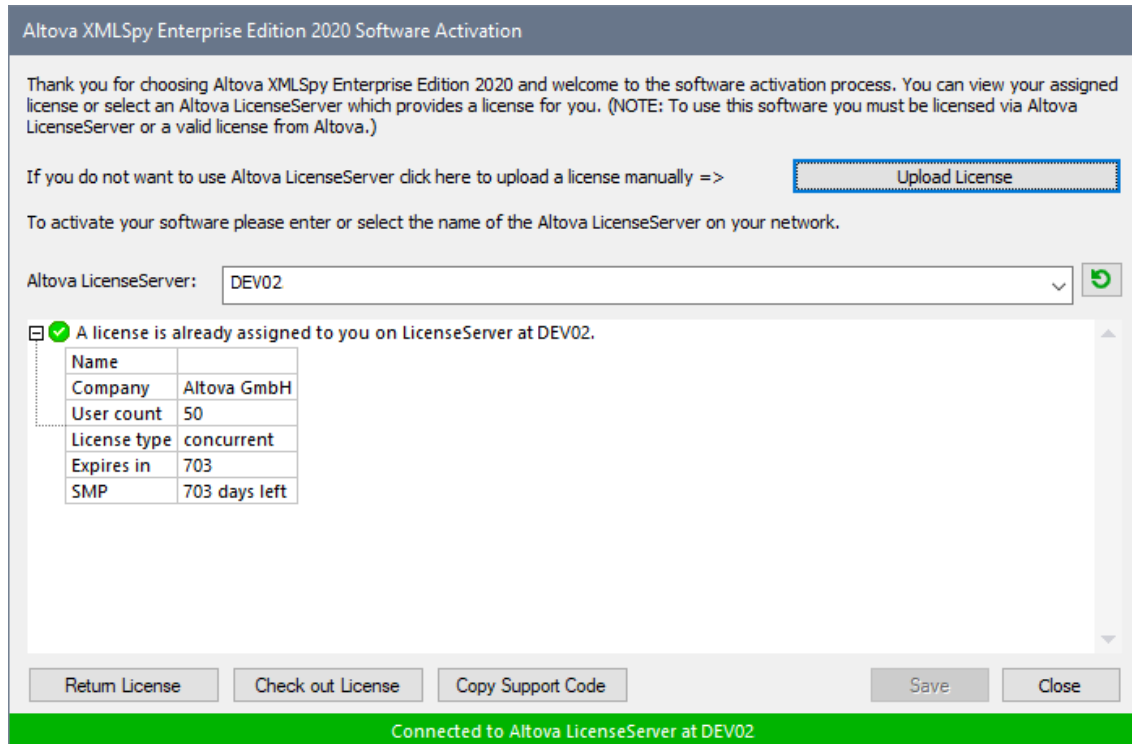
- Save the license file (`.altova_licenses`) to a suitable location, double-click the license file, enter any requested details in the dialog that appears, and finish by clicking **Apply Keys**.
- Save the license file (`.altova_licenses`) to a suitable location. In your Altova product, select the menu command **Help | Software Activation**, and then **Upload a New License**. Browse for or enter the path to the license file, and click **OK**.
- Save the license file (`.altova_licenses`) to any suitable location, and upload it from this location to the license pool of your [Altova LicenseServer](#). You can then either: (i) acquire the license from your Altova product via the product's Software Activation dialog (see below) or (ii) assign the license to the product from Altova LicenseServer. *For more information about licensing via LicenseServer, read the rest of this topic.*

You can access the **Software Activation** dialog (screenshot below) at any time by clicking the **Help | Software Activation** command.

Activate your software

You can activate the software by registering the license in the Software Activation dialog or by licensing via [Altova LicenseServer](#) (see details below).

- *Registering the license in the Software Activation dialog.* In the dialog, click **Upload a New License** and browse for the license file. Click **OK** to confirm the path to the license file and to confirm any data you entered (your name in the case of multi-user licenses). Finish by clicking **Save**.
- *Licensing via Altova LicenseServer on your network:* To acquire a license via an Altova LicenseServer on your network, click **Use Altova LicenseServer**, located at the bottom of the **Software Activation** dialog. Select the machine on which the LicenseServer you want to use has been installed. Note that the auto-discovery of License Servers works by means of a broadcast sent out on the LAN. As these broadcasts are limited to a subnet, License Server must be on the same subnet as the client machine for auto-discovery to work. If auto-discovery does not work, then type in the name of the server. The Altova LicenseServer must have a license for your Altova product in its license pool. If a license is available in the LicenseServer pool, this is indicated in the **Software Activation** dialog (see screenshot below showing the dialog in Altova XMLSpy). Click **Save** to acquire the license.



After a machine-specific (aka installed) license has been acquired from LicenseServer, it cannot be returned to LicenseServer for a period of seven days. After that time, you can return the machine license to LicenseServer (click **Return License**) so that this license can be acquired from LicenseServer by another client. (A LicenseServer administrator, however, can unassign an acquired license at any time via the administrator's Web UI of LicenseServer.) Note that the returning of licenses applies only to machine-specific licenses, not to concurrent licenses.

Check out license

You can check out a license from the license pool for a period of up to 30 days so that the license is stored on the product machine. This enables you to work offline, which is useful, for example, if you wish to work in an environment where there is no access to your Altova LicenseServer (such as when your Altova product is installed on a laptop and you are traveling). While the license is checked out, LicenseServer displays the license as being in use, and the license cannot be used by any other machine. The license automatically reverts to the checked-in state when the check-out period ends. Alternatively, a checked-out license can be checked in at any time via the **Check in** button of the **Software Activation** dialog.

To check out a license, do the following: (i) In the **Software Activation** dialog, click **Check out License** (see screenshot above); (ii) In the **License Check-out** dialog that appears, select the check-out period you want and click **Check out**. The license will be checked out. After checking out a license, two things happen: (i) The **Software Activation** dialog will display the check-out information, including the time when the check-out period ends; (ii) The **Check out License** button in the dialog changes to a **Check In** button. You can check the license in again at any time by clicking **Check In**. Because the license automatically reverts to the checked-in status after the check-out period elapses, make sure that the check-out period you select adequately covers the period during which you will be working offline.

If the license being checked out is a Installed User license or Concurrent User license, then the license is checked out to the machine and is available to the user who checked out the license. If the license being checked out is a Named User license, then the license is checked out to the Windows account of the named user. License check-out will work for virtual machines, but not for virtual desktop (in a VDI). Note that, when a Named User license is checked out, the data to identify that license check-out is stored in the user's profile. For license check-out to work, the user's profile must be stored on the local machine that will be used for offline work. If the user's profile is stored at a non-local location (such as a file-share), then the checkout will be reported as invalid when the user tries to start the Altova application.

License check-ins must be to the same major version of the Altova product from which the license was checked out. So make sure to check in a license before you upgrade your Altova product to the next major version.

Note: For license check-outs to be possible, the check-out functionality must be enabled on LicenseServer. If this functionality has not been enabled, you will get an error message to this effect when you try to check out. In this event, contact your LicenseServer administrator.

Copy Support Code

Click **Copy Support Code** to copy license details to the clipboard. This is the data that you will need to provide when requesting support via the [online support form](#).

Altova LicenseServer provides IT administrators with a real-time overview of all Altova licenses on a network, together with the details of each license as well as client assignments and client usage of licenses. The advantage of using LicenseServer therefore lies in administrative features it offers for large-volume Altova license management. Altova LicenseServer is available free of cost from the [Altova website](#). For more information about Altova LicenseServer and licensing via Altova LicenseServer, see the [Altova LicenseServer documentation](#).

☐ Order Form

When you are ready to order a licensed version of the software product, you can use either the **Purchase a Permanent License Key** button in the **Software Activation** dialog (see *previous section*) or the **Order Form** command to proceed to the secure Altova Online Shop.

☐ Registration

Opens the Altova Product Registration page in a tab of your browser. Registering your Altova software will help ensure that you are always kept up to date with the latest product information.

☐ Check for Updates

Checks with the Altova server whether a newer version than yours is currently available and displays a message accordingly.

16.16.3 Other Commands

☐ Support Center

A link to the Altova Support Center on the Internet. The Support Center provides FAQs, discussion forums

where problems are discussed, and access to Altova's technical support staff.

☐ Download Components and Free Tools

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

A link to the [Altova website](#) on the Internet. You can learn more about StyleVision, related technologies and products on the [Altova website](#).

☐ About StyleVision

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.

17 Programmers' Reference

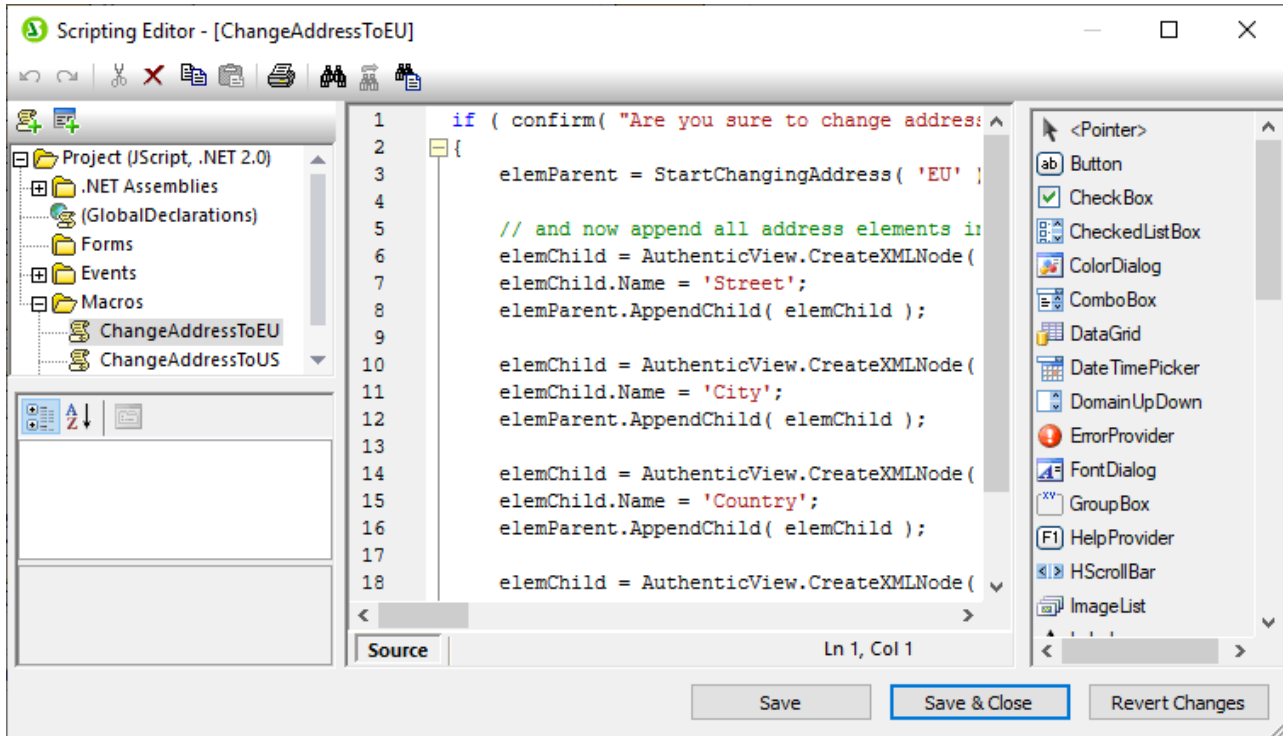
StyleVision as an Automation Server

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](#)⁸⁷⁷).

17.1 Scripting Editor

Scripting Editor is a development environment built into StyleVision from where you can customize the functionality of the Authentic View with the help of JScript or VBScript scripts. For example, you can add a new toolbar button that performs a custom task such as text formatting, or you can have StyleVision display a notification message box each time when an action is taken in the Authentic View. To make this possible, you create scripting projects—files with .asprj extension (Altova Scripting Project).



Scripting Editor

Scripting projects are portable; you can create them as part of the .sps (StyleVision Power Stylesheet) file. An Altova application that consumes the .sps file will be able to execute the macros inside the project—this can be not only StyleVision, but also XMLSpy or Authentic Desktop.

Scripting Editor requires .NET Framework 2.0 or later to be installed before StyleVision is installed.

17.1.1 Creating a Scripting Project

All scripts and scripting information created in the Scripting Editor are stored in Altova Scripting Projects (.asprj files). A scripting project may contain macros, application event handlers, and forms (which can have their own event handlers). In addition, you can add global variables and functions to a "Global Declarations" script—this makes such variables and functions accessible across the entire project. To start a new project:

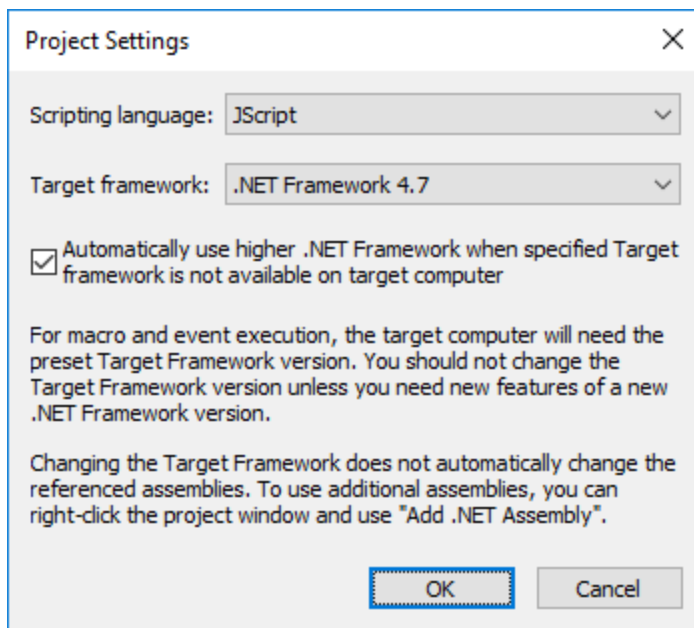
1. Create or open the .sps file that the scripting project should be part of.

2. Do one of the following:
 - On the **Authentic** menu, click **Authentic Scripts**.
 - Click the **Design** tab at the base of the main window, and then select **Authentic Script**.

Each .sps file can have a single scripting project assigned to it. To open an existing scripting project associated with an .sps file, follow the same instructions as above. This opens the Scripting Editor window. As you edit the scripting project, you can temporarily save changes in memory, by clicking the **Save** button at the base of Scripting Editor. To save the scripting project to disk, first click **Save & Close** in Scripting Editor, and then save the .sps file.

The languages supported for use in a scripting project are JScript and VBScript (not to be confused with Visual Basic, which is not supported). These scripting engines are available by default on Windows and have no special requirements to run. You can select a scripting language as follows:

1. Right-click the **Project** item in the upper-left pane, and select **Project settings** from the context menu.
2. Select a language (JScript or VBScript), and click **OK**.



From the Project settings dialog box above, you can also change the target .NET Framework version. This is typically necessary if your scripting project requires features available in a newer .NET Framework version. Note that any clients using your scripting project will need to have the same .NET Framework version installed (or a later compatible version).

By default, a scripting project references several .NET assemblies, like `System`, `System.Data`, `System.Windows.Forms`, and others. If necessary, you can import additional .NET assemblies, including assemblies from .NET Global Assembly Cache (GAC) or custom .dll files. You can import assemblies as follows:

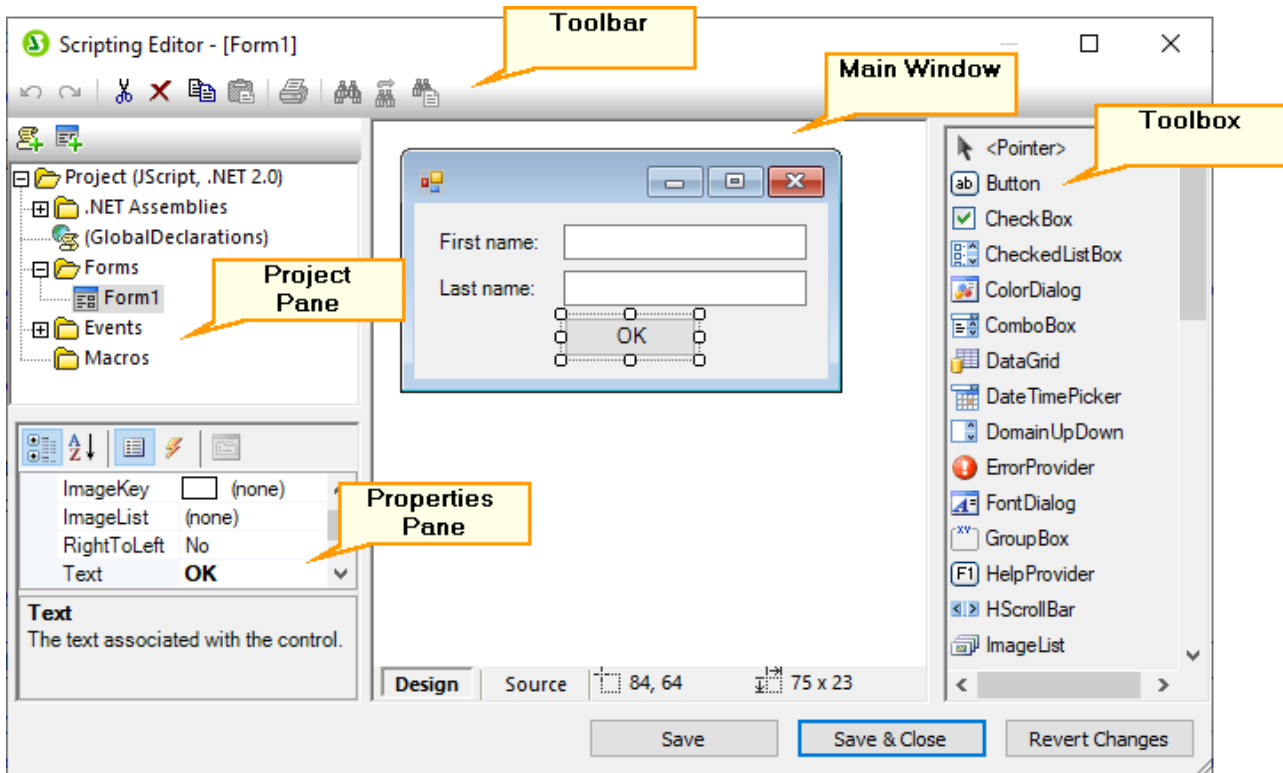
1. Statically, by adding them manually to the project. Right-click **Project** in the top-left pane, and select **Add .NET Assembly** from the context menu.
2. Dynamically, at runtime, by calling the `CLR.LoadAssembly` ⁸⁶⁹ command from the code.

The next sections focus on the parts that your scripting project may need: global declarations, macros, forms, and events. For further information about scripting and examples of scripting projects, see [Authentic Scripting](#)⁶⁵⁰.

17.1.1.1 Overview of the Environment

The Scripting Editor consists of the following parts:

- Toolbar
- Project pane
- Properties pane
- Main window
- Toolbox



The scripting project is saved when you save the .sps file of which the scripting project is part of. The **Save** button illustrated above creates only a temporary in-memory copy.

Toolbar

The toolbar includes the **Undo (Ctrl+Z)** and **Redo (Ctrl+Y)** commands and editor commands (**Copy**, **Cut**, **Delete**, **Paste**). When editing source code, the **Find** and **Replace** commands are additionally available, as well as the **Print** command.

Project pane

The project pane helps you view and manage the structure of the project. A scripting project consists of several components that can work together and may be created in any order:

- A "*Global Declarations*" script. As the name suggests, this script stores information available globally across the project. You can declare in this script any variables or functions that you need to be available in all forms, event handler scripts, and macros.
- *Forms*. Forms are typically necessary to collect user input, or provide some informative dialog boxes. A form is invoked by a call to it either within a function (in the Global Declarations script) or directly in a macro.
- *Events*. The "Events" folder displays StyleVision application events provided by the COM API. To write a script that will be executed when an event occurs, double-click any event, and then type the handling code in the editor. The application events should not be confused with form events; the latter are handled at form level, as further detailed below.
- *Macros*. A macro is a script that can be invoked either on demand from a context menu or be executed automatically when StyleVision starts. Macros do not have parameters or return values. A macro can access all variables and functions declared in the Global Declarations script and it can also display forms.

Right-click any of the components to see the available context menu commands and their shortcuts. Double-click any file (such as a form or a script) to open it in the main window.

The toolbar buttons provide the following quick commands:



New macro Adds a new macro to the project, in the **Macros** directory.







New form Adds a new form to the project, in the **Forms** directory.

Properties pane

The Properties pane is very similar to the one in Visual Studio. It displays the following:

- Form properties, when a form is selected
- Object properties, when an object in a form is selected
- 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 the **Properties**  or **Events**  buttons, respectively.

The **Categorized**  and **Alphabetical**  icons 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 pane.

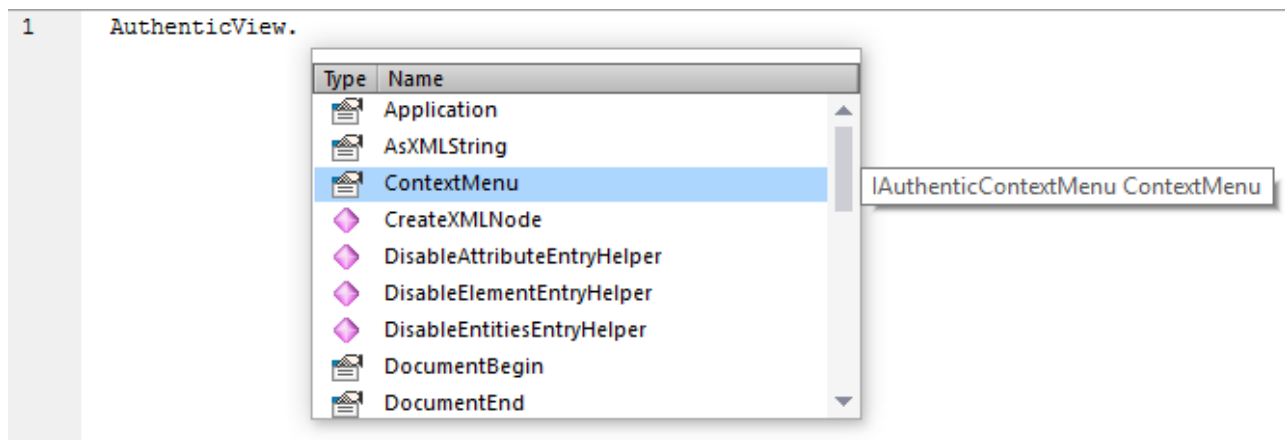
Main window

The main window is the working area where you can enter source code or modify the design of the form. When editing forms, you can work in two tabs: the **Design** tab and the **Source** tab. The **Design** tab shows the layout of the form, while the **Source** tab contains the source code such as handler methods for the form events.

The source code editor provides code editing aids such as syntax coloring, source code folding, highlighting of starting and ending braces, zooming, autocompletion suggestions, bookmarks.

Autocompletion suggestions

JScript and VBScript are untyped languages, so autocompletion is limited to COM API names and StyleVision built-in [commands](#)⁸⁶⁶. The full method or property signature is shown next to the autocompletion entry helper.



Placing the mouse over a known method or property displays its signature (and documentation if available), for example:

The auto-completion entry helper is normally shown automatically during editing, but it can also be obtained on demand by pressing **Ctrl+Space**.

Bookmarks

- To set or remove a bookmark, click inside a line, and then press **Ctrl+F2**
- To navigate to the next bookmark, press **F2**
- To navigate to the previous bookmark, press **Shift+F2**
- To delete all bookmarks, press **Ctrl+Shift+F2**

Zooming in/out

- To zoom in or out, hold the **Ctrl** key pressed and then press the "+" or "-" keys or rotate the mouse wheel.

Text view settings

To trigger text settings, right-click inside the editor, and select **Text View Settings** from the context menu.

Font settings

To change the font, right-click inside the editor, and select **Text View Font** from the context menu.

Toolbox

The Toolbox contains all the objects that are available for designing forms, such as buttons, text boxes, combo boxes, and so on.

To add a Toolbox item to a form:

1. Create or open a form and make sure that the **Design** tab is selected.
2. Click the Toolbox object (for example, **Button**), and then click at the location in the form where you wish to insert it. Alternatively, drag the object directly onto the form.

Some objects such as `Timer` are not added to the Form but are created 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 from the Properties pane. For an example of handling tray components from the code, see [Handling form events](#)⁸⁶¹.

You can also add registered ActiveX controls to the form. To do this, right-click the Toolbox area and select **Add ActiveX Control** from the context menu.

17.1.1.2 Global Declarations

The "Global Declarations" script is present by default in any scripting project; you do not need to create it explicitly. Any variables or functions that you add to this script are considered global across the entire project. Consequently, you can refer to such variables and functions from any of the project's macros and events. The following is an example of a global declarations script that imports the `System.Windows.Forms` namespace into the project. To achieve that, the code below invokes the `CLR.Import` command built into Scripting Editor.

```
// import System.Windows.Forms namespace for all macros, forms and events:  
CLR.Import( "System.Windows.Forms" );
```

Note: Every time a macro is executed or an event handler is called, the global declarations are re-initialized.

17.1.1.3 Macros

Macros are scripts that contain JScript (or VBScript, depending on your project's language) statements, such as variable declarations and functions.

If your projects should use macros, you can add them as follows: right-click inside the Project pane, select **Add Macro** from the context menu, and then enter the macro's code in the main form. The code of a macro could be as simple as an alert, for example:

```
alert("Hello, I'm a macro!");
```

More advanced macros can contain variables and local functions. Macros can also contain code that invokes forms from the project. The listing below illustrates an example of a macro that shows a form. It is assumed that this form has already been created in the "Forms" folder and has the name "SampleForm", see also [Forms](#)⁸⁶¹.

```
// display a form
ShowForm( "SampleForm" );
```

In the code listing above, `ShowForm` is a command built into Scripting Editor. For reference to other similar commands that you can use to work with forms and .NET objects, see the [Built-in Commands](#) ⁸⁶⁶.

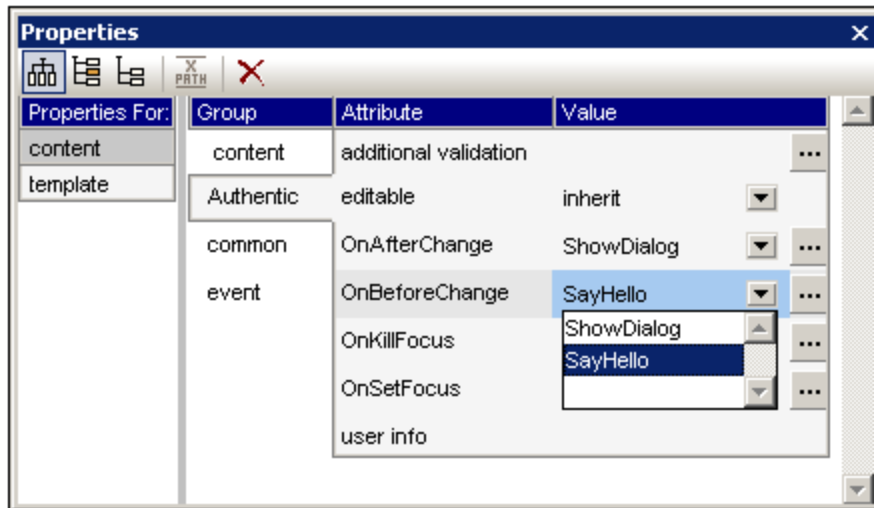
Only one macro can be run at a time. After a macro (or event) is executed, the script is closed and global variables lose their values.

To run a macro directly in Script Editor, click **Run Macro** . To debug a macro using the Visual Studio debugger, click **Debug Macro** .

Running macros

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.



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.

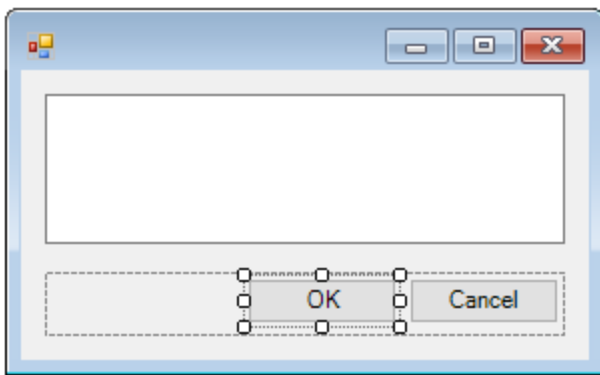
See the section, [Authentic Scripting >> Macros](#) ⁶⁵³, for a detailed description of how to use macros in a StyleVision design document.

17.1.1.4 Forms

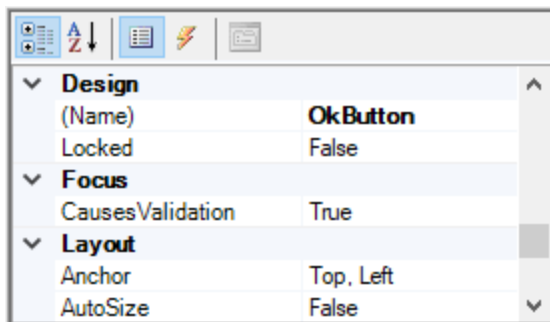
Forms are particularly useful if you need to collect input data from users or display data to users. A form can contain miscellaneous controls to facilitate this, such as buttons, check boxes, combo boxes, and so on.

To add a form, right-click inside the Project pane, and then select **Add Form** from the context menu. To add a control to a form, drag it from the Toolbox available to the right side of Scripting Editor and drop it onto the form.

You can change the position and size of the controls directly on the form, by using the handles that appear when you click any control, for example:




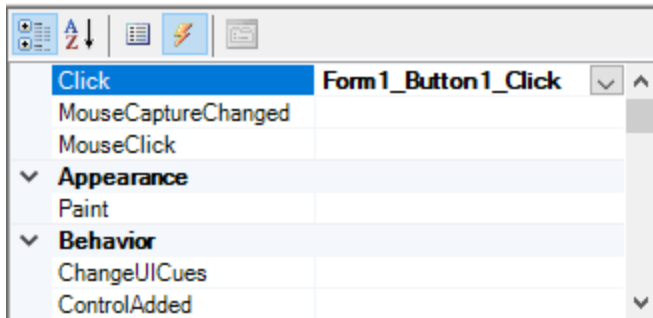
All form controls have properties that you can easily adjust in the Properties pane. To do this, first select the control on the form, and then edit the required properties in the Properties pane.



Handling form events

Each form control also exposes various events to which your scripting project can bind. For example, you might want to invoke some StyleVision COM API method whenever a button is clicked. To create a function that binds to a form event, do the following:

1. In the Properties pane, click **Events** .
2. In the **Action** column, double-click the event where you need the method (for example, in the image below, the handled event is "Click").



You can also add handler methods by double-clicking a control on the form. For example, double-clicking a button in the form design generates a handler method for the "Click" event of that button.

Once the body of the handler method is generated, you can type code that handles this event, for example:

```
//Occurs when the component is clicked.
function MyForm_ButtonClick( objSender, e_EventArgs )
{
    alert("A button was clicked");
}
```

To display a work-in-progress form detached from the Scripting Editor, right-click the form in the Project window, and select **Test Form** from the context menu. Note that the **Test Form** command just displays the form; the form's events (such as button clicks) are still disabled. To have the form react to events, call it from a macro, for example:

```
// Instantiate and display a form
ShowForm( "SampleForm" );
```

Accessing form controls

You can access any components on a form from your code by using field access syntax. For example, suppose there is a form designed as follows:

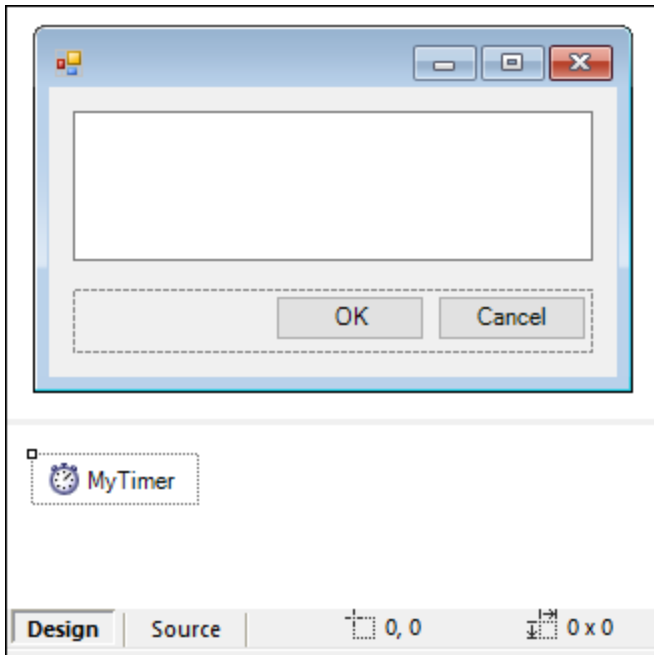
```
// MyForm
//   ButtonPanel
//     OkButton
//     CancelButton
//   TextEditor
//     AxMediaPlayer1
// TrayComponents
//   MyTimer
```

The code below shows how to instantiate the form, access some of its controls using field access syntax, and then display the form:

```
// Instantiate the form
var objForm = CreateForm("MyForm");
```

```
// Disable the OK button
objForm.ButtonPanel.OkButton.Enabled = false;
// Change the text of TextEditor
objForm.TextEditor.Text = "Hello";
// Show the form
objForm.ShowDialog();
```

When you add certain controls such as timers to the form, they are not displayed on the form; instead, they are shown as tray components at the base of the form design, for example:



To access controls from the tray, use the `GetTrayComponent` method on the form object, and supply the name of the control as argument. In this example, to get a reference to `MyTimer` and enable it, use the following code:

```
var objTimer = objForm.GetTrayComponent("MyTimer");
objTimer.Enabled = true;
```

For ActiveX Controls, you can access the underlying COM object via the `OCX` property:

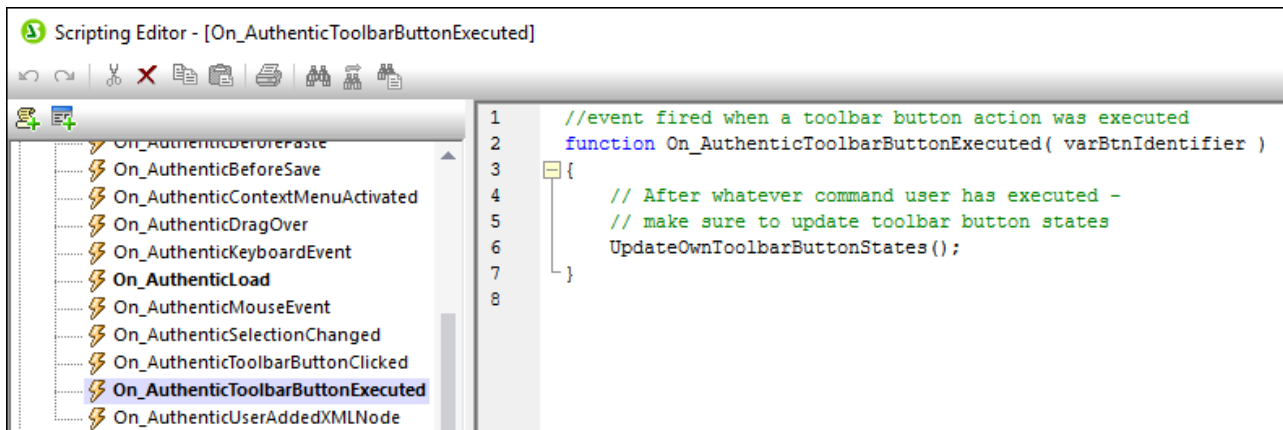
```
var ocx = lastform.AxMediaPlayer1.OCX; // get underlying COM object
ocx.enableContextMenu = true;
ocx.URL = "mms://apasf.apa.at/fm4_live_worldwide";
```

17.1.1.5 Events

Your scripting project may optionally include scripts that handle events in the Authentic View, such as cutting, copying, or pasting text, or opening a context menu. These events are provided by the StyleVision COM API,

and you can find them in the "Events" folder of your scripting project. Note that these events are StyleVision-specific, as opposed to form events.

To create an event handler script, right-click an event, and select **Open** from the context menu (or double-click the event). The event handler script is displayed in the main window, where you can start editing it. For example, the event handler illustrated below calls a function when a toolbar button is executed. The function definition is in the [Global Declarations](#) ⁸⁵⁹ script, so it is available to the event handler script.



The name of the event handler function must not be changed; otherwise, the event handler script will not be called.

You can optionally define local variables and helper functions within event handler scripts, for example:

```

var local;

function On_AuthenticLoad( )
{
    local = "OnAuthenticLoad";
    Helper();
}

function Helper()
{
    alert("I'm a helper function for " + local);
}

```

17.1.1.6 JScript Programming Tips

Below are a few JScript programming tips that you may find useful while developing a scripting project in StyleVision Scripting Editor.

Out parameters

Out parameters from methods of the .NET Framework require special variables in JScript. For example:


```
var dictionary =
CLR.Create("System.Collections.Generic.Dictionary<System.String, System.String>");
dictionary.Add("1", "A");
dictionary.Add("2", "B");

// use JScript method to access out-parameters
var strOut = new Array(1);
if ( dictionary.TryGetValue("1", strOut) ) // TryGetValue will set the out parameter
    alert( strOut[0] ); // use out parameter
```

Integer arguments

.NET Methods that require integer arguments should not be called directly with JScript number objects which are floating point values. For example, instead of:

```
var objCustomColor = CLR.Static("System.Drawing.Color").FromArgb(128,128,128);
```

use:

```
var objCustomColor =
CLR.Static("System.Drawing.Color").FromArgb(Math.floor(128),Math.floor(128),Math.floor(128));
```

Iterating .NET collections

To iterate .NET collections, the JScript Enumerator as well as the .NET iterator technologies can be used, for example:

```
// iterate using the JScript iterator
var itr = new Enumerator( coll );
for ( ; !itr.atEnd(); itr.moveNext() )
    alert( itr.item() );

// iterate using the .NET iterator
var itrNET = coll.GetEnumerator();
while( itrNET.MoveNext() )
    alert( itrNET.Current );
```

.NET templates

.NET templates can be instantiated as shown below:

```
var coll = CLR.Create( "System.Collections.Generic.List<System.String>" );
```

or

```
CLR.Import( "System" );
CLR.Import( "System.Collections.Generic" );
```

```
var dictionary = CLR.Create( "Dictionary<String,Dictionary<String,String>>" );
```

.NET enumeration values

.NET enumeration values are accessed as shown below:

```
var enumValStretch = CLR.Static( "System.Windows.Forms.ImageLayout" ).Stretch;
```

Enumeration literals

The enumeration literals from the StyleVision API can be accessed as shown below (there is no need to know their numerical value).

```
objExportXMIFileDlg.XMIType = eXMI21ForUML23;
```

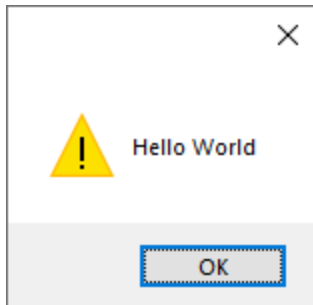
17.1.2 Built-in Commands

This section provides reference to all the commands you can use in the StyleVision Scripting Editor.

- [alert](#) ⁸⁶⁶
- [confirm](#) ⁸⁶⁷
- [CLR.Create](#) ⁸⁶⁸
- [CLR.Import](#) ⁸⁶⁸
- [CLR.LoadAssembly](#) ⁸⁶⁹
- [CLR.ShowImports](#) ⁸⁷⁰
- [CLR.ShowLoadedAssemblies](#) ⁸⁷⁰
- [CLR.Static](#) ⁸⁷¹
- [CreateForm](#) ⁸⁷²
- [doevents](#) ⁸⁷³
- [lastform](#) ⁸⁷³
- [prompt](#) ⁸⁷⁴
- [ShowForm](#) ⁸⁷⁴
- [watchdog](#) ⁸⁷⁵

17.1.2.1 alert

Displays a message box that shows a given message and the "OK" button. To proceed, the user will have to click "OK".



Signature

For JScript, the signature is:

```
alert(strMessage : String) -> void
```

For VBScript, the signature is:

```
MsgBox(strMessage : String) -> void
```

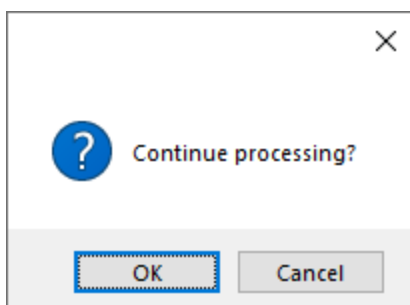
Example

The following JScript code displays a message box with the text "Hello World".

```
alert("Hello World");
```

17.1.2.2 confirm

Opens a dialog box that shows a given message, a confirmation button, and a cancel button. The user will have to click either "OK" or "Cancel" to proceed. Returns a Boolean that represents the user's answer. If the user clicked "OK", the function returns **true**; if the user clicked "Cancel", the function returns **false**.



Signature

```
confirm(strMessage : String) -> result : Boolean
```

Example (JScript)

```
if ( confirm( "Continue processing?" ) == false )
    alert("You have cancelled this action");
```

Example (VBScript)

```
If ( confirm( "Continue processing?" ) = false ) Then
    MsgBox("You have cancelled this action")
End If
```

17.1.2.3 CLR.Create

Creates a new .NET object instance of the type name supplied as argument. If more than one argument is passed, the successive arguments are interpreted as the arguments for the constructor of the .NET object. The return value is a reference to the created .NET object

Signature

```
CLR.Create(strTypeNameCLR : String, constructor arguments ... ) -> object
```

Example

The following JScript code illustrates how to create instances of various .NET classes.

```
// Create an ArrayList
var objArray = CLR.Create("System.Collections.ArrayList");
// Create a ListViewItem
var newItem = CLR.Create( "System.Windows.Forms.ListViewItem", "NewItemText" );
// Create a List<string>
var coll = CLR.Create( "System.Collections.Generic.List<System.String>" );
// Import required namespaces and create a Dictionary object
CLR.Import( "System" );
CLR.Import( "System.Collections.Generic" );
var dictionary = CLR.Create( "Dictionary<String, Dictionary<String, String >>" );
```

17.1.2.4 CLR.Import

Imports a namespace. This is the scripting equivalent of C# `using` and VB.Net `imports` keyword. Calling `CLR.Import` makes it possible to leave out the namespace part in subsequent calls like `CLR.Create()` and `CLR.Static()`.

Note: Importing a namespace does not add or load the corresponding assembly to the scripting project. You can add assemblies to the scripting project dynamically (at runtime) in the source code by calling [CLR.LoadAssembly](#)⁸⁶⁹.

Signature

```
CLR.Import(strNamespaceCLR : String) -> void
```

Example

Instead of having to use fully qualified namespaces like:

```
if ( ShowForm( "FormName" ) == CLR.Static( "System.Windows.Forms.DialogResult" ).OK )
{
    var sName = lastform.textboxFirstName.Text + " " + lastform.textboxLastName.Text;
    CLR.Static( "System.Windows.Forms.MessageBox" ).Show( "Hello " + sName );
}
```

One can import namespaces first and subsequently use the short form:

```
CLR.Import( "System.Windows.Forms" );

if ( ShowForm( "FormName" ) == CLR.Static( "DialogResult" ).OK )
{
    var sName = lastform.textboxFirstName.Text + " " + lastform.textboxLastName.Text;
    CLR.Static( "MessageBox" ).Show( "Hello " + sName );
}
```

17.1.2.5 CLR.LoadAssembly

Loads the .NET assembly with the given long assembly name or file path. Returns Boolean **true** if the assembly could be loaded; **false** otherwise.

Signature

```
CLR.LoadAssembly(strAssemblyNameCLR : String, showLoadErrors : Boolean) -> result : Boolean
```

Example

The following JScript code attempts to set the clipboard text by loading the required assembly dynamically.

```
// set clipboard text (if possible)
// System.Windows.Clipboard is part of the PresentationCore assembly, so load this
// assembly first:
if ( CLR.LoadAssembly( "PresentationCore, Version=3.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35", true ) )
```

```
{  
    var clipboard = CLR.Static( "System.Windows.Clipboard" );  
    if ( clipboard != null )  
        clipboard.SetText( "HelloClipboard" );  
}
```

17.1.2.6 CLR.ShowImports

Opens a message box that shows the currently imported namespaces. The user will have to click "OK" to proceed.

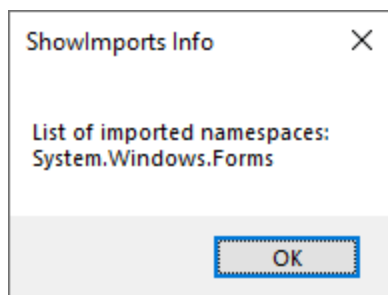
Signature

```
CLR.ShowImports() -> void
```

Example

The following JScript code first imports a namespace, and then displays the list of imported namespaces:

```
CLR.Import( "System.Windows.Forms" );  
CLR.ShowImports();
```



17.1.2.7 CLR.ShowLoadedAssemblies

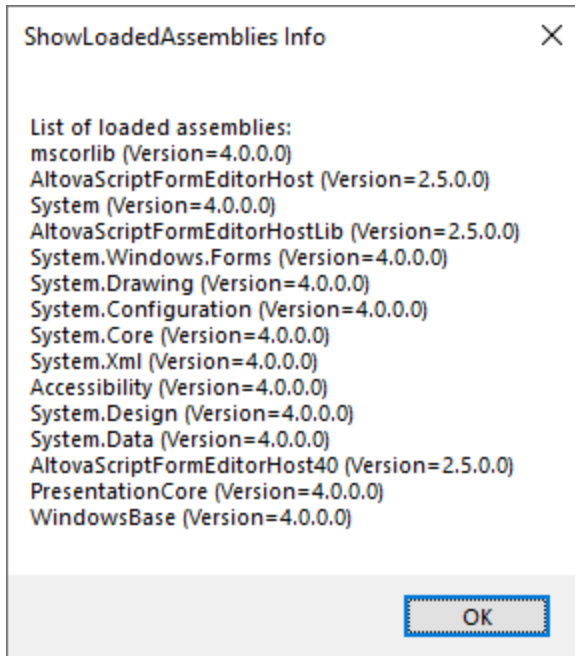
Opens a message box that shows the currently loaded assemblies. The user will have to click "OK" to proceed.

Signature

```
CLR.ShowLoadedAssemblies() -> void
```

Example

```
CLR.ShowLoadedAssemblies();
```



17.1.2.8 CLR.Static

Returns a reference to a static .NET object. You can use this function to get access to .NET types that have no instances and contain only static members.

Signature

```
CLR.Static(strTypeNameCLR : String) -> object
```

Example (JScript)

```
// Get the value of a .NET Enum into a variable
var enumValStretch = CLR.Static( "System.Windows.Forms.ImageLayout" ).Stretch

// Set the value of the Windows clipboard
var clipboard = CLR.Static( "System.Windows.Clipboard" );
clipboard.SetText( "HelloClipboard" );

// Check the buttons pressed by the user on a dialog box
if ( ShowForm( "FormName" ) == CLR.Static( "System.Windows.Forms.DialogResult" ).OK )
    alert( "ok" );
else
    alert( "cancel" );
```

17.1.2.9 CreateForm

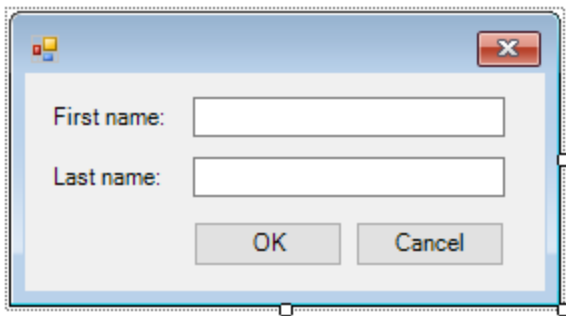
Instantiates the `Form` object identified by the name supplied as argument. The form must exist in the "Forms" folder of the scripting project. Returns the form object (`System.Windows.Forms.Form`) corresponding to the given name, or `null` if no form with such name exists.

Signature

```
CreateForm (strFormName : String) -> System.Windows.Forms.Form | null
```

Example

Let's assume that a form called "FormName" exists in the scripting project.



The following JScript code instantiates the form with some default values and displays it to the user.

```
var myForm = CreateForm( "FormName" );
if ( myForm != null )
{
    myForm.textboxFirstName.Text = "Daniela";
    myForm.textboxLastName.Text = "Heidegger";
    var dialogResult = myForm.ShowDialog();
}
```

The `dialogResult` can subsequently be evaluated as follows:

```
if ( dialogResult == CLR.Static( "System.Windows.Forms.DialogResult" ).OK )
    alert( "ok" );
else
    alert( "cancel" );
```

Note: The code above will work only if the **DialogResult** property of the "OK" and "Cancel" buttons is set correctly from the Properties pane (for example, it must be **OK** for the "OK" button).

17.1.2.10 doevents

Processes all Windows messages currently in the message queue.

Signature

```
doevents() -> void
```

Example (JScript)

```
for ( i=0; i < nLongLastingProcess; ++i )
{
    // do long lasting process

    doevents(); // process Windows messages; give UI a chance to update
}
```

17.1.2.11 lastform

This is a global field that returns a reference to the last form object that was created via `CreateForm()` or `ShowForm()`.

Signature

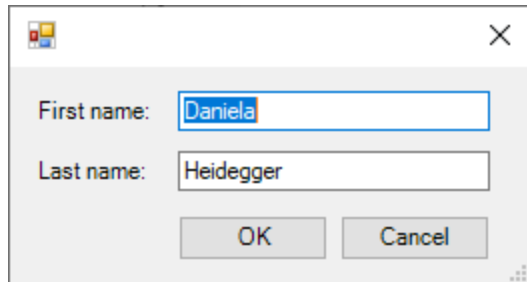
```
lastform -> formObj : System.Windows.Forms.Form
```

Example

The following JScript code shows the form "FormName" as a dialog box.

```
CreateForm( "FormName" );
if ( lastform != null )
{
    lastform.textboxFirstName.Text = "Daniela";
    lastform.textboxLastName.Text = "Heidegger";
    var dialogResult = lastform.ShowDialog();
}
```

The values of both textbox controls are initialized with the help of `lastform`.



17.1.2.12 prompt

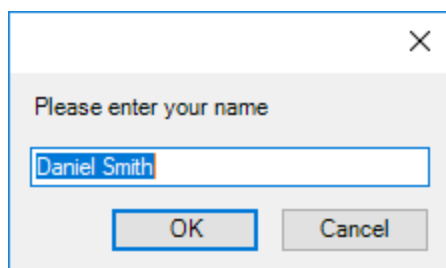
Opens a dialog box that shows a message and a textbox control with a default answer. This can be used to let the user input a simple string value. The return value is a string that contains the textbox value or null if the user selected "Cancel".

Signature

```
prompt(strMessage : String, strDefault : String) -> val : String
```

Example

```
var name = prompt( "Please enter your name", "Daniel Smith" );  
if ( name != null )  
    alert( "Hello " + name + "!" );
```



17.1.2.13 ShowForm

Instantiates a new form object from the given form name and immediately shows it as dialog box. The return value is an integer that represents the generated `DialogResult` (`System.Windows.Forms.DialogResult`). For the list of possible values, refer to the documentation of the `DialogResult` Enum (<https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.dialogresult?view=netframework-4.8>).

Signature

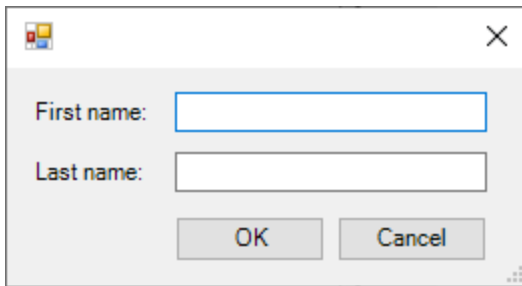
```
ShowForm(strFormName : String) -> result : Integer
```

Example

The following JScript code

```
var dialogResult = ShowForm( "FormName" );
```

Shows the form "FormName" as a dialog box:



The `DialogResult` can subsequently be evaluated, for example:

```
if ( dialogResult == CLR.Static( "System.Windows.Forms.DialogResult" ).OK )  
    alert( "ok" );  
else  
    alert( "cancel" );
```

Note: The code above will work only if the **DialogResult** property of the "OK" and "Cancel" buttons is set correctly from the Properties pane (for example, it must be **OK** for the "OK" button).

17.1.2.14 watchdog

Long running CPU-intensive scripts may ask the user if the script should be terminated. The `watchdog()` method is used to disable or enable this behavior. By default, the watchdog is enabled.

Calling `watchdog(true)` can also be used to reset the watchdog. This can be useful before executing long running CPU-intensive tasks to ensure they have the maximum allowed script processing quota.

Signature

```
watchdog(bEnable : boolean) -> void
```

Example

```
watchdog( false ); // disable watchdog - we know the next statement is CPU intensive but  
it will terminate for sure  
doCPUIntensiveScript();  
watchdog( true ); // re-enable watchdog
```

17.2 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](#)⁸⁷⁷ and in the [Overview](#)⁸⁷⁸ 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](#)⁸⁵⁴.
- Via an ActiveX Control, which is available if the [integration package](#)¹⁰⁰² is installed. For more information, see the section [ActiveX Integration](#)¹⁰⁰².

External programs

In the [Overview](#)⁸⁷⁸ 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](#)⁸⁷⁸, 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#](#)⁸⁸² 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](#)⁸⁹¹: 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.

17.2.1 Overview

This overview of the Application API is organized as follows:

- [The Object Model](#)⁸⁷⁸ describes the relationships between the objects of the Application API.
- [Programming Languages](#)⁸⁷⁸ 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.

17.2.1.1 Object Model

The starting point for every application which uses the Application API is the [Application](#)⁸⁹⁸ object. This object contains general methods like import/export support and references to the open documents and any open project.

The `Application` object is created differently in various programming languages. In scripting languages such as JScript or VBScript, this involves calling a function which initializes the application's COM object. For examples, see the [Programming Languages](#)⁸⁷⁸ section.

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.

17.2.1.2 Programming Languages

Programming languages differ in the way they support COM access. A few examples for the most frequently used languages (*links below*) will help you get started. The code listings in this section show how basic functionality can be accessed. The files in the `API` subfolder of the `Examples` folder can be used to test this functionality:

Windows 7, Windows 8, Windows 10, Windows 11	C:\Users\ <username>\Documents\ Altova\StyleVision\2024\%APPNAME%\Examples</username>
---	---

JScript

The JScript listings demonstrate the following basic functionality:

- [Start application or attach to a running instance](#) ⁸⁸⁰
- [Simple document access](#) ⁸⁸¹
- [Iteration](#) ⁸⁸²

VBScript

VBScript is different than JScript only syntactically; otherwise it works in the same way. For more information, refer to the [JScript examples](#) ⁸⁷⁹.

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](#) ⁸⁸⁷: Starts StyleVision, which is registered as an automation server, or activates StyleVision if it is already running.
- [Open OrgChart.pxf](#) ⁸⁸⁸: 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](#) ⁸⁸⁹: 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](#) ⁸⁸⁸: Opens another example document.
- [Shutdown StyleVision](#) ⁸⁸⁷: Stops StyleVision.

Java

The StyleVision API can be accessed from Java code. [The Java sub-section of this section](#) ⁸⁹¹ 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](#) ⁸⁹¹
- [Example Java Project](#) ⁸⁹²
- [Application Startup and Shutdown](#) ⁸⁹⁶
- [Simple Document Access](#) ⁸⁹⁷
- [Iterations](#) ⁸⁹⁷
- [Event Handlers](#) ⁸⁹⁷

17.2.1.2.1 JScript

This section contains listings of JScript code that demonstrate the following basic functionality:

- [Start application or attach to a running instance](#) ⁸⁸⁰

- [Simple document access](#) ⁸⁸¹
- [Iteration](#) ⁸⁸²

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` subfolder of the API Examples folder:

Windows 7, Windows 8, Windows 10, Windows 11	C:\Users\<<username>\Documents\ Altova\StyleVision\2024\%APPNAME%\Examples
---	---

The example files can be run in one of two ways:

- *From the command line:* Open a command prompt window, change the directory to the path above, and type the name of one of the example scripts (for example, `Start.js`).
- *From Windows Explorer:* In Windows Explorer, browse for the JScript file and double-click it.

The script is executed by Windows Script Host that is packaged with Windows operating system. For more information about Windows Script Host, refer to MSDN documentation (<https://msdn.microsoft.com>).

17.2.1.2.1.1 Start Application

The JScript below starts the application and shuts it down. If the COM object of the 32-bit StyleVision cannot be found, the code attempts to get the COM object of the 64-bit application; otherwise, an error is thrown. If an instance of the application is already running, the running instance will be returned.

Note: For 32-bit StyleVision, the registered name, or programmatic identifier (ProgId) of the COM object is `StyleVision.Application`. For 64-bit StyleVision, the name is `StyleVision_x64.Application`. Be aware, though, that the calling program will access the CLASSES registry entries in its own registry hive, or group (32-bit or 64-bit). Therefore, if you run scripts using the standard command prompt and Windows Explorer on 64-bit Windows, the 64-bit registry entries will be accessed, which point to the 64-bit StyleVision. For this reason, if both StyleVision 32-bit and 64-bit are installed, special handling is required in order to call the 32-bit StyleVision. For example, assuming that Windows Scripting Host is the calling program, do the following:

1. Change the current directory to **C:\Windows\SysWOW64**.
2. At the command line, type **wscript.exe** followed by the path to the script that you would like to run, for example:

```
wscript.exe "C:\Users\...
\Documents\Altova\StyleVision2024\StyleVisionExamples\API\JScript\start.js"
```

```
// Initialize application's COM object. This will start a new instance of the application
and
// return its main COM object. Depending on COM settings, the main COM object of an
```



```
already
// running application might be returned.
try { objStyleVision = WScript.GetObject("", "StyleVision.Application"); }
catch(err) {}

if( typeof( objStyleVision ) == "undefined" )
{
  try { objStyleVision = WScript.GetObject("", "StyleVision_x64.Application") }
  catch(err)
  {
    WScript.Echo( "Can't access or create StyleVision.Application" );
    WScript.Quit();
  }
}

// if newly started, the application will start without its UI visible. Set it to
visible.
objStyleVision.Visible = true;

WScript.Echo(objStyleVision.Edition + " has successfully started. ");

objStyleVision.Visible = false; // will shutdown application if it has no more COM
connections
//objStyleVision.Visible = true; // will keep application running with UI visible
```

The JScript code listed above is available in the sample file `Start.js` (see [Example Files](#)⁸⁸⁰).

17.2.1.2.1.2 Simple Document Access

After you have started the application as shown in [Start Application](#)⁸⁸⁰, you will most likely want to programmatically open a document in order to work with it. The JScript code listing below illustrates how to open two documents from the StyleVision Examples folder.

```
// Locate examples via USERPROFILE shell variable. The path needs to be adapted to major
release versions.
objWshShell = WScript.CreateObject("WScript.Shell");
majorVersionYear = objStyleVision.MajorVersion + 1998
strExampleFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\\My
Documents\\Altova\\StyleVision" + majorVersionYear + "\\StyleVisionExamples\\";

// Tell StyleVision to open two documents. No dialogs
objDoc1 = objStyleVision.Documents.OpenDocument(strExampleFolder + "OrgChart.pxf");
objStyleVision.Documents.OpenDocument(strExampleFolder +
"BiggestCitiesPerContinent.sps");
```

The JScript code listed above is available in the sample file `DocumentAccess.js` (see [Example Files](#)⁸⁸⁰).

17.2.1.2.1.3 Iteration

The JScript listing below shows how to iterate through the open documents. It is assumed that you have already started the application and opened some documents as shown in the previous sections.

```
// go through all open documents using a JScript Enumerator
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();
```

The JScript code listed above is available in the sample file `DocumentAccess.js` (see [Example Files](#) ⁸⁸⁰).

17.2.1.2.2 C#

The C# programming language can be used to access the Application API functionality. You could use Visual Studio 2012/2013/2015/2017/2019/2022 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 appears. Browse for the StyleVision Type Library component, which is located in the StyleVision application folder, and 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 `API\C#` subfolder of the `Examples` folder :

Windows 7, Windows 8, Windows 10, Windows 11	C:\Users\<<username>\Documents\ Altova\StyleVision\2024\%APPNAME%\Examples
---	---

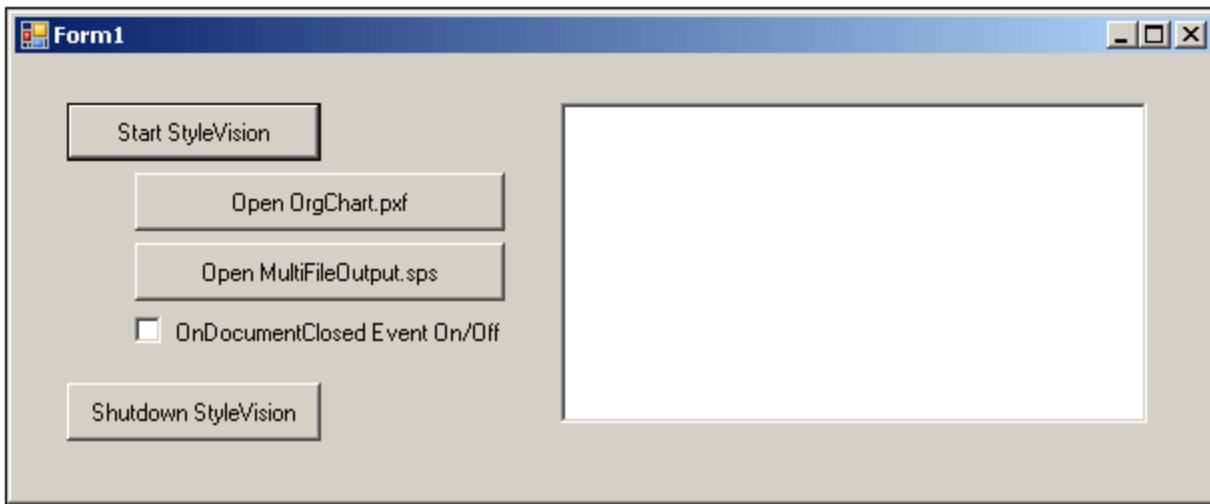
You can compile and run the project from within Visual Studio 2012/2013/2015/2017/2019/2022. The code listing below shows how basic application functionality can be used. This code is similar to the example C# project in the API Examples folder of your application package, but might differ slightly.

Platform configuration

If you have a 64-bit operating system and are using a 32-bit installation of StyleVision, you must add the x86 platform in the solution's Configuration Manager and build the sample using this configuration. A new x86 platform (for the active solution in Visual Studio) can be created in the New Solution Platform dialog (**Build | Configuration Manager | Active solution platform | <New...>**).

What the code listing below does

The example code listing below creates a simple user interface (*screenshot below*) with buttons that invoke basic StyleVision operations:



- [Start StyleVision](#)⁸⁸⁷: Starts StyleVision, which is registered as an automation server, or activates the application if it is already running.
- [Open OrgChart.pxf](#)⁸⁸⁸: 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](#)⁸⁸⁸: Opens another example document.
- [Shut down StyleVision](#)⁸⁸⁷: 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` or the file `AutomateStyleVision_VS2010.sln` (to open in Visual Studio 2012/2013/2015/2017/2019/2022). 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.

```
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;

                Cursor.Current = Cursors.Default;
            }
            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 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)
    {
        // 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.
        StyleVision.Documents.OpenDocument(strExamplesFolder + "MultiFileOutput.sps");
        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 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)
                    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)
                    {
                        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 + "\tType name : " + source.TypeName + "\n");
    }
    }
    }
    }
}

```

17.2.1.2.2.1 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:

```

// An instance of StyleVision is accessed via its automation interface.
StyleVisionLib.Application StyleVision;

```

17.2.1.2.2.2 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](#)⁸⁸² that, respectively, start up and shut down the application. This example is located in the C# subfolder of the API Examples folder (see *the file Form1.cs*):

Windows 7, Windows 8, Windows 10, Windows 11	C:\Users\<<username>\Documents\ Altova\StyleVision\2024\%APPNAME%\Examples
---	---

You can compile and run the project from within Visual Studio 2012/2013/2015/2017/2019/2022.

Starting StyleVision

The following code snippet from the [AutomateStyleVision example](#)⁸⁸² shows how to start up the application.

```

// 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;
    }
}

```

```

        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](#)⁸⁸² shows how to shut down the application.

```

// 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;
        }
    }
}

```

17.2.1.2.2.3 Opening Documents

The code snippets below (from the [AutomateStyleVision example](#)⁸⁸²) 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\(string\)](#)⁹⁷⁶.

The [AutomateStyleVision example](#)⁸⁸² (see the file *Form1.cs*) is located in the C# subfolder of the API Examples folder:

Windows 7, Windows 8, Windows 10, Windows 11	C:\Users\<<username>\Documents\ Altova\StyleVision\2024\%APPNAME%\Examples
---	---

You can compile and run the project from within Visual Studio 2012/2013/2015/2017/2019/2022.

Code snippet

```
// 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.
    StyleVision.Documents.OpenDocument(strExamplesFolder + "MultiFileOutput.sps");
    updateListBox();
}
```

The file opened last will be the active file.

17.2.1.2.2.4 Events

The code snippet below (from the [AutomateStyleVision example](#)⁸⁸²) lists the code for two event handlers. The [AutomateStyleVision example](#)⁸⁸² (see the file *Form1.cs*) is located in the C# subfolder of the API Examples folder:

Windows 7, Windows 8, Windows 10, Windows 11	C:\Users\<<username>\Documents\ Altova\StyleVision\2024\%APPNAME%\Examples
---	---

You can compile and run the project from within Visual Studio 2012/2013/2015/2017/2019/2022.

Code snippet

```
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)
                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)
            {
                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 + "\tType name : " + source.TypeName + "\n");
            }
        }
    }
}

```

17.2.1.2.3 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](#)⁸⁹².

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:

Interface	Java name
Document, method SetEncoding	setFileEncoding
AuthenticView, method Goto	gotoElement
AuthenticRange, method Goto	gotoElement
AuthenticRange, method Clone	cloneRange

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](#)⁸⁹²
- [Application Startup and Shutdown](#)⁸⁹⁶
- [Simple Document Access](#)⁸⁹⁷
- [Iterations](#)⁸⁹⁷
- [Event Handlers](#)⁸⁹⁷

17.2.1.2.3.1 Example Java Project

The StyleVision installation package contains an example Java project, located in the the `API\Java` subfolder of the `Examples` folder :

Windows 7, Windows 8, Windows 10, Windows 11	C:\Users\ <username>\Documents\ Altova\StyleVision\2024\%APPNAME%\Examples</username>
---	---

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)`.

<code>AltovaAutomation(_x64).dll</code>	Java-COM bridge: DLL part
<code>AltovaAutomation.jar</code>	Java-COM bridge: Java library part
<code>StyleVisionAPI.jar</code>	Java classes of the StyleVision API
<code>RunStyleVision.java</code>	Java example source code
<code>BuildAndRun.bat</code>	Batch file to compile and run example code from the command line prompt. Expects folder where Java Virtual Machine resides as parameter.
<code>.classpath</code>	Eclipse project helper file
<code>.project</code>	Eclipse project file
<code>StyleVision_JavaDoc.zip</code>	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](#)⁸⁹⁶: Starts StyleVision, which is registered as an automation server, or activates StyleVision if it is already running.
- [Open OrgChart.pxf](#)⁸⁹⁷: Locates one of the example documents installed with StyleVision and opens it.
- [Iteration and Changing the View Mode](#)⁸⁹⁷: Shows how to iterate through open documents.
- [Event Handling](#)⁸⁹⁷: Shows how to handle StyleVision events.
- [Shut down StyleVision](#)⁸⁹⁶: 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 14 or later installation on your computer. Press the **Return** key. The Java source in `RunStyleVision.java` will be compiled and then executed.

Loading the example in Eclipse

Open Eclipse and use the **Import | Existing Projects into Workspace** command to add the Eclipse project file (`.project`) located in the Java folder of the API Examples folder (see above for location). The project

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.

```
01 // Access general JAVA-COM bridge classes
02 import com.altova.automation.libs.*;
03
04 // Access StyleVision Java-COM bridge
05 import com.altova.automation.StyleVision.*;
06 import com.altova.automation.StyleVision.Enums.ENUMApplicationStatus;
07
08 /**
09  * A simple example that starts the COM server and performs a View operations on it.
10  * Feel free to extend.
11  */
12 public class RunStyleVision
13 {
14     public static void main(String[] args)
15     {
16         // An instance of the application.
17         Application stylevision = null;
18
19         // Instead of COM error-handling, use Java exception mechanism.
20         try
21         {
22             // Start StyleVision as COM server.
23             stylevision = new Application();
24
25             ENUMApplicationStatus status = ENUMApplicationStatus.eApplicationRunning;
26             do{
27                 // Check the application status
28                 status = stylevision.getStatus();
29                 System.out.println("status : " + status + "\n");
30             } while (status != ENUMApplicationStatus.eApplicationRunning);
31
32             // COM servers start up invisible, so we make the server visible
33             stylevision.setVisible(true);
34
35             // Locate samples installed with the product.
36             String strExamplesFolder = System.getenv("USERPROFILE") + "\\My Documents\\Altova\\
37             \\StyleVision2012\\StyleVisionExamples\\";
38
39             // Open two files from the product samples.
40             stylevision.getDocuments().openDocument(strExamplesFolder + "OrgChart.pxf");
41             stylevision.getDocuments().openDocument(strExamplesFolder +
42             "BiggestCitiesPerContinent.sps");
43
44             // Iterate through all open documents
45             for (Document doc:stylevision.getDocuments())
```

```
44     {
45         System.out.println("Document name : " + doc.getName() + "\n");
46         SchemaSources sources = doc.getSchemaSources();
47
48         for (int i = 1; i <= sources.getCount(); i++)
49             {
50                 SchemaSource source = sources.getItem(i);
51                 System.out.println("\tSchema      file name : " + source.getSchemaFileName() +
52 "\n");
53                 System.out.println("\tWorking XML file name : " +
54 source.getWorkingXMLFileName() + "\n");
55                 System.out.println("\tIs main schema source : " +
56 source.getIsMainSchemaSource() + "\tType name : " + source.getTypeName() + "\n");
57             }
58         // The following lines attach to the document events using a default
59 implementation
60 // for the events and override one of its methods.
61 // If you want to override all document events it is better to derive your
62 listener class
63 // from DocumentEvents and implement all methods of this interface.
64 Document doc = stylevision.getActiveDocument();
65 doc.addListener(new DocumentEventsDefaultHandler()
66 {
67     @Override
68     public void onDocumentClosed(Document i_ipDoc) throws AutomationException
69     {
70         System.out.println("Document " + i_ipDoc.getName() + " requested closing.");
71     }
72 });
73 doc.close();
74 doc = null;
75
76 System.out.println("Watch StyleVision!");
77 }
78 catch (AutomationException e)
79 {
80     e.printStackTrace();
81 }
82 finally
83 {
84     // Make sure that StyleVision can shut down properly.
85     if (stylevision != null)
86         stylevision.dispose();
87 }
88 }
89 }
```

17.2.1.2.3.2 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*).

```
01 // Access general JAVA-COM bridge classes
02 import com.altova.automation.libs.*;
03
04 // Access StyleVision Java-COM bridge
05 import com.altova.automation.StyleVision.*;
06 import com.altova.automation.StyleVision.Enums.ENUMApplicationStatus;
07
08 /**
09  * A simple example that starts the COM server and performs a View operations on it.
10  * Feel free to extend.
11  */
12 public class RunStyleVision
13 {
14     public static void main(String[] args)
15     {
16         // An instance of the application.
17         Application stylevision = null;
18
19         // Instead of COM error-handling, use Java exception mechanism.
20         try
21         {
22             // Start StyleVision as COM server.
23             stylevision = new Application();
24
25             ENUMApplicationStatus status = ENUMApplicationStatus.eApplicationRunning;
26             do{
27                 // Check the application status
28                 status = stylevision.getStatus();
29                 System.out.println("status : " + status + "\n");
30             } while (status != ENUMApplicationStatus.eApplicationRunning);
31
32             // COM servers start up invisible, so we make the server visible
33             stylevision.setVisible(true);
34
35             ...
36         }
37     }
38 }
```

Application shutdown

The application can be shut down as shown below.

```
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 keep running
7     // and needs to be closed manually.
8     System.out.println("Now close StyleVision!");
9 }
```

17.2.1.2.3.3 Simple Document Access

The code listing below shows how to open a document.

```
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");
```

17.2.1.2.3.4 Iterations

The listing below shows how to iterate through open documents.

```
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 }
```

17.2.1.2.3.5 Event Handlers

The listing below shows how to listen for and use events.

```
01 // The following lines attach to the document events using a default implementation
02 // for the events and override one of its methods.
03 // If you want to override all document events it is better to derive your listener
class
04 // from DocumentEvents and implement all methods of this interface.
05 Document doc = stylevision.getActiveDocument();
06 doc.addListener(new DocumentEventsDefaultHandler()
07 {
08     @Override
09     public void onDocumentClosed(Document i_ipDoc) throws AutomationException
10     {
11         System.out.println("Document " + i_ipDoc.getName() + " requested closing.");
12     }
13 });
14 doc.close();
15 doc = null;
```

17.2.2 Interfaces

This chapter contains the reference of the StyleVision Type Library.

17.2.2.1 Application

Methods

[Quit](#) ⁹⁰²

Properties

[Application](#) ⁸⁹⁹

[Parent](#) ⁹⁰¹

[ActiveDocument](#) ⁸⁹⁹

[Documents](#) ⁸⁹⁹

[Status](#) ⁹⁰²

[MajorVersion](#) ⁹⁰⁰

[MinorVersion](#) ⁹⁰⁰

[Edition](#) ⁹⁰⁰

[IsAPISupported](#) ⁹⁰⁰

[ServicePackVersion](#) ⁹⁰²

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")
```

17.2.2.1.1 Events

17.2.2.1.1.1 OnShutDown

Event: OnShutDown()

Description

Sent just before StyleVision shuts down.

17.2.2.1.2 ActiveDocument

Property: ActiveDocument as [Document](#) ⁹⁶¹

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.

17.2.2.1.3 Application

Property: Application as [Application](#) ⁸⁹⁸ (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.

17.2.2.1.4 Documents

Property: Documents as [Documents](#) ⁹⁷⁴

Description

Collection of all open documents.

Errors

1111	The application object is no longer valid.
1100	Invalid address for the return parameter was specified.

17.2.2.1.5 Edition

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.

17.2.2.1.6 IsAPISupported

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.

17.2.2.1.7 MajorVersion

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.

17.2.2.1.8 MinorVersion

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.

17.2.2.1.9 NewDocument

Method: NewDocument as [Document](#) ⁹⁶¹

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

17.2.2.1.10 OpenDocument

Method: OpenDocument(strFileName as *String*) as [Document](#) ⁹⁶¹

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.

17.2.2.1.11 Parent

Property: Parent as [Application](#) ⁹⁶⁸ (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.

17.2.2.1.12 Quit

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.
------	--

17.2.2.1.13 ServicePackVersion

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.

17.2.2.1.14 Status

Property: Status as [ENUMApplicationStatus](#) ⁹⁹⁶

Description

Returns the current status of the running application.

Errors

1111	The application object is no longer valid.
1100	Invalid address for the return parameter was specified.

17.2.2.1.15 Visible

Property: Visible as VARIANT_BOOL

Description

Sets or gets the visibility attribute of StyleVision.

Errors

1110	The application object is no longer valid.
1100	Invalid address for the return parameter was specified.

17.2.2.2 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](#)⁹⁰⁴
[Parent](#)⁹⁰⁸

Line access:

[GetLineSeverity](#)⁹⁰⁶
[GetLineSymbol](#)⁹⁰⁷
[GetLineText](#)⁹⁰⁷
[GetLineTextEx](#)⁹⁰⁷
[GetLineTextWithChildren](#)⁹⁰⁷
[GetLineTextWithChildrenEx](#)⁹⁰⁸

A single AppOutputLine consists of one or more sub-lines.

Sub-line access:

[GetLineCount](#)⁹⁰⁶

A sub-line consists of one or more cells.

Cell access:

[GetCellCountInLine](#)⁹⁰⁴
[GetCellIcon](#)⁹⁰⁵
[GetCellSymbol](#)⁹⁰⁷
[GetCellText](#)⁹⁰⁵
[GetCellTextDecoration](#)⁹⁰⁵
[GetIsCellText](#)⁹⁰⁵

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](#)⁹⁰⁴

17.2.2.2.1 Application

Property: Application as [Application](#)⁸⁹⁸ (read-only)

Description

Retrieves the application's top-level object.

Errors

4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.2.2.2.2 ChildLines

Property: ChildLines as [AppOutputLines](#)⁹⁰⁸ (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.

17.2.2.2.3 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.

17.2.2.2.4 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.

17.2.2.2.5 GetCellSymbol

Method: GetCellSymbol (*nLine* as Long, *nCell* as Long) as [AppOutputLineSymbol](#)⁹¹⁰

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.

17.2.2.2.6 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.

17.2.2.2.7 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 [ENUMAppOutputLine_TextDecoration](#)⁹⁹⁷ values.

Errors

4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.2.2.2.8 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.

17.2.2.2.9 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.

17.2.2.2.10 GetLineSeverity

Method: GetLineSeverity () as Long

Description

Gets the severity of the line. It can be one of the [ENUMAppOutputLine_Severity](#)⁹⁹⁶ values:

Errors

4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.2.2.2.11 GetLineSymbol

Method: GetLineSymbol () as [AppOutputLineSymbol](#)⁹¹⁰

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.

17.2.2.2.12 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.

17.2.2.2.13 GetLineTextEx

Method: GetLineTextEx (*psTextPartSeperator* as String, *psLineSeperator* 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.

17.2.2.2.14 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.

17.2.2.2.15 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.

17.2.2.2.16 Parent

Property: Parent as [AppOutputLines](#)⁹⁰⁸ (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.

17.2.2.3 AppOutputLines

Represents a collection of `AppOutputLine` message lines.

Properties and Methods

Properties to navigate the object model:

[Application](#)⁹⁰⁹
[Parent](#)⁹⁰⁹

Iterating through the collection:

[Count](#)⁹⁰⁹
[Item](#)⁹⁰⁹

17.2.2.3.1 Application

Property: Application as [Application](#)⁸⁹⁸ (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.

17.2.2.3.2 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.

17.2.2.3.3 Item

Property: Item (*nIndex* as Integer) as [AppOutputLine](#)⁹⁰³ (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.

17.2.2.3.4 Parent

Property: Parent as [AppOutputLine](#)⁹⁰³ (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.

17.2.2.4 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](#)⁹¹⁰
[Parent](#)⁹¹¹

Access to `AppOutputLineSymbol` methods:

[GetSymbolHREF](#)⁹¹⁰
[GetSymbolID](#)⁹¹¹
[IsSymbolHREF](#)⁹¹¹

17.2.2.4.1 Application

Property: Application as [Application](#)⁸⁹⁸ (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.

17.2.2.4.2 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.

.

17.2.2.4.3 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.

17.2.2.4.4 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.

17.2.2.4.5 Parent

Property: Parent as [Application](#) ⁸⁹⁸ (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.

17.2.2.5 AuthenticContextMenu

The context menu interface provides the means for the user to customize the context menus shown in Authentic. The interface has the methods listed in this section.

17.2.2.5.1 CountItems

Method: CountItems () nItems as long

Return Value

Returns the number of menu items.

Errors

2501	Invalid object.
------	-----------------

17.2.2.5.2 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

17.2.2.5.3 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

17.2.2.5.4 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

17.2.2.5.5 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

17.2.2.6 AuthenticEventContext

The `EventContext` interface gives access to many properties of the context in which a macro is executed.

17.2.2.6.1 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 expression with the given event context. The result is returned as a 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.

17.2.2.6.2 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.

17.2.2.6.3 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.

17.2.2.6.4 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

17.2.2.6.5 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.

17.2.2.6.6 IsAvailable

Method: IsAvailable () as Boolean

Return Value

Returns true if `EventContext` is set, false otherwise.

Errors

2201	Invalid object.
------	-----------------

17.2.2.6.7 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.

```
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

17.2.2.7 AuthenticRange

The first table lists the properties and methods of `AuthenticRange` that can be used to navigate through the document and select specific portions.

Properties	Methods	
Application ⁹¹⁹	Clone ⁹²⁰	MoveBegin ⁹³⁹
FirstTextPosition ⁹²⁴	CollapsToBegin ⁹²⁰	MoveEnd ⁹³⁹
FirstXMLData ⁹²⁵	CollapsToEnd ⁹²¹	NextCursorPosition ⁹³⁰
FirstXMLDataOffset ⁹²⁶	ExpandTo ⁹²⁴	PreviousCursorPosition ⁹³¹
LastTextPosition ⁹³⁶	Goto ⁹²⁹	Select ⁹⁴¹
LastXMLData ⁹³⁷	GotoNext ⁹²⁹	SelectNext ⁹⁴²
LastXMLDataOffset ⁹³⁷	GotoPrevious ⁹³⁰	SelectPrevious ⁹⁴³
Parent ⁹⁴⁰	IsEmpty ⁹³⁴	SetFromRange ⁹⁴⁵
	IsEqual ⁹³⁴	

The following table lists the content modification methods, most of which can be found on the right/button mouse menu.

Properties	Edit operations	Dynamic table operations
Text ⁹⁴⁵	Copy ⁹²¹	AppendRow ⁹¹⁸
	Cut ⁹²¹	DeleteRow ⁹²²
	Delete ⁹²²	DuplicateRow ⁹²²
	IsCopyEnabled ⁹³³	InsertRow ⁹³²
	IsCutEnabled ⁹³³	IsFirstRow ⁹³⁴
	IsDeleteEnabled ⁹³³	IsInDynamicTable ⁹³⁵
	IsPasteEnabled ⁹³⁵	IsLastRow ⁹³⁵
	Paste ⁹⁴⁰	MoveRowDown ⁹³⁹
		MoveRowUp ⁹⁴⁰

The following methods provide the functionality of the Authentic entry helper windows for range objects.

Operations of the entry helper windows		
Elements	Attributes	Entities
CanPerformActionWith ⁹²⁰	GetElementAttributeValue ⁹²⁷	GetEntityNames ⁹²⁸
CanPerformAction ⁹¹⁹	GetElementAttributeNames ⁹²⁷	InsertEntity ⁹³²
PerformAction ⁹⁴¹	GetElementHierarchy ⁹²⁸	
	HasElementAttribute ⁹³¹	
	IsTextStateApplied ⁹³⁶	
	SetElementAttributeValue ⁹⁴⁴	

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](#)⁹⁵⁹, 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](#)⁹⁵⁴, [AuthenticView.DocumentEnd](#)⁹⁵⁴, and [AuthenticView.WholeDocument](#)⁹⁶¹. The most flexible method is [AuthenticView.Goto](#)⁹⁵⁶, 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](#)⁹²⁴ and [LastTextPosition](#)⁹³⁶. 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](#)⁹²⁵, [FirstXMLDataOffset](#)⁹²⁶, [LastXMLData](#)⁹³⁷, and [LastXMLDataOffset](#)⁹³⁷. This method is very efficient but requires knowledge of 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](#)⁹⁴⁵ 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](#)⁹²¹, [Copy](#)⁹²¹, [Paste](#)⁹⁴⁰ and [Delete](#)⁹²².
- Table operations for tables that can grow dynamically.
- Methods that map the functionality of the Authentic entry helper windows.
- Access to the [XMLData](#)⁹⁸⁴ objects of the underlying document to modify them directly.

17.2.2.7.1 AppendRow

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
    
```

17.2.2.7.2 Application

Property: Application as [Application](#)⁸⁹⁸ (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.

17.2.2.7.3 CanPerformAction

Method: CanPerformAction (*eAction* as [SPYAuthenticActions](#)⁸⁹⁸, *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](#)⁹²⁰.

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](#)⁹⁴¹.

17.2.2.7.4 CanPerformActionWith

Method: CanPerformActionWith (eAction as [SPYAuthenticActions](#)⁹⁹⁸, out_arrElementNames as Variant)

Description

PerformActionWith 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.

This method returns an array of those element names that the specified action can be performed with.

HINT: To apply the action use [CanPerformActionWith](#)⁹²⁰.

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](#)⁹⁴¹.

17.2.2.7.5 Clone

Method: Clone() as [AuthenticRange](#)⁹¹⁶

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.

17.2.2.7.6 CollapsToBegin

Method: CollapsToBegin() as [AuthenticRange](#)⁹¹⁶

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.
------	---

17.2.2.7.7 CollapsToEnd

Method: CollapsToEnd() as [AuthenticRange](#)⁹¹⁶

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.

17.2.2.7.8 Copy

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.

17.2.2.7.9 Cut

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.

17.2.2.7.10 Delete

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.

17.2.2.7.11 DeleteRow

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
```

17.2.2.7.12 DuplicateRow

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
```

17.2.2.7.13 EvaluateXPath

Method: EvaluateXPath (strExpression as string) strValue as string

Return Value

The method returns a string

Description

EvaluateXPath() executes an XPath expression with the context node being the beginning of the range selection. The result is returned as a 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

17.2.2.7.14 ExpandTo

Method: ExpandTo (eKind as [SPYAuthenticElementKind](#)⁹⁹⁸), as [AuthenticRange](#)⁹¹⁶

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.

17.2.2.7.15 FirstTextPosition

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](#)⁹³⁶. Indexing starts with 0 at document beginning, and increments with every different position that the text cursor can occupy. Incrementing the test position by 1, has the same effect as the cursor-right key. Decrementing the test position by 1 has the same effect as the cursor-left key.

If you set FirstTextPosition to a value greater than the current [LastTextPosition](#)⁹³⁶, [LastTextPosition](#)⁹³⁶ gets set to the new FirstTextPosition.

HINT: Use text cursor positions with care, since this is a costly operation compared to XMLData based cursor positioning.

Errors

2001	The authentic range object, or its related view object is not valid.
2005	Invalid address for the return parameter was specified.
2006	A text position outside the document was specified.

Examples

```
' -----
' Scripting environment - VBScript
' -----
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

nDocStartPosition = objAuthenticView.DocumentBegin.FirstTextPosition
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
```

17.2.2.7.16 FirstXMLData

Property: FirstXMLData as [XMLData](#)⁹⁸⁴

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](#)⁹²⁶ attribute.

Whenever you set FirstXMLData to a new data object, [FirstXMLDataOffset](#)⁹²⁶ 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](#)⁹²⁶ selects a position greater than the current [LastXMLData](#)⁹³⁷ / [LastXMLDataOffset](#)⁹³⁷, the latter gets moved to the new start position.

HINT: You can use the [FirstXMLData](#)⁹²⁵ and [LastXMLData](#)⁹³⁷ properties to directly access and manipulate the underlying XML document in those cases where the methods available with the [AuthenticRange](#)⁹¹⁶ 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
```

17.2.2.7.17 FirstXMLDataOffset

Property: FirstXMLDataOffset as Long

Description

Set or get the cursor position offset inside [FirstXMLData](#)⁹²⁵ element for the beginning of the range. Offset positions are based on the characters returned by the [Text](#)⁹⁴⁵ 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](#)⁹²⁶ selects a position after the current [LastXMLData](#)⁹³⁷ / [LastXMLDataOffset](#)⁹³⁷, 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
```

17.2.2.7.18 GetElementAttributeNames

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](#)⁹²⁷ and [SetElementAttributeValue](#)⁹⁴⁴.

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](#)⁹⁴⁴.

17.2.2.7.19 GetElementAttributeValue

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](#)⁹²⁷, or [HasElementAttribute](#)⁹³¹.

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](#)⁹⁴⁴.

17.2.2.7.20 GetElementHierarchy

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](#)⁹²⁷, 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.

17.2.2.7.21 GetEntityNames

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](#)⁹³² 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](#)⁹²⁸ and [InsertEntity](#)⁹³².

17.2.2.7.22 GetVariableValue

Method: GetVariableValue (*strName* as string) *strVal* as string

Return Value

Gets the value of the variable named as the method's parameter.

Errors

2001	Invalid object.
2202	No context.

2204	No such variable in scope
2205	Variable cannot be evaluated
2206	Variable returns sequence
2209	Invalid parameter

17.2.2.7.23 Goto

Method: Goto (eKind as [SPYAuthenticElementKind](#)⁹⁹⁸, nCount as Long, eFrom as [SPYAuthenticDocumentPosition](#)⁹⁹⁸) as [AuthenticRange](#)⁹¹⁶

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. Invalid start position specified. Invalid address for the return parameter was specified.

17.2.2.7.24 GotoNext

Method: GotoNext (eKind as [SPYAuthenticElementKind](#)⁹⁹⁸) as [AuthenticRange](#)⁹¹⁶

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. Invalid address for the return parameter was specified.

Examples

```
' -----
' Scripting environment - VBScript
' Scan through the whole document word-by-word
```

```

' -----
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

Dim objRange
Set objRange = objAuthenticView.DocumentBegin
Dim bEndOfDocument
bEndOfDocument = False

On Error Resume Next
While Not bEndOfDocument
    objRange.GotoNext(spyAuthenticWord).Select
    If ((Err.number - vbObjecterror) = 2003) Then
        bEndOfDocument = True
        Err.Clear
    ElseIf (Err.number <> 0) Then
        Err.Raise ' forward error
    End If
Wend

```

17.2.2.7.25 GotoNextCursorPosition

Method: GotoNextCursorPosition() as [AuthenticRange](#)⁹¹⁶

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.

17.2.2.7.26 GotoPrevious

Method: GotoPrevious (eKind as [SPYAuthenticElementKind](#)⁹⁹⁸) as [AuthenticRange](#)⁹¹⁶

Description

Sets the range to the beginning of the element of type eKind which is before the beginning of the current range. The method returns the modified range object.

Errors

2001	The authentic range object, or its related view object is no longer valid.
2004	Target lies before beginning of document.
2005	Invalid element kind specified. Invalid address for the return parameter was specified.

Examples

```
' -----
' Scripting environment - VBScript
' Scan through the whole document tag-by-tag
' -----
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

Dim objRange
Set objRange = objAuthenticView.DocumentEnd
Dim bBeginOfDocument
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
```

17.2.2.7.27 GotoPreviousCursorPosition

Method: GotoPreviousCursorPosition() as [AuthenticRange](#) ⁹¹⁶

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.

17.2.2.7.28 HasElementAttribute

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.
------	--

17.2.2.7.29 InsertEntity

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](#)⁹²⁸.

Errors

2001	The authentic range object, or its related view object is no longer valid.
2005	Unknown entry name was specified.

Examples

```
' -----
' Scripting environment - VBScript
' Insert the first entity in the list of available entities
' -----
Dim objRange
' we assume that the active document is open in authentic view mode
Set objRange = Application.ActiveDocument.AuthenticView.Selection

' first we get the names of all available entities as they
' are shown in the entry helper of XMLSpy
Dim arrEntities
objRange.GetEntityNames arrEntities

' we insert the first one of the list
If UBound(arrEntities) >= 0 Then
    objRange.InsertEntity arrEntities(0)
Else
    MsgBox "Sorry, no entities are available for this document"
End If
```

17.2.2.7.30 InsertRow

Method: InsertRow() as Boolean

Description

If the beginning of the range is inside a dynamic table, this method inserts a new row before the current one. The selection of the range gets modified to point to the beginning of the newly inserted row. The function returns *true* if the insert operation was successful, otherwise *false*.

Errors

2001	The authentic range object, or its related view object is no longer valid.
------	--

2005	Invalid address for the return parameter was specified.
------	---

Examples

```

'-----
' Scripting environment - VBScript
' Insert row at beginning of current dynamically growing table
'-----
Dim objRange
' we assume that the active document is open in authentic view mode
Set objRange = Application.ActiveDocument.AuthenticView.Selection

' check if we can insert something
If objRange.IsInDynamicTable Then
    objRange.InsertRow
    ' objRange points to beginning of new row
    objRange.Select
End If
    
```

17.2.2.7.31 [IsCopyEnabled](#)

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.

17.2.2.7.32 [IsCutEnabled](#)

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.

17.2.2.7.33 [IsDeleteEnabled](#)

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.

17.2.2.7.34 IsEmpty

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.

17.2.2.7.35 IsEqual

Method: IsEqual (*objCmpRange* as [AuthenticRange](#)⁹¹⁶) 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.

17.2.2.7.36 IsFirstRow

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 extent 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.

17.2.2.7.37 IsInDynamicTable

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.

17.2.2.7.38 IsLastRow

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 extent of the range. If the selection exceeds a single row of a table, the check is if this table is the last element in an embedding table. See the entry helpers of the user manual for more information.

Errors

2001	The authentic range object, or its related view object is no longer valid.
2005	Invalid address for the return parameter was specified.

17.2.2.7.39 IsPasteEnabled

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.

17.2.2.7.40 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.

17.2.2.7.41 IsTextStateApplied

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.

17.2.2.7.42 LastTextPosition

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](#)⁹²⁴. Indexing starts with 0 at the document beginning, and increments with every different position that the text cursor can occupy. Incrementing the test position by 1, has the same effect as the cursor-right key. Decreasing the test position by 1 has the same effect as the cursor-left key.

If you set LastTextPosition to a value less then the current [FirstTextPosition](#)⁹²⁴, [FirstTextPosition](#)⁹²⁴ 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

nDocStartPosition = objAuthenticView.DocumentBegin.FirstTextPosition
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
```

17.2.2.7.43 LastXMLData

Property: LastXMLData as [XMLData](#)⁹⁸⁴

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](#)⁹³⁷ attribute.

Whenever you set LastXMLData to a new data object, [LastXMLDataOffset](#)⁹³⁷ 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](#)⁹³⁷, select a position less than the current [FirstXMLData](#)⁹²⁵ / [FirstXMLDataOffset](#)⁹²⁶, the latter gets moved to the new end position.

HINT: You can use the [FirstXMLData](#)⁹²⁵ and [LastXMLData](#)⁹³⁷ properties to directly access and manipulate the underlying XML document in those cases, where the methods available with the [AuthenticRange](#)⁹¹⁶ 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.

17.2.2.7.44 LastXMLDataOffset

Property: LastXMLDataOffset as Long

Description

Set or get the cursor position inside [LastXMLData](#)⁹³⁷ element for the end of the range.

Offset positions are based on the characters returned by the [Text](#)⁹⁴⁵ 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](#)⁹³⁷ / [LastXMLDataOffset](#)⁹³⁷ selects a position before [FirstXMLData](#)⁹²⁵ / [FirstXMLDataOffset](#)⁹²⁶, 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. 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 "Ooops"
End If
```

17.2.2.7.45 MoveBegin

Method: MoveBegin (eKind as [SPYAuthenticElementKind](#)⁹⁹⁸, nCount as Long) as [AuthenticRange](#)⁹¹⁶

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.

17.2.2.7.46 MoveEnd

Method: MoveEnd (eKind as [SPYAuthenticElementKind](#)⁹⁹⁸, nCount as Long) as [AuthenticRange](#)⁹¹⁶

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.

17.2.2.7.47 MoveRowDown

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.

17.2.2.7.48 MoveRowUp

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.

17.2.2.7.49 Parent

Property: Parent as [AuthenticView](#)⁹⁴⁶ (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.

17.2.2.7.50 Paste

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.

17.2.2.7.51 PerformAction

Method: PerformAction (eAction as [SPYAuthenticActions](#)⁹⁹⁸, 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](#)⁹²⁰.

Errors

2001	The authentic range object, or its related view object is no longer valid.
2005	Invalid address for the return parameter was specified.
2007	Invalid action was specified.

Examples

```
' -----
' Scripting environment - VBScript
' Insert the innermost element
' -----

Dim objRange
' we assume that the active document is open in authentic view mode
Set objRange = Application.ActiveDocument.AuthenticView.Selection

' we determine the elements that can be inserted at the current position
Dim arrElements()
objRange.CanPerformActionWith spyAuthenticInsertBefore, arrElements

' we insert the first (innermost) element
If UBound(arrElements) >= 0 Then
    objRange.PerformAction spyAuthenticInsertBefore, arrElements(0)
    ' objRange now points to the beginning of the inserted element
    ' we set a default value and position at its end
    objRange.Text = "Hello"
    objRange.ExpandTo(spyAuthenticTag).CollapsToEnd().Select
Else
    MsgBox "Can't insert any elements at current position"
End If
```

17.2.2.7.52 Select

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()
```

17.2.2.7.53 SelectNext

Method: SelectNext (eKind as [SPYAuthenticElementKind](#)⁹⁹⁸) as [AuthenticRange](#)⁹¹⁶

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

```

17.2.2.7.54 SelectPrevious

Method: GotoPrevious (eKind as [SPYAuthenticElementKind](#)⁹⁹⁸) as [AuthenticRange](#)⁹¹⁶

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

```

17.2.2.7.55 SetElementAttributeValue

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](#)⁹²⁷, or [HasElementAttribute](#)⁹³¹.

Errors

2001	The authentic range object or its related view object is no longer valid.
2005	Invalid element name was specified. Invalid attribute name was specified. Invalid attribute value was specified.

Examples

```
' -----
' Scripting environment - VBScript
' Get and set element attributes
' -----

Dim objRange
' we assume that the active document is open in authentic view mode
Set objRange = Application.ActiveDocument.AuthenticView.Selection

' first we find out all the elements below the beginning of the range
Dim arrElements
objRange.GetElementHierarchy arrElements

If IsArray(arrElements) Then
    If UBound(arrElements) >= 0 Then
        ' we use the top level element and find out its valid attributes
        Dim arrAttrs()
        objRange.GetElementAttributeNames arrElements(0), arrAttrs

        If UBound(arrAttrs) >= 0 Then
            ' we retrieve the current value of the first valid attribute
            Dim strAttrVal
            strAttrVal = objRange.GetElementAttributeValue (arrElements(0), arrAttrs(0))
            msgbox "current value of " & arrElements(0) & "/" & arrAttrs(0) & " is: " & strAttrVal

            ' we change this value and read it again
            strAttrVal = "Hello"
            objRange.SetElementAttributeValue arrElements(0), arrAttrs(0), strAttrVal
            strAttrVal = objRange.GetElementAttributeValue (arrElements(0), arrAttrs(0))
            msgbox "new value of " & arrElements(0) & "/" & arrAttrs(0) & " is: " & strAttrVal
        End If
    End If
End If
```


17.2.2.7.56 SetFromRange

Method: SetFromRange (*objSrcRange* as [AuthenticRange](#)⁹¹⁶)

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.

17.2.2.7.57 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

17.2.2.7.58 Text

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 than one element, the text is the concatenation of the single texts. XML entities are expanded so that '&' is expected as '&'.

Setting the text to the empty string, does not delete any XML elements. Use [Cut](#)⁹²¹, [Delete](#)⁹²² or [PerformAction](#)⁹⁴¹ 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.

17.2.2.8 AuthenticView

Properties	Methods	Events
Application ⁹⁵²	Goto ⁹⁵⁶	OnBeforeCopy ⁹⁴⁷
AsXMLString ⁹⁵²	IsRedoEnabled ⁹⁵⁷	OnBeforeCut ⁹⁴⁷
DocumentBegin ⁹⁵⁴	IsUndoEnabled ⁹⁵⁷	OnBeforeDelete ⁹⁴⁷
DocumentEnd ⁹⁵⁴	Print ⁹⁵⁸	OnBeforeDrop ⁹⁴⁸
Event ⁹⁵⁶	Redo ⁹⁵⁹	OnBeforePaste ⁹⁴⁸
MarkupVisibility ⁹⁵⁸	Undo ⁹⁶⁰	
Parent ⁹⁵⁸	UpdateXMLInstanceEntities ⁹⁶⁰	
Selection ⁹⁵⁹		OnMouseEvent ⁹⁴⁹
XMLDataRoot ⁹⁶¹		OnSelectionChanged ⁹⁵⁰
WholeDocument ⁹⁶¹		

Description

AuthenticView and its child objects [AuthenticRange](#)⁹¹⁶ 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](#)⁹¹⁶)

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](#)⁹¹⁶ to make navigation inside the document straightforward, and to allow for the flexible selection of logical text elements. Use the properties [DocumentBegin](#)⁹⁵⁴, [DocumentEnd](#)⁹⁵⁴, or [WholeDocument](#)⁹⁶¹ for simple selections, while using the [Goto](#)⁹⁵⁶ method for more complex selections. To navigate relative to a given document range, see the methods and properties of the [AuthenticRange](#)⁹¹⁶ object.

17.2.2.8.1 Events

17.2.2.8.1.1 *OnBeforeCopy*

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 */  
}
```

17.2.2.8.1.2 *OnBeforeCut*

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 */  
}
```

17.2.2.8.1.3 *OnBeforeDelete*

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 */  
}
```

17.2.2.8.1.4 *OnBeforeDrop*

Event: OnBeforeDrop (*i_nXPos* as Long, *i_nYPos* as Long, *i_ipRange* as [AuthenticRange](#)⁹¹⁶, *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.

17.2.2.8.1.5 *OnBeforePaste*

Event: OnBeforePaste (*objData* as Variant, *strType* as String) as Boolean

Scripting environment - VBScript:

```
Function On_AuthenticBeforePaste(objData, strType)
    ' On_AuthenticBeforePaste = False ' to disable operation
End Function
```

Scripting environment - JScript:

```
function On_AuthenticBeforePaste(objData, strType)
{
    // return false; /* to disable operation */
}
```

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.

17.2.2.8.1.6 *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.)

17.2.2.8.1.7 *OnLoad*

Event: OnLoad ()

Description: OnLoad can be used e.g. to restrict some AuthenticView functionality, as shown in the example below:

```
function On_AuthenticLoad( )
{
    // We are disabling all entry helpers in order to prevent user from manipulating XML tree
    AuthenticView.DisableElementEntryHelper();
    AuthenticView.DisableAttributeEntryHelper();

    // We are also disabling the markup buttons for the same purpose
    AuthenticView.SetToolBarButtonState( 'AuthenticMarkupSmall', authenticToolBarButtonDisabled );
    AuthenticView.SetToolBarButtonState( 'AuthenticMarkupLarge', authenticToolBarButtonDisabled );
    AuthenticView.SetToolBarButtonState( 'AuthenticMarkupMixed', authenticToolBarButtonDisabled );
}
```

In the example the status of the Markup Small, Markup Large, Markup Mixed toolbar buttons are manipulated with the help of button identifiers. See [complete list](#)⁹⁵⁰.

17.2.2.8.1.8 *OnMouseEvent*

Event: OnMouseEvent (*nXPos* as Long, *nYPos* as Long, *eMouseEvent* as [SPYMouseEvent](#)⁹⁹⁹, *objRange* as [AuthenticRange](#)⁹¹⁶) as Boolean

Scripting environment - VBScript:

```
Function On_AuthenticMouseEvent(nXPos, nYPos, eMouseEvent, objRange)
    ' On_AuthenticMouseEvent = True ' to cancel bubbling of event
End Function
```

Scripting environment - JScript:

```
function On_AuthenticMouseEvent(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](#)⁹⁹⁹ 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 object 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*.

17.2.2.8.1.9 OnSelectionChanged

Event: OnSelectionChanged (*objNewSelection* as [AuthenticRange](#)⁹¹⁶)

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.

17.2.2.8.1.10 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
- AuthenticTableDeleteTable
- AuthenticTableProperties
- AuthenticAppendRow
- AuthenticInsertRow
- AuthenticDuplicateRow
- AuthenticMoveRowUp
- AuthenticMoveRowDown
- AuthenticDeleteRow
- AuthenticDefineEntities
- AuthenticXMLSignature

For custom buttons the user might add his own identifiers. Please, note that the user must take care, as the identifiers are not checked for uniqueness. The same identifiers can be used to identify buttons in the `Set/GetToolBarState()` COM API calls. By adding code for different buttons, the user is in the position to completely redefine the `AuthenticView` toolbar behavior, adding own methods for table manipulation, etc.

17.2.2.8.1.11 *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](#)⁹⁵⁰.

`OnToolBarButtonExecuted` is fired after the toolbar action was executed. It is useful e.g. to add update code, as shown in the example below:

```
//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.

17.2.2.8.1.12 *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.

17.2.2.8.2 Application

Property: Application as [Application](#)⁸⁹⁸ (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.

17.2.2.8.3 AsXMLString

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.
------	---

17.2.2.8.4 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.

17.2.2.8.5 CreateXMLNode

Method: CreateXMLNode (*nKind* as [SPYXMLDataKind](#)¹⁰⁰¹) as [XMLData](#)⁹⁸⁴

Return Value

The method returns the new [XMLData](#)⁹⁸⁴ object.

Description

To create a new XMLData object use the CreateXMLNode() method.

Errors

2000	Invalid object.
2012	Cannot create XML node.

17.2.2.8.6 DisableAttributeEntryHelper

Method: DisableAttributeEntryHelper ()

Description

DisableAttributeEntryHelper () disables the attribute entry helper in XMLSpy, Authentic Desktop and Authentic Browser plug-in.

Errors

2000	Invalid object.
------	-----------------

17.2.2.8.7 DisableElementEntryHelper

Method: DisableElementEntryHelper ()

Description

DisableElementEntryHelper () disables the element entry helper in XMLSpy, Authentic Desktop and Authentic Browser plug-in.

Errors

2000	Invalid object.
------	-----------------

17.2.2.8.8 DisableEntityEntryHelper

Method: DisableEntityEntryHelper ()

Description

DisableEntityEntryHelper () disables the entity entry helper in XMLSpy, Authentic Desktop and Authentic Browser plug-in.

Errors

2000	Invalid object.
------	-----------------

17.2.2.8.9 DocumentBegin

Property: DocumentBegin as [AuthenticRange](#)⁹¹⁶ (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.

17.2.2.8.10 DocumentEnd

Property: DocumentEnd as [AuthenticRange](#)⁹¹⁶ (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.
------	---

17.2.2.8.11 DoNotPerformStandardAction

Method: DoNotPerformStandardAction ()

Description

DoNotPerformStandardAction() serves as cancel bubble for macros, and stops further execution after macro has finished.

Errors

2000	Invalid object.
------	-----------------

17.2.2.8.12 EvaluateXPath

Method: EvaluateXPath (XMLData as [XMLData](#)⁹⁸⁴, strExpression as string) strValue as string

Return Value

The method returns a string

Description

EvaluateXPath() executes an XPath expression with the given XML context node. The result is returned as a 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.

17.2.2.8.13 Event

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.

17.2.2.8.14 EventContext

Property: EventContext() as EventContext

Description

EventContext property gives access to the running macros context. See the [EventContext](#)⁹¹³ interface description for more details.

Errors

2000	Invalid object.
------	-----------------

17.2.2.8.15 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](#)⁹⁵⁰). 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](#)⁹⁹⁹.

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.

17.2.2.8.16 Goto

Method: Goto (eKind as SPYAuthenticElementKind⁹⁹⁸, nCount as Long, eFrom as SPYAuthenticDocumentPosition⁹⁹⁸) as AuthenticRange⁹¹⁶

Description

Retrieve a range object that points to the beginning of the *nCount* element of type *eKind*. The start position is defined by the parameter *eFrom*. Use positive values for *nCount* to navigate to the document end. Use negative values to navigate towards the beginning of the document.

Errors

2000	The authentic view object is no longer valid.
2003	Target lies after end of document.
2004	Target lies before beginning of document.
2005	Invalid element kind specified. The document position to start from is not one of <i>spyAuthenticDocumentBegin</i> or <i>spyAuthenticDocumentEnd</i> . 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
```

17.2.2.8.17 **IsRedoEnabled**

Property: IsRedoEnabled as Boolean (read-only)

Description

True if redo steps are available and [Redo](#)⁹⁵⁹ is possible.

Errors

2000	The authentic view object is no longer valid.
2005	Invalid address for the return parameter was specified.

17.2.2.8.18 **IsUndoEnabled**

Property: IsUndoEnabled as Boolean (read-only)

Description

True if undo steps are available and [Undo](#)⁹⁶⁰ is possible.

Errors

2000	The authentic view object is no longer valid.
2005	Invalid address for the return parameter was specified.

17.2.2.8.19 MarkupVisibility

Property: MarkupVisibility as [SPYAuthenticMarkupVisibility](#)⁹⁹⁹

Description

Set or get current visibility of markup.

Errors

2000	The authentic view object is no longer valid.
2005	Invalid enumeration value was specified. Invalid address for the return parameter was specified.

17.2.2.8.20 Parent

Property: Parent as [Document](#)⁹⁶¹ (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.

17.2.2.8.21 Print

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.
------	---

17.2.2.8.22 Redo

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.

17.2.2.8.23 Selection

Property: Selection as [AuthenticRange](#)⁹¹⁶

Description

Set or get current text selection in user interface.

Errors

2000	The authentic view object is no longer valid.
2002	No cursor selection is active.
2005	Invalid address for the return parameter was specified.

Examples

```
' -----
' Scripting environment - VBScript
' -----
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

' if we are the end of the document, re-start at the beginning
If (objAuthenticView.Selection.IsEqual(objAuthenticView.DocumentEnd)) Then
    objAuthenticView.Selection = objAuthenticView.DocumentBegin
Else
    ' objAuthenticView.Selection = objAuthenticView.Selection.GotoNextCursorPosition()
    ' or shorter:
    objAuthenticView.Selection.GotoNextCursorPosition().Select
End If
```

17.2.2.8.24 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](#)⁹⁵⁰). 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](#)⁹⁹⁹.

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.

17.2.2.8.25 Undo

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.

17.2.2.8.26 UpdateXMLInstanceEntities

Method: `UpdateXMLInstanceEntities()`

Description

Updates the internal representation of the declared entities, and refills the entry helper. In addition, the validator is reloaded, allowing the XML file to validate correctly. Please note that this may also cause schema files to be reloaded.

Errors

The method never returns an error.

Example

```
// -----
// Scripting environment - JavaScript
// -----
if(Application.ActiveDocument && (Application.ActiveDocument.CurrentViewMode == 4))
{
    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();
}
}

```

17.2.2.8.27 WholeDocument

Property: WholeDocument as [AuthenticRange](#)⁹¹⁶ (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.

17.2.2.8.28 XMLDataRoot

Property: XMLDataRoot as [XMLData](#)⁹⁸⁴ (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.

17.2.2.9 Document

The `Document` interface has the following properties and methods.

Methods

- [Activate](#)⁹⁶³
- [Application](#)⁹⁶³
- [AssignWorkingXMLFile](#)⁹⁶³

- [Close](#)⁹⁶⁴
- [Save](#)⁹⁶⁶
- [SaveAs](#)⁹⁶⁶
- [Saved](#)⁹⁶⁷
- [SaveGeneratedFOFile](#)⁹⁶⁷
- [SaveGeneratedFOFileEx](#)⁹⁶⁷
- [SaveGeneratedHTMLFile](#)⁹⁶⁷
- [SaveGeneratedHTMLFileEx](#)⁹⁶⁸
- [SaveGeneratedPDFFile](#)⁹⁶⁸
- [SaveGeneratedPDFFileEx](#)⁹⁶⁸
- [SaveGeneratedRTFFile](#)⁹⁶⁸
- [SaveGeneratedRTFFileEx](#)⁹⁶⁹
- [SaveGeneratedWord2007File](#)⁹⁶⁹
- [SaveGeneratedWord2007FileEx](#)⁹⁶⁹
- [SaveGeneratedTextFile](#)⁹⁷⁰
- [SaveGeneratedTextFileEx](#)⁹⁷⁰
- [SaveGeneratedXSLTFOFile](#)⁹⁷⁰
- [SaveGeneratedXSLTFOFileEx](#)⁹⁷⁰
- [SaveGeneratedXSLTHTMLFile](#)⁹⁷¹
- [SaveGeneratedXSLTHTMLFileEx](#)⁹⁷¹
- [SaveGeneratedXSLTRTFFile](#)⁹⁷¹
- [SaveGeneratedXSLTRTFFileEx](#)⁹⁷²
- [SaveGeneratedXSLTWord2007File](#)⁹⁷²
- [SaveGeneratedXSLTWord2007FileEx](#)⁹⁷²
- [SaveGeneratedXSLTTextFile](#)⁹⁷²
- [SaveGeneratedXSLTTextFileEx](#)⁹⁷³

Properties

- [Application](#)⁹⁶³
- [FullName](#)⁹⁶⁴
- [Name](#)⁹⁶⁵
- [Parameters](#)⁹⁶⁵
- [Parent](#)⁹⁶⁵
- [Path](#)⁹⁶⁶
- [SchemaSources](#)⁹⁷³

17.2.2.9.1 Events

17.2.2.9.1.1 *OnDocumentClosed*

Event: OnDocumentClosed(*Document* as [Document](#)⁹⁶¹)

Description

This event gets fired as a result of closing a document.

17.2.2.9.1.2 *OnModifiedFlagChanged*

Event: OnModifiedFlagChanged(as Boolean)

Description

Returns true if the Modified flag has been changed.

17.2.2.9.2 *Activate*

Method: Activate ()

Description

Activate document frame.

Errors

1200	Document object is invalid.
1201	Invalid input parameter.

17.2.2.9.3 *Application*

Property: Application as [Application](#)⁸⁹⁸ (read-only)

Description

Accesses the StyleVision application object.

Errors

140011 11	The object is no longer valid.
140711 00	Invalid address for the return parameter was specified.

17.2.2.9.4 *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

17.2.2.9.5 Close

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.

17.2.2.9.6 FullName

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](#)⁹⁶⁴ and [SetPathName](#)⁹⁷³ obsolete.

Errors

1400	The document object is no longer valid.
1402	Empty string has been specified as full file name.

17.2.2.9.7 GetPathName (obsolete)

Superseded by [Document.FullName](#)⁹⁶⁴

```
// ---- javascript sample ----
// instead of:
// strPathName = Application.ActiveDocument.GetPathName();
// use now:
strPathName = Application.ActiveDocument.FullName;
```

Method: GetPathName() as String

Description

The method `GetPathName` gets the path of the active document.

See also [Document.SetPathName](#)⁹⁷³ (obsolete).

17.2.2.9.8 Name

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](#)⁹⁶⁴.

Errors

1400	The document object is no longer valid.
1407	Invalid address for the return parameter was specified.

17.2.2.9.9 Parameters

Property: Parameters as Parameters (read-only)

Description

Reference to the current `Parameters` object.

Errors

1200	Document object is invalid.
1201	Invalid input parameter.

17.2.2.9.10 Parent

Property: Parent as [Documents](#)⁹⁷⁴ (read-only)

Description

Access the parent of the document object.

Errors

1400	The document object is no longer valid.
1407	Invalid address for the return parameter was specified.

Property: Parent as [Application](#)⁸⁹⁸ (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.

17.2.2.9.11 Path

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](#)⁹⁶⁴.

Errors

1400	The document object is no longer valid.
1407	Invalid address for the return parameter was specified.

17.2.2.9.12 Save

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.

17.2.2.9.13 SaveAs

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.

17.2.2.9.14 Saved

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.

17.2.2.9.15 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.

17.2.2.9.16 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.

17.2.2.9.17 SaveGeneratedHTMLFile

Method: SaveGeneratedHTMLFile (*strFileName* as String)

Description

Saves the generated file to the location specified.

Errors

1201	Invalid document object.
------	--------------------------

1204	Cannot generate output file.
------	------------------------------

17.2.2.9.18 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.

17.2.2.9.19 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.

17.2.2.9.20 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.

17.2.2.9.21 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.

17.2.2.9.22 SaveGeneratedRTFFileEx

Method: SaveGeneratedRTFFileEx (*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.

17.2.2.9.23 SaveGeneratedWord2007File

Method: SaveGeneratedWord2007File (*strFileName* as String)

Description

Saves the generated file to the location specified.

Errors

1201	Invalid document object.
1204	Cannot generate output file.

17.2.2.9.24 SaveGeneratedWord2007FileEx

Method: SaveGeneratedWord2007FileEx (*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.

17.2.2.9.25 SaveGeneratedTextFile

Method: SaveGeneratedTextFile (*strFileName* as String)

Description

Saves the generated file to the location specified.

Errors

1201	Invalid document object.
1204	Cannot generate output file.

17.2.2.9.26 SaveGeneratedTextFileEx

Method: SaveGeneratedTextFileEx (*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.

17.2.2.9.27 SaveGeneratedXSLTFOFile

Method: SaveGeneratedXSLTFOFile (*strFileName* as String)

Description

Saves the generated file to the location specified.

Errors

1201	Invalid document object.
1204	Cannot generate output file.

17.2.2.9.28 SaveGeneratedXSLTFOFileEx

Method: SaveGeneratedXSLTFOFileEx (*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.

17.2.2.9.29 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.

17.2.2.9.30 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.

17.2.2.9.31 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.

17.2.2.9.32 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.

17.2.2.9.33 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.

17.2.2.9.34 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.

17.2.2.9.35 SaveGeneratedXSLTTextFile

Method: SaveGeneratedXSLTTextFile (*strFileName* as String)

Description

Saves the generated file to the location specified.

Errors

1201	Invalid document object.
1204	Cannot generate output file.

17.2.2.9.36 SaveGeneratedXSLTTextFileEx

Method: SaveGeneratedXSLTTextFileEx (*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.

17.2.2.9.37 SchemaSources

Property: SchemaSources as SchemaSources (read-only)

Description

Reference to the current `SchemaSources` object.

Errors

1200	Document object is invalid.
1201	Invalid input parameter.

17.2.2.9.38 SetPathName (obsolete)

Superseded by [Document.FullName](#) ⁹⁶⁴

```
// ---- javascript sample ----
// instead of:
// Application.ActiveDocument.SetPathName("C:\\myXMLFiles\\test.xml");
// use now:
Application.ActiveDocument.FullName = "C:\\myXMLFiles\\test.xml";
```

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.

17.2.2.10 Documents

The `Documents` interface has the following methods and properties:

Methods

- [ActiveDocument](#)⁹⁷⁴
- [Item](#)⁹⁷⁵
- [NewDocument](#)⁹⁷⁵
- [OpenDocument](#)⁹⁷⁶

Properties

- [Application](#)⁹⁷⁴
- [Count](#)⁹⁷⁵
- [Parent](#)⁹⁷⁶

17.2.2.10.1 ActiveDocument

Property: ActiveDocument as [Document](#)⁹⁶¹

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.
1600	Invalid <code>Documents</code> object
1601	Invalid input parameter

17.2.2.10.2 Application

Property: Application as [Application](#)⁸⁹⁸ (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.

17.2.2.10.3 Count

Property: Count as long

Description

Count of open documents.

Errors

1600	Invalid Documents object
1601	Invalid input parameter

17.2.2.10.4 Item

Method: Item (*n* as long) as [Document](#)⁹⁶¹

Description

Gets the document with the index *n* in this collection. Index is 1-based.

Errors

1600	Invalid Documents object
1601	Invalid input parameter

17.2.2.10.5 NewDocument

Method: NewDocument() as [Document](#)⁹⁶¹

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

17.2.2.10.6 OpenDocument

Method: OpenDocument(strFileName as *String*) as [Document](#)⁹⁶¹

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 <code>Documents</code> object
1601	Invalid input parameter

17.2.2.10.7 Parent

Property: Parent as [Application](#)⁸⁹⁸ (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.

17.2.2.11 Parameter

The `Parameter` interface has the following properties:

Properties

- [Application](#)⁹⁷⁷
- [Name](#)⁹⁷⁷
- [Parent](#)⁹⁷⁷
- [Value](#)⁹⁷⁷

17.2.2.11.1 Application

Property: Application as [Application](#)⁸⁹⁸ (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.

17.2.2.11.2 Name

Property: Name as String (read-only)

Errors

1600	Invalid Documents object
1601	Invalid input parameter

17.2.2.11.3 Parent

Property: Parent as [Application](#)⁸⁹⁸ (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.

17.2.2.11.4 Value

Property: Value as String (read-only)

Errors

1600	Invalid Documents object
1601	Invalid input parameter

17.2.2.12 Parameters

The `Parameters` interface has the following properties:

Properties

- [Application](#)⁹⁷⁸
- [Count](#)⁹⁷⁸
- [Item](#)⁹⁷⁸
- [Parent](#)⁹⁷⁹

17.2.2.12.1 Application

Property: Application as [Application](#)⁸⁹⁸ (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.

17.2.2.12.2 Count

Property: Count as long

Description

Count of parameters.

Errors

1600	Invalid <code>Parameters</code> object
1601	Invalid input parameter

17.2.2.12.3 Item

Method: Item (*n* as long) as [Document](#)⁹⁶¹

Description

Gets the document with the index *n* in this collection. Index is 1-based.

Errors

1600	Invalid <code>Documents</code> object
1601	Invalid input parameter

17.2.2.12.4 Parent

Property: Parent as [Application](#)⁸⁹⁸ (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.

17.2.2.13 SchemaSource

The `SchemaSource` interface has the following properties:

Properties

- [Application](#)⁹⁷⁹
- [IsMainSchemaSource](#)⁹⁸⁰
- [Name](#)⁹⁸⁰
- [Parent](#)⁹⁸⁰
- [SchemaFileName](#)⁹⁸⁰
- [TemplateFileName](#)⁹⁸¹
- [Type](#)⁹⁸¹
- [TypeName](#)⁹⁸¹
- [WorkingXMLFileName](#)⁹⁸²

17.2.2.13.1 Application

Property: Application as [Application](#)⁸⁹⁸ (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.

17.2.2.13.2 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.

17.2.2.13.3 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.

17.2.2.13.4 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.

17.2.2.13.5 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

17.2.2.13.6 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

17.2.2.13.7 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.

17.2.2.13.8 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.

17.2.2.13.9 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

17.2.2.14 SchemaSources

The `SchemaSources` interface has the following properties:

Properties

- [Application](#) ⁹⁸²
- [MainSchemaSource](#) ⁹⁸³
- [Parent](#) ⁹⁸³

17.2.2.14.1 Application

Property: Application as [Application](#) ⁸⁹⁸ (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.

17.2.2.14.2 Count

Property: Count as long (read-only)

Description

Count of schema sources.

Errors

1300	Invalid <code>SchemaSources</code> object
1301	Invalid input parameter

17.2.2.14.3 Item

Method: Item (*n* as long) as [Document](#) ⁹⁶¹

Description

Gets the document with the index *n* in this collection. Index is 1-based.

Errors

1300	Invalid <code>SchemaSources</code> object
1301	Invalid input parameter

17.2.2.14.4 MainSchemaSource

Property: MainSchemaSource as SchemaSource (read-only)

Errors

1300	Invalid <code>SchemaSources</code> object
1301	Invalid input parameter

17.2.2.14.5 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.

17.2.2.15 XMLData

Properties

[Kind](#)⁹⁹⁴

[Name](#)⁹⁹⁵

[TextValue](#)⁹⁹⁶

[HasChildren](#)⁹⁹²

[MayHaveChildren](#)⁹⁹⁴

[Parent](#)⁹⁹⁵

Methods

[GetFirstChild](#)⁹⁹⁰

[GetNextChild](#)⁹⁹⁰

[GetCurrentChild](#)⁹⁸⁹

[InsertChild](#)⁹⁹³

[InsertChildAfter](#)⁹⁹³

[InsertChildBefore](#)⁹⁹³

[AppendChild](#)⁹⁸⁵

[EraseAllChildren](#)⁹⁸⁶

[EraseChild](#)⁹⁸⁷

[EraseCurrentChild](#)⁹⁸⁷

[IsSameNode](#)⁹⁹⁴

[CountChildren](#)⁹⁸⁵

[CountChildrenKind](#)⁹⁸⁶

[GetChild](#)⁹⁸⁸

[GetChildAttribute](#)⁹⁸⁸

[GetChildElement](#)⁹⁸⁸

[GetChildKind](#)⁹⁸⁹

[GetNamespacePrefixForURI](#)⁹⁹⁰

[HasChildrenKind](#)⁹⁹²

[SetTextValueXMLEncoded](#)⁹⁹⁵

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](#)⁹⁹⁵ 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.

17.2.2.15.1 AppendChild

Declaration: AppendChild (pNewData as [XMLData](#)⁹⁸⁴)

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

```
Dim objCurrentParent As XMLData
Dim objNewChild As XMLData

Set objNewChild = objSpy.ActiveDocument.CreateChild(spyXMLDataElement)
Set objCurrentParent = objSpy.ActiveDocument.RootElement

objCurrentParent.AppendChild objNewChild

Set objNewChild = Nothing
```

17.2.2.15.2 CountChildren

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.
------	--

17.2.2.15.3 CountChildrenKind

Declaration: CountChildrenKind (*nKind* as [SPYXMLDataKind](#)¹⁰⁰¹) 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.
------	--

17.2.2.15.4 EraseAllChildren

Declaration: EraseAllChildren

Description

EraseAllChildren deletes all associated children of the XMLData object.

Errors

1500	The XMLData object is no longer valid.
1900	Document must not be modified

Example

The sample erases all elements of the active document.

```
Dim objCurrentParent As XMLData
```

```
Set objCurrentParent = objSpy.ActiveDocument.RootElement
objCurrentParent.EraseAllChildren
```

17.2.2.15.5 EraseChild

Method: EraseChild (Child as [XMLData](#)⁹⁸⁴)

Description

Deletes the given child node.

Errors

1500	Invalid object.
1506	Invalid input xml
1510	Invalid parameter.

17.2.2.15.6 EraseCurrentChild

Declaration: EraseCurrentChild

Description

EraseCurrentChild deletes the current XMLData child object. Before you call EraseCurrentChild you must initialize an internal iterator with [XMLData.GetFirstChild](#)⁹⁹⁰. 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
var objList = Application.ActiveDocument.GridView.CurrentFocus;

// 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); }
}

```

17.2.2.15.7 GetChild

Declaration: GetChild (*position* as long) as [XMLData](#)⁹⁸⁴

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.

17.2.2.15.8 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.

17.2.2.15.9 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.

17.2.2.15.10 [GetChildKind](#)

Declaration: `GetChildKind` (*position* as long, *nKind* as [SPYXMLDataKind](#)¹⁰⁰¹) as [XMLData](#)⁹⁸⁴

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.

17.2.2.15.11 [GetCurrentChild](#)

Declaration: `GetCurrentChild` as [XMLData](#)⁹⁸⁴

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](#)⁹⁹⁰.

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.

17.2.2.15.12 GetFirstChild

Declaration: GetFirstChild (*nKind* as [SPYXMLDataKind](#)¹⁰⁰¹) as [XMLData](#)⁹⁸⁴

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](#)⁹⁸⁹, [GetNextChild](#)⁹⁹⁰ or [EraseCurrentChild](#)⁹⁸⁷.

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](#)⁹⁹⁰.

17.2.2.15.13 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.

17.2.2.15.14 GetNextChild

Declaration: GetNextChild as [XMLData](#)⁹⁸⁴

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](#)⁹⁹⁰.

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);
    }

```

17.2.2.15.15 GetTextValueXMLDecoded

Method: GetTextValueXMLDecoded ()as string

Description

Gets the decoded text value of the XML.

Errors

1500	Invalid object.
1510	Invalid parameter.

17.2.2.15.16 HasChildren

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.

17.2.2.15.17 HasChildrenKind

Declaration: HasChildrenKind (*nKind* as [SPYXMLDataKind](#)¹⁰⁰¹) 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.

17.2.2.15.18 InsertChild

Declaration: InsertChild (pNewData as [XMLData](#)⁹⁸⁴)

Description

InsertChild inserts the new child before the current child (see also [XMLData.GetFirstChild](#)⁹⁹⁰, [XMLData.GetNextChild](#)⁹⁹⁰ 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

17.2.2.15.19 InsertChildAfter

Method: InsertChildAfter (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

17.2.2.15.20 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

17.2.2.15.21 IsSameNode

Declaration: IsSameNode (pNodeToCompare as [XMLData](#)⁹⁸⁴) 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.

17.2.2.15.22 Kind

Declaration: Kind as [SPYXMLDataKind](#)¹⁰⁰¹

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.

17.2.2.15.23 MayHaveChildren

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.

17.2.2.15.24 Name

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.

17.2.2.15.25 Parent

Declaration: Parent as [XMLData](#)⁹⁸⁴

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.

17.2.2.15.26 SetTextValueXMLEncoded

Method: SetTextValueXMLEncoded (*strVal* as [String](#)¹⁰⁰¹)

Description

Sets the encoded text value of the XML.

Errors

1500	Invalid object.
1513	Modification not allowed.

17.2.2.15.27 TextValue

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.

17.2.3 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.

17.2.3.1 ENUMApplicationStatus

Enumeration to specify the current Application status.

eApplicationRunning	= 0
eApplicationAfterLicenseCheck	= 1
eApplicationBeforeLicenseCheck	= 2
eApplicationConcurrentLicenseCheckFailed	= 3
eApplicationProcessingCommandLine	= 4

17.2.3.2 ENUMAppOutputLine_Severity

Enumeration values to identify the severity of an AppOutputLine.

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

17.2.3.3 ENUMAppOutputLine_TextDecoration

Enumeration values for the different kinds of text decoration of an AppOutputLine.

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_UnkownParam	= 10

17.2.3.4 ENUMSchemaSourceType

Enumeration to specify the source schema type: XML Schema, DTD, DB, DB cell, User-Defined, or XBRL.

eSchemaSourceType_XSDorDTD	= 0
eSchemaSourceType_DB	= 1
eSchemaSourceType_DBCell	= 2
eSchemaSourceType_User	= 3

eSchemaSourceType_XBRL	= 4
------------------------	-----

17.2.3.5 ENUMSchemaType

Enumeration to specify the schema type: W3C XML Schema or DTD.

eSchemaTypeW3CSchema	= 0
eSchemaTypeDTD	= 1

17.2.3.6 SPYAuthenticActions

Actions that can be performed on [AuthenticRange](#)⁹¹⁶ objects.

spyAuthenticInsertAt	= 0
spyAuthenticApply	= 1
spyAuthenticClearSurr	= 2
spyAuthenticAppend	= 3
spyAuthenticInsertBefore	= 4
spyAuthenticRemove	= 5

17.2.3.7 SPYAuthenticDocumentPosition

Relative and absolute positions used for navigating with [AuthenticRange](#)⁹¹⁶ objects.

spyAuthenticDocumentBegin	= 0
spyAuthenticDocumentEnd	= 1
spyAuthenticRangeBegin	= 2
spyAuthenticRangeEnd	= 3

17.2.3.8 SPYAuthenticElementKind

Enumeration of the different kinds of elements used for navigation and selection within the [AuthenticRange](#)⁹¹⁶ and [AuthenticView](#)⁹⁴⁶ objects.

spyAuthenticChar	= 0
spyAuthenticWord	= 1

spyAuthenticLine	= 3
spyAuthenticParagraph	= 4
spyAuthenticTag	= 6
spyAuthenticDocument	= 8
spyAuthenticTable	= 9
spyAuthenticTableRow	= 10
spyAuthenticTableColumn	= 11

17.2.3.9 SPYAuthenticMarkupVisibility

Enumeration values to customize the visibility of markup with [MarkupVisibility](#)⁹⁵⁸.

spyAuthenticMarkupHidden	= 0
spyAuthenticMarkupSmall	= 1
spyAuthenticMarkupLarge	= 2
spyAuthenticMarkupMixed	= 3

17.2.3.10 SPYAuthenticToolBarButtonState

Authentic toolbar button states are given by the following enumerations.

authenticToolBarButtonDefault	= 0
authenticToolBarButtonEnabled	= 1
authenticToolBarButtonDisabled	= 2

17.2.3.11 SPYMouseEvent

Enumeration type that defines the mouse status during a mouse event. Use the enumeration values as bitmasks rather than directly comparing with them.

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

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
```

17.2.3.12 SPYValidateXSDVersion

Description

Enumeration values that select what XSD version to use. The XSD version that is selected depends on both (i) the presence/absence—and, if present, the value—of the `/xs:schema/@vc:minVersion` attribute of the XSD document, and (ii) the value of this enumeration.

spyValidateXSDVersion_AutoDetect	= 0
spyValidateXSDVersion_1_1	= 1
spyValidateXSDVersion_1_0	= 2

`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.

17.2.3.13 SPYValidateErrorFormat

Enumeration values that select the format of the error message.

spyValidateErrorFormat_Text	= 0
spyValidateErrorFormat_ShortXML	= 1
spyValidateErrorFormat_LongXML	= 2

17.2.3.14 SPYXMLDataKind

The different types of XMLData elements available for XML documents.

spyXMLDataXMLDocStruct	= 0
spyXMLDataXMLEntityDocStruct	= 1
spyXMLDataDTDDocStruct	= 2
spyXMLDataXML	= 3
spyXMLDataElement	= 4
spyXMLDataAttr	= 5
spyXMLDataText	= 6
spyXMLDataCDATA	= 7
spyXMLDataComment	= 8
spyXMLDataPI	= 9
spyXMLDataDefDoctype	= 10
spyXMLDataDefExternalID	= 11
spyXMLDataDefElement	= 12
spyXMLDataDefAttlist	= 13
spyXMLDataDefEntity	= 14
spyXMLDataDefNotation	= 15
spyXMLDataKindsCount	= 16

17.3 ActiveX Integration

The StyleVision user interface and the functionality described in this section can be integrated into custom applications that can consume ActiveX controls. ActiveX technology enables a wide variety of languages to be used for integration, such as C++, C#, and VB.NET. All components are full OLE Controls. Integration into Java is provided through wrapper classes.

To integrate the ActiveX controls into your custom code, the StyleVision Integration Package must be installed (see <https://www.altova.com/components/download>). Ensure that you install StyleVision first, and then the StyleVision Integration Package. Other prerequisites apply, depending on language and platform (see [Prerequisites](#)¹⁰⁰²).

You can flexibly choose between two different levels of integration: application level and document level.

Integration at application level means embedding the complete interface of StyleVision (including its menus, toolbars, panes, etc) as an ActiveX control into your custom application. For example, in the most simple scenario, your custom application could consist of only one form that embeds the StyleVision graphical user interface. This approach is easier to implement than integration at document level but may not be suitable if you need flexibility to configure the StyleVision graphical user interface according to your custom requirements.

Integration at document level means embedding StyleVision into your own application piece-by-piece. This includes implementing not only the main StyleVision control but also the main document editor window, and, optionally, any additional windows. This approach provides greater flexibility to configure the GUI, but requires advanced interaction with ActiveX controls in your language of choice.

The sections [Integration at the Application Level](#)¹⁰⁰⁵ and [Integration at Document Level](#)¹⁰⁰⁷ describe the key steps at these respective levels. The [ActiveX Integration Examples](#)¹⁰¹⁰ section provides examples in C# and Java. Looking through these examples will help you to make the right decisions quickly. The [Object Reference](#)¹⁰³⁷ section describes all COM objects that can be used for integration, together with their properties and methods.

For information about using StyleVision as a Visual Studio plug-in, see [StyleVision in Visual Studio](#)⁶⁷⁰.

17.3.1 Prerequisites

To integrate the StyleVision ActiveX control into a custom application, the following must be installed on your computer:

- StyleVision
- The StyleVision Integration Package, available for download at <https://www.altova.com/components/download>

To integrate the 64-bit ActiveX control, install the 64-bit versions of StyleVision and StyleVision Integration Package. For applications developed under Microsoft .NET platform with Visual Studio, both the 32-bit and 64-bit versions of StyleVision and StyleVision Integration Package must be installed, as explained below.

Microsoft .NET (C#, VB.NET) with Visual Studio

To integrate the StyleVision ActiveX control into a 32-bit application developed under Microsoft .NET, the following must be installed on your computer:

- Microsoft .NET Framework 4.0 or later
- Visual Studio 2012/2013/2015/2017/2019/2022
- StyleVision 32-bit and StyleVision Integration Package 32-bit
- The ActiveX controls must be added to the Visual Studio toolbox (see [Adding the ActiveX Controls to the Toolbox](#)¹⁰⁰³).

If you want to integrate the 64-bit ActiveX control, the following prerequisites apply in addition to the ones above:

- StyleVision 32-bit and StyleVision Integration Package 32-bit must still be installed (this is required to provide the 32-bit ActiveX control to the Visual Studio designer, since Visual Studio runs on 32-bit)
- StyleVision 64-bit and StyleVision Integration Package 64-bit must be installed (provides the actual 64-bit ActiveX control to your custom application at runtime)
- In Visual Studio, create a 64-bit build configuration and build your application using this configuration. For an example, see [Running the Sample C# Solution](#)¹⁰¹⁰.

Java

To integrate the StyleVision ActiveX control into Java application using the Eclipse development environment, the following must be installed on your computer:

- Java Runtime Environment (JRE) or Java Development Kit (JDK) 7 or later
- Eclipse
- StyleVision and StyleVision Integration Package

Note: To run the 64-bit version of the StyleVision ActiveX control, use a 64-bit version of Eclipse, as well as the 64-bit version of StyleVision and the StyleVision Integration Package.

StyleVision integration and deployment on client computers

If you create a .NET application and intend to distribute it to other clients, you will need to install the following on the client computer(s):

- StyleVision
- The StyleVision Integration Package
- The custom integration code or application.

17.3.2 Adding the ActiveX Controls to the Toolbox

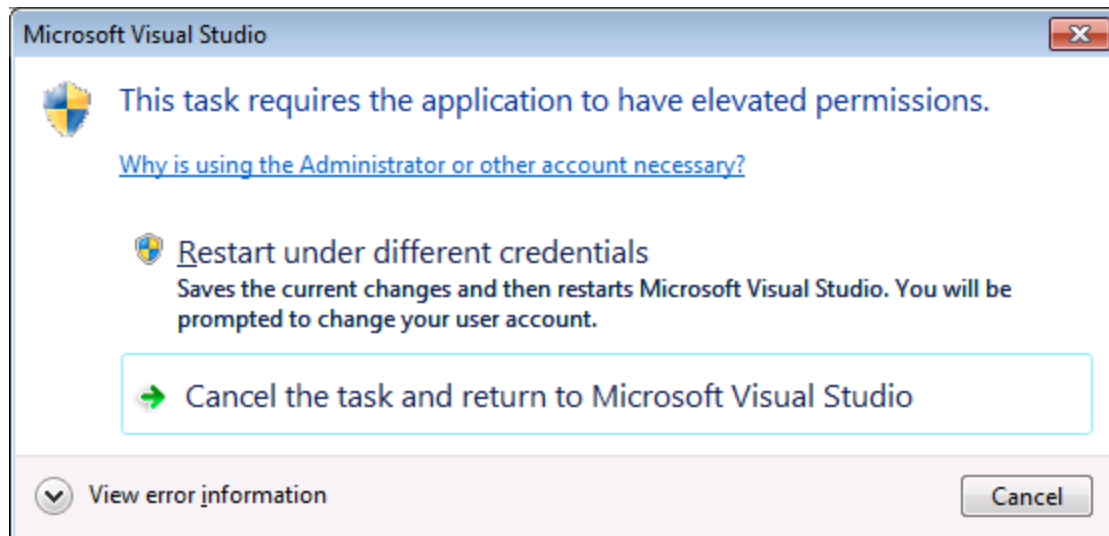
To use the StyleVision ActiveX controls in an application developed with Visual Studio, the controls must first be added to the Visual Studio Toolbox, as follows:

1. On the **Tools** menu of Visual Studio, click **Choose Toolbox Items**.

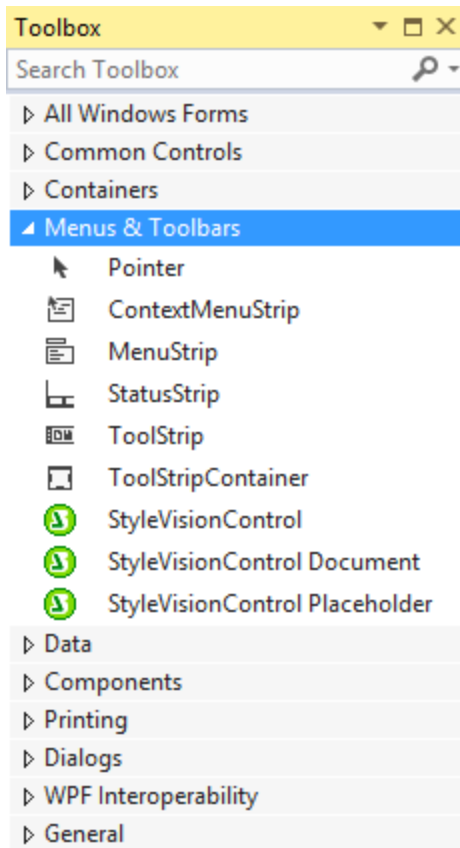
2. On the **COM Components** tab, select the check boxes next to the StyleVisionControl, StyleVisionControl Document, and StyleVisionControl Placeholder.

In case the controls above are not available, follow the steps below:

1. On the **COM Components** tab, click **Browse**, and select the **StyleVisionControl.ocx** file from the StyleVision installation folder. Remember that the StyleVision Integration Package must be installed; otherwise, this file is not available, see [Prerequisites](#)¹⁰⁰².
2. If prompted to restart Visual Studio with elevated permissions, click **Restart under different credentials**.



If the steps above were successful, the StyleVision ActiveX controls become available in the Visual Studio Toolbox.



Note: For an application-level integration, only the **StyleVisionControl** ActiveX control is used (see [Integration at Application Level](#)¹⁰⁰⁵). The **StyleVisionControl Document** and **StyleVisionControl Placeholder** controls are used for document-level integration (see [Integration at Document Level](#)¹⁰⁰⁷).

17.3.3 Integration at Application Level

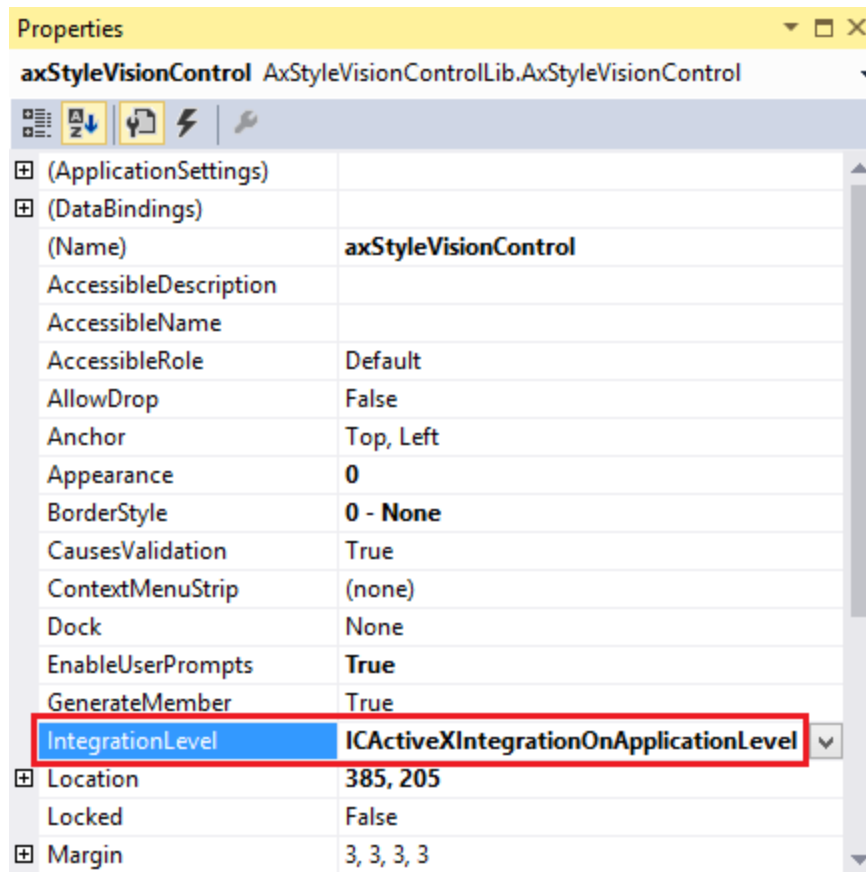
Integration at application level allows you to embed the complete interface of StyleVision into a window of your application. With this type of integration, you get the whole user interface of StyleVision, including all menus, toolbars, the status bar, document windows, and helper windows. Customization 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](#)¹⁰⁴¹. Do not instantiate or access [StyleVisionControlDocument](#)¹⁰⁴⁸ or [StyleVisionControlPlaceholder](#)¹⁰⁵⁵ ActiveX controls when integrating at application-level.

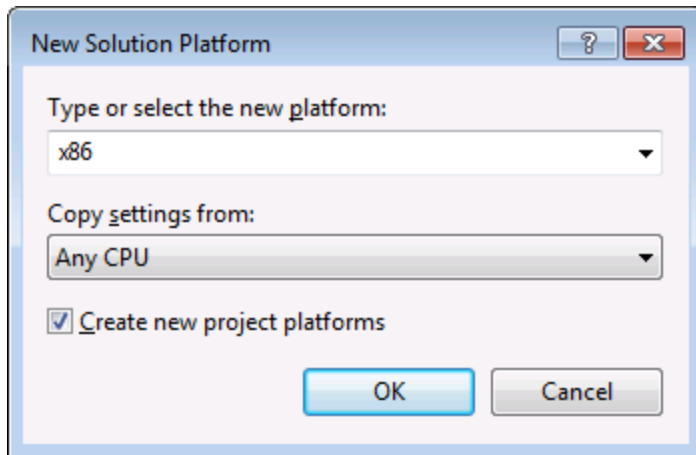
If you have any initialization to do or if you want to automate some behaviour of StyleVision, use the properties, methods, and events described for [StyleVisionControl](#)¹⁰⁴¹. Consider using [StyleVisionControl.Application](#)¹⁰⁴² for more complex access to StyleVision functionality.

In C# or VB.NET with Visual Studio, the steps to create a basic, one-form application which integrates the StyleVision ActiveX controls at application level are as follows:

1. Check that all prerequisites are met (see [Prerequisites](#)¹⁰⁰²).
2. Create a new Visual Studio Windows Forms project with a new empty form.
3. If you have not done that already, add the ActiveX controls to the toolbox (see [Adding the ActiveX Controls to the Toolbox](#)¹⁰⁰³).
4. Drag the **StyleVisionControl** from the toolbox onto your new form.
5. Select the **StyleVisionControl** on the form, and, in the Properties window, set the **IntegrationLevel** property to **ICActiveXIntegrationOnApplicationLevel**.



6. Create a build platform configuration that matches the platform under which you want to build (x86, x64). Here is how you can create the build configuration:
 - a. Right-click the solution in Visual Studio, and select **Configuration Manager**.
 - b. Under **Active solution platform**, select **New...** and then select the x86 or x64 configuration (in this example, **x86**).



You are now ready to build and run the solution in Visual Studio. Remember to build using the configuration that matches your target platform (x86, x64).

17.3.4 Integration at Document Level

Compared to integration at application level, integration at document level is a more complex, yet more flexible way to embed StyleVision functionality into your application by means of ActiveX controls. With this approach, your code can access selectively the following parts of the StyleVision user interface:

- Document editing window
- Project window
- Design Overview window
- Schema Tree window
- Design Tree window
- Styles window
- Style Repository window
- Properties window
- Messages window

As mentioned in [Integration at Application Level](#)¹⁰⁰⁵, for an ActiveX integration at application level, only one control is required, namely the **StyleVisionControl**. However, for an ActiveX integration at document level, StyleVision functionality is provided by the following ActiveX controls:

- [StyleVisionControl](#)¹⁰⁴¹
- [StyleVisionControl Document](#)¹⁰⁴⁸
- [StyleVisionControl Placeholder](#)¹⁰⁵⁵

These controls are supplied by the StyleVisionControl.ocx file available in the application installation folder of StyleVision. When you develop the ActiveX integration with Visual Studio, you will need to add these controls to the Visual Studio toolbox (see [Adding the ActiveX Controls to the Toolbox](#)¹⁰⁰³).

The basic steps to integrate the ActiveX controls at document level into your application are as follows:

1. First, instantiate StyleVisionControl in your application. Instantiating this control is mandatory; it enables support for the StyleVisionControl Document and StyleVisionControl Placeholder controls

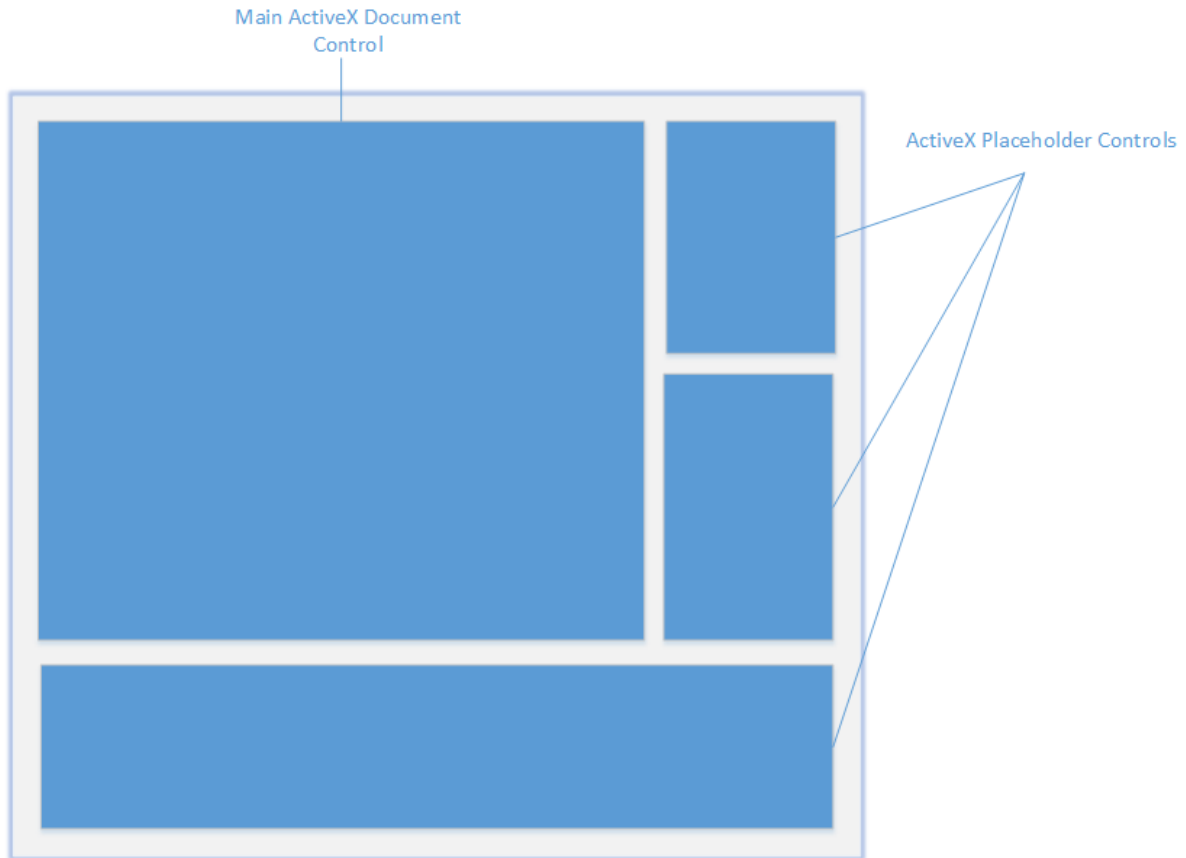
mentioned above. It is important to set the [IntegrationLevel](#)⁽¹⁰⁴³⁾ property to `ICActiveXIntegrationOnDocumentLevel` (or "1"). To hide the control from the user, set its `Visible` property to `False`. Note that, when integrating at document level, do not use the `Open` method of the `StyleVisionControl`; this might lead to unexpected results. Use the corresponding open methods of `StyleVisionControl Document` and `StyleVisionControl Placeholder` instead.

2. Create at least one instance of `StyleVisionControl Document` in your application. This control supplies the document editing window of `StyleVision` to your application and can be instantiated multiple times if necessary. Use the method `Open` to load any existing file. To access document-related functionality, use the `Path` and `Save` or methods and properties accessible via the property `Document`. Note that the control does not support a read-only mode. The value of the property `ReadOnly` is ignored.
3. Optionally, add to your application the `StyleVisionControl Placeholder` control for each additional window (other than the document window) that must be available to your application. Instances of `StyleVisionControl Placeholder` allow you to selectively embed additional windows of `StyleVision` into your application. The window kind (for example, Project window) is defined by the property `PlaceholderWindowID`. Therefore, to set the window kind, set the property `PlaceholderWindowID`. For valid window identifiers, see [StyleVisionControlPlaceholderWindow](#)⁽¹⁰⁵⁸⁾. Use only one `StyleVisionControl Placeholder` for each window identifier.

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.

For example, in C# or VB.NET with Visual Studio, the steps to create a basic, one-form application which integrates the `StyleVision` ActiveX controls at document level could be similar to those listed below. Note that your application may be more complex if necessary; however, the instructions below are important to understand the minimum requirements for an ActiveX integration at document level.

1. Create a new Visual Studio Windows Forms project with a new empty form.
2. If you have not done that already, add the ActiveX controls to the toolbox (see [Adding the ActiveX Controls to the Toolbox](#)⁽¹⁰⁰³⁾).
3. Drag the [StyleVisionControl](#)⁽¹⁰⁴¹⁾ from the toolbox onto your new form.
4. Set the `IntegrationLevel` property of the `StyleVisionControl` to `ICActiveXIntegrationOnDocumentLevel`, and the `Visible` property to `False`. You can do this either from code or from the Properties window.
5. Drag the [StyleVisionControl Document](#)⁽¹⁰⁴⁸⁾ from the toolbox onto the form. This control provides the main document window of `StyleVision` to your application, so you may need to resize it to a reasonable size for a document.
6. Optionally, add one or more [StyleVisionControl Placeholder](#)⁽¹⁰⁵⁶⁾ controls to the form (one for each additional window type that your application needs, for example, the Project window). You will typically want to place such additional placeholder controls either below or to the right or left of the main document control, for example:



7. Set the PlaceholderWindowID property of each StyleVisionControl Placeholder control to a valid window identifier. For the list of valid values, see [StyleVisionControlPlaceholderWindow](#)¹⁰⁵⁸.
8. Add commands to your application (at minimum, you will need to open, save and close documents), as shown below.

Querying StyleVision Commands

When you integrate at document level, no StyleVision menu or toolbar is available to your application. Instead, you can retrieve the required commands, view their status, and execute them programmatically, as follows:

- To retrieve all available commands, use the [CommandsList](#)¹⁰⁴² property of the StyleVisionControl.
- To retrieve commands organized according to their menu structure, use the [MainMenu](#)¹⁰⁴³ property.
- To retrieve commands organized by the toolbar in which they appear, use the [Toolbars](#)¹⁰⁴⁴ property.
- To send commands to StyleVision, use the [Exec](#)¹⁰⁴⁵ method.
- To query if a command is currently enabled or disabled, use the [QueryStatus](#)¹⁰⁴⁶ method.

This enables you to flexibly integrate StyleVision commands into your application's menus and toolbars.

Your installation of StyleVision also provides you with command label images used within StyleVision. See the folder <ApplicationFolder>\Examples\ActiveXImages of your StyleVision installation for icons in GIF format. The file names correspond to the command names as they are listed in the [Command Reference](#)¹⁰²⁴ section.

General considerations

To automate the behaviour of StyleVision, use the properties, methods, and events described for the [StyleVisionControl](#)¹⁰⁴¹, [StyleVisionControl Document](#)¹⁰⁴⁸, and [StyleVisionControl Placeholder](#)¹⁰⁵⁵.

For more complex access to StyleVision functionality, consider using the following properties:

- [StyleVisionControl.Application](#)¹⁰⁴²
- [StyleVisionControlDocument.Document](#)¹⁰⁵⁰
- [StyleVisionControlPlaceholder.Project](#)¹⁰⁵⁶

These properties give you access to the StyleVision automation interface (StyleVisionAPI)

Note: To open a document, always use [StyleVisionControlDocument.Open](#)¹⁰⁵² or [StyleVisionControlDocument.New](#)¹⁰⁵¹ on the appropriate document control. To open a project, always use [StyleVisionControlPlaceholder.OpenProject](#)¹⁰⁵⁶ on a placeholder control embedding a StyleVision project window.

For examples that show how to instantiate and access the necessary controls in different programming environments, see [ActiveX Integration Examples](#)¹⁰¹⁰.

17.3.5 ActiveX Integration Examples

This section contains examples of StyleVision document-level integration using different container environments and programming languages. Source code for all examples is available in the folder `<ApplicationFolder>\Examples\ActiveX` of your StyleVision installation.

17.3.5.1 C#

A basic ActiveX integration example solution for C# and Visual Studio is available in the folder `<ApplicationFolder>\Examples\ActiveX\C#`. Before you compile the source code and run the sample, make sure that all prerequisites are met (see [Running the Sample C# Solution](#)¹⁰¹⁰).

17.3.5.1.1 Running the Sample C# Solution

The sample Visual Studio solution available in the folder `<ApplicationFolder>\Examples\ActiveX\C#` illustrates how to consume the StyleVision ActiveX controls. Before attempting to build and run this solution, note the following steps:

Step 1: Check the prerequisites

Visual Studio 2010 or later is required to open the sample solution. For the complete list of prerequisites, see [Prerequisites](#)¹⁰⁰².

Step 2: Copy the sample to a directory where you have write permissions

To avoid running Visual Studio as an Administrator, copy the source code to a directory where you have write permissions, instead of running it from the default location.

Step 3: Check and set all required control properties

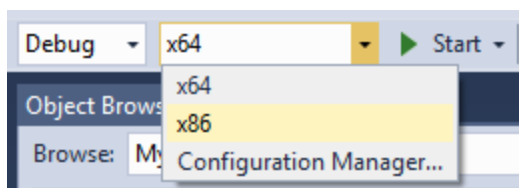
The sample application contains one instance of [StyleVisionControlDocument](#)¹⁰⁴⁸ and several instances of [StyleVisionControlPlaceholder](#)¹⁰⁵⁵ controls. Double-check that the following properties of these controls are set as shown in the table below:

Control name	Property	Property value
axStyleVisionControl	IntegrationLevel	ICActiveXIntegrationOnDocumentLevel
axStyleVisionControlHelperWndSchemaSource	PlaceholderWindowID	2
axStyleVisionControlHelperWndStyleRepository	PlaceholderWindowID	4
axStyleVisionControlHelperWndContextStyle	PlaceholderWindowID	6
axStyleVisionControlHelperWndContextProperties	PlaceholderWindowID	5
axStyleVisionControlHelperWndProject	PlaceholderWindowID	0
axStyleVisionControlHelperWndDesignOverview	PlaceholderWindowID	1
axStyleVisionControlHelperWndDesignTree	PlaceholderWindowID	3

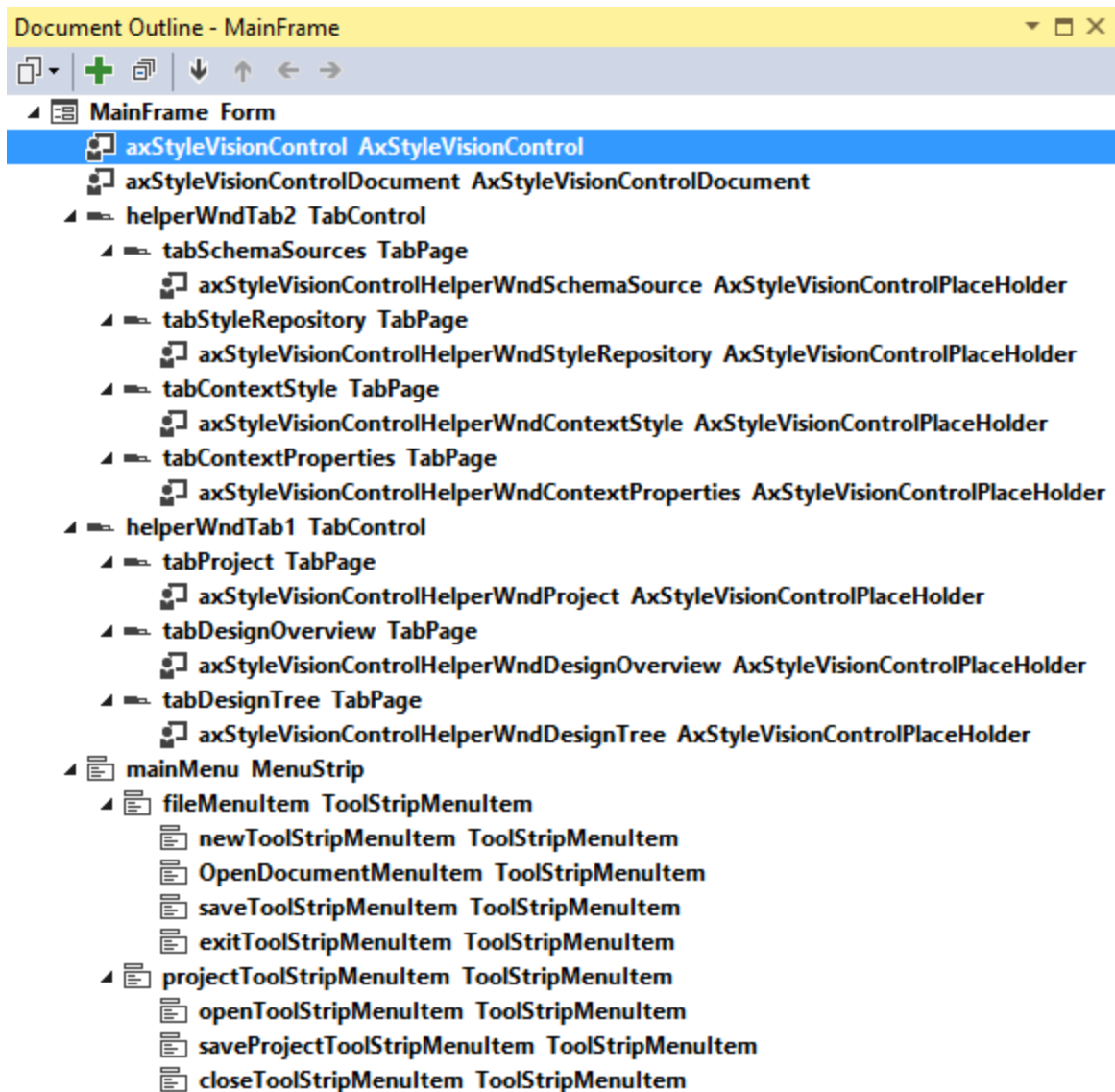
Here is how you can view or set the properties of an ActiveX control:

1. Open the **MDIMain.cs** form in the designer window.

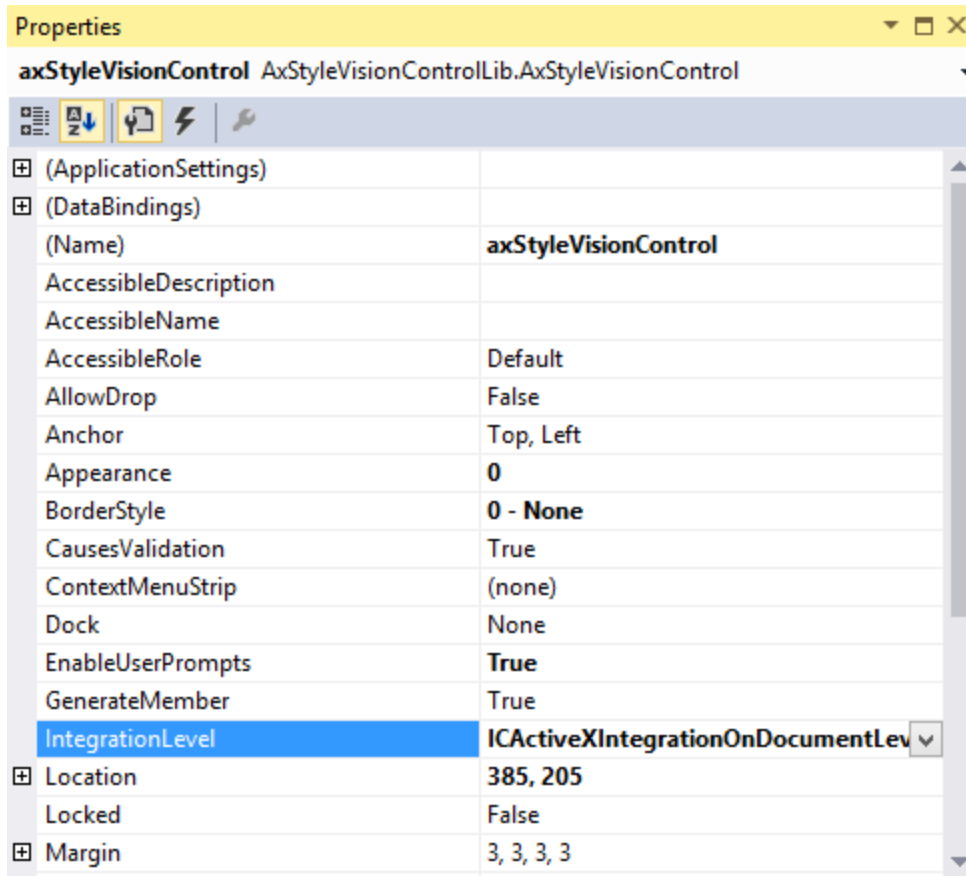
Note: On 64-bit Windows, it may be necessary to change the build configuration of the Visual Studio solution to "x86" **before** opening the designer window. If you need to build the sample as a 64-bit application, see [Prerequisites](#)¹⁰⁰².



2. Open the **Document Outline** window of Visual Studio (On the **View** menu, click **Other Windows | Document Outline**).

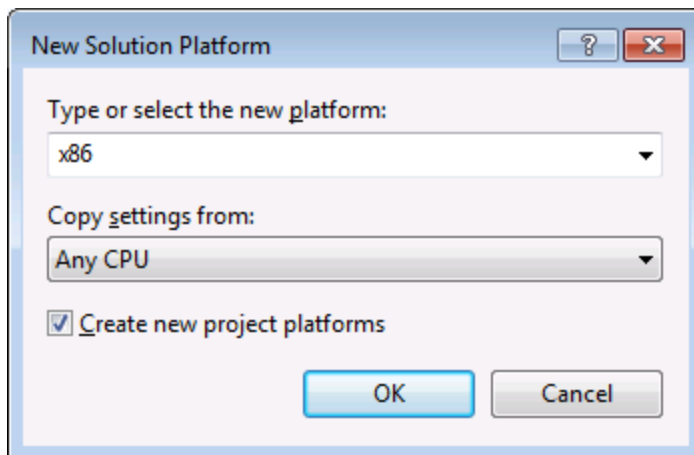


3. Click an ActiveX control in the **Document Outline** window, and edit its required property in the **Properties** window, for example:



Step 4: Set the build platform

- Create a build platform configuration that matches the platform under which you want to build (x86, x64). Here is how you can create the build configuration:
 - a. Right-click the solution in Visual Studio, and select **Configuration Manager**.
 - b. Under **Active solution platform**, select **New...** and then select the x86 or x64 configuration (in this example, **x86**).



You are now ready to build and run the solution in Visual Studio. Remember to build using the configuration that matches your target platform (x86, x64); otherwise, runtime errors might occur.

17.3.5.2 Java

StyleVision ActiveX components can be accessed from Java code. Java integration is provided by the libraries listed below. These libraries are available in the folder `<ApplicationFolder>\Examples\JavaAPI` of your StyleVision installation, after you have installed both StyleVision and the StyleVision Integration Package (see also [Prerequisites](#)¹⁰⁰²).

- `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 included in the Java class search path.

Example Java project

An example Java project is supplied with your product installation. You can test the Java project and modify and use it as you like. For more details, see [Example Java Project](#)¹⁰¹⁵.

Rules for mapping the ActiveX Control names to Java

For the documentation of ActiveX controls, see [Object Reference](#)¹⁰³⁷. Note that the object naming conventions are slightly different in Java compared to other languages. Namely, the rules for mapping between the ActiveX controls and the Java wrapper are as follows:

Classes and class names

For every component of the 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:

Interface	Java name
StyleVisionControlDocument, method New	newDocument
AuthenticView, method Goto	gotoElement
AuthenticRange, method Goto	gotoElement
AuthenticRange, method Clone	cloneRange

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](#) ¹⁰¹⁵
- [Creating the ActiveX Controls](#) ¹⁰¹⁷
- [Loading Data in the Controls](#) ¹⁰¹⁸
- [Basic Event Handling](#) ¹⁰¹⁸
- [Menus](#) ¹⁰¹⁹
- [UI Update Event Handling](#) ¹⁰²¹
- [Creating a StyleVision Node Tree](#) ¹⁰²²

17.3.5.2.1 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:

.classpath	Eclipse project helper file
------------	-----------------------------

<code>.project</code>	Eclipse project file
<code>AltovaAutomation.dll</code>	Java-COM bridge: DLL part (for the 32-bit installation)
<code>AltovaAutomation_x64.dll</code>	Java-COM bridge: DLL part (for the 64-bit installation)
<code>AltovaAutomation.jar</code>	Java-COM bridge: Java library part
<code>BuildAndRun.bat</code>	Batch file to compile and run example code from the command line prompt. Expects folder where Java Virtual Machine resides as parameter.
<code>StyleVisionActiveX.jar</code>	Java classes of the StyleVision ActiveX control
<code>StyleVisionActiveX_JavaDoc.zip</code>	Javadoc file containing help documentation for the Java API
<code>StyleVisionContainer.java</code>	Java example source code
<code>StyleVisionContainerEventHandler.java</code>	Java example source code
<code>StyleVisionDialog.java</code>	Java example source code

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](#)¹⁰¹⁷: Starts StyleVision, which is registered as an automation server, or activates StyleVision if it is already running.
- [Loading Data in the Controls](#)¹⁰¹⁸: Locates one of the example documents installed with StyleVision and opens it.
- [Basic Event Handling](#)¹⁰¹⁹: Changes the view of all open documents to Text View. The code also shows how to iterate through open documents.
- [Menus](#)¹⁰¹⁹: Validates the active document and shows the result in a message box. The code shows how to use output parameters.
- [UI Update Event Handling](#)¹⁰²¹: Shows how to handle StyleVision events.
- [Creating a StyleVision Node Tree](#)¹⁰²²: Shows how to create a StyleVision node tree and prepare it for modal activation.

Updating the path to the Examples folder

Before running the provided sample, you may need to edit the **StyleVisionContainer.java** file. Namely, check that the following path refers to the actual folder where the StyleVision example files are stored on your operating system:

```
// Locate samples installed with the product.
```



```
final String strExamplesFolder = System.getenv( "USERPROFILE" ) + "\\Documents\\Altova\\  
\\StyleVision2024\\StyleVisionExamples\\";
```

Running the example from the command line

To run the example from the command line:

1. Check that all prerequisites are met (see [Prerequisites](#)¹⁰⁰²).
2. Open a command prompt window, change the current directory to the sample Java project folder, and type:

```
buildAndRun.bat "<Path-to-the-Java-bin-folder>"
```

3. Press **Enter**.

The Java source in `StyleVisionContainer.java` will be compiled and then executed.

Compiling and running the example in Eclipse

To import the sample Java project into Eclipse:

1. Check that all prerequisites are met (see [Prerequisites](#)¹⁰⁰²).
2. On the **File** menu, click **Import**.
3. Select **Existing Projects into Workspace**, and browse for the Eclipse project file located at `<ApplicationFolder>\Examples\ActiveX\Java\`. Since you may not have write-access in this folder, it is recommended to select the **Copy projects into workspace** check box on the Import dialog box.

To run the example application, right-click the project in Package Explorer and select the command **Run as | Java Application**.

Help for Java API classes is available through comments in code as well as the Javadoc view of Eclipse. To enable the Javadoc view in Eclipse, select the menu command **Window | Show View | Javadoc**.

17.3.5.2.2 Creating the ActiveX Controls

The code listing below show how ActiveX controls can be created. The constructors will create the Java wrapper objects. Adding these Canvas-derived objects to a panel or to a frame will trigger the creation of the wrapped ActiveX object.

```
01  /**  
02   * StyleVision manager control - always needed  
03   */  
04  public static StyleVisionControl          styleVisionControl = null;  
05  
06  /**  
07   * StyleVisionDocument editing control  
08   */
```

```

09  public static StyleVisionControlDocument    styleVisionDocument = null;
10
11  /**
12   * Tool windows - StyleVision place-holder controls
13   */
14  private static StyleVisionControlPlaceHolder    styleVisionProjectToolWindow = null;
15  private static StyleVisionControlPlaceHolder    styleVisionDesignToolWindow = null;
16  private static StyleVisionControlPlaceHolder    styleVisionSchemaTreeToolWindow = null;
17
18  // Create the StyleVision ActiveX controls. First should be StyleVisionControl
19  // determining that we want to place document controls and place-holder
20  // controls individually. It gives us full control over the menu, as well.
21  styleVisionControl = new StyleVisionControl(
22      IActiveXIntegrationLevel.IActiveXIntegrationOnDocumentLevel.getValue() );
23
24  styleVisionDocument = new StyleVisionControlDocument();
25  frame.add( styleVisionDocument, BorderLayout.CENTER );
26
27  // Create a project window and open the sample project in it
28  styleVisionProjectToolWindow = new StyleVisionPlaceHolder(
29      StyleVisionControlPlaceHolderWindow.StyleVisionControlProjectWindowToolWnd.getValue
30      ( ) );
31  styleVisionProjectToolWindow.setPreferredSize( new Dimension( 200, 200 ) );

```

17.3.5.2.3 Loading Data in the Controls

The code listing below show how data can be loaded in the ActiveX controls.

```

1  // Locate samples installed with the product.
2  final String strExamplesFolder = System.getenv( "USERPROFILE" ) +
3  "\\Documents\\Altova\\StyleVision2024\\StyleVisionExamples\\";
4  styleVisionProjectToolWindow.openProject( strExamplesFolder + "Examples.svp" );

```

17.3.5.2.4 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.

```

01  // Open the PXF file when button is pressed
02  btnOpenPxf.addActionListener( new ActionListener() {
03      public void actionPerformed(ActionEvent e) {
04          try {
05              styleVisionControl.open( strExamplesFolder + "OrgChart.pxf" );
06          } catch (AutomationException e1) {
07              e1.printStackTrace();

```

```

08     }
09     }
10     } );
11     public void onOpenedOrFocused( String i_strFileName, boolean
i_bOpenWithThisControl, boolean i_bFileAlreadyOpened ) throws AutomationException
12     {
13         // Handle the New/Open events coming from the Project tree or from the menus
14         if ( !i_bFileAlreadyOpened )
15         {
16             // This is basically an SDI interface, so open the file in the already existing
document control
17             try {
18                 StyleVisionContainer.initStyleVisionDocument();
19                 StyleVisionContainer.styleVisionDocument.open( i_strFileName );
20             } catch (Exception e) {
21                 e.printStackTrace();
22             }
23         }
24     }

```

17.3.5.2.5 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](#)¹⁰²¹).

```

01
02     // Load the file menu when the button is pressed
03     btnMenu.addActionListener( new ActionListener() {
04         public void actionPerformed(ActionEvent e) {
05             try {
06                 // Create the menubar that will be attached to the frame
07                 MenuBar mb = new MenuBar();
08                 // Load the main menu's first item - the File menu
09                 StyleVisionCommand xmlSpyMenu =
styleVisionControl.getMainMenu().getSubCommands().getItem( 0 );
10                 // Create Java menu items from the Commands objects
11                 Menu fileMenu = new Menu();
12                 handlerObject.fillMenu( fileMenu, xmlSpyMenu.getSubCommands() );
13                 fileMenu.setLabel( xmlSpyMenu.getLabel().replace( "&", "" ) );
14                 mb.add( fileMenu );
15                 frame.setMenuBar( mb );
16                 frame.validate();
17             } catch (AutomationException e1) {
18                 e1.printStackTrace();
19             }
20             // Disable the button when the action has been performed
21             ((AbstractButton) e.getSource()).setEnabled( false );
22         }

```

```

23     } ) ;
24     /**
25     * Populates a menu with the commands and submenus contained in a StyleVisionCommands
object
26     */
27     public void fillMenu(Menu newMenu, StyleVisionCommands styleVisionMenu) throws
AutomationException
28     {
29         // For each command/submenu in the xmlSpyMenu
30         for ( int i = 0 ; i < styleVisionMenu.getCount() ; ++i )
31         {
32             StyleVisionCommand styleVisionCommand = styleVisionMenu.getItem( i );
33             if ( styleVisionCommand.getIsSeparator() )
34                 newMenu.addSeparator();
35             else
36             {
37                 StyleVisionCommands subCommands = styleVisionCommand.getSubCommands();
38                 // Is it a command (leaf), or a submenu?
39                 if ( subCommands.isNull() || subCommands.getCount() == 0 )
40                 {
41                     // Command -> add it to the menu, set its ActionCommand to its ID and store it
in the menuMap
42                     MenuItem mi = new MenuItem( styleVisionCommand.getLabel().replace( "&",
"" ) );
43                     mi.setActionCommand( "" + styleVisionCommand.getID() );
44                     mi.addActionListener( this );
45                     newMenu.add( mi );
46                     menuMap.put( styleVisionCommand.getID(), mi );
47                 }
48                 else
49                 {
50                     // Submenu -> create submenu and repeat recursively
51                     Menu newSubMenu = new Menu();
52                     fillMenu( newSubMenu, subCommands );
53                     newSubMenu.setLabel( styleVisionCommand.getLabel().replace( "&", "" ) );
54                     newMenu.add( newSubMenu );
55                 }
56             }
57         }
58     }
59     /**
60     * Action handler for the menu items
61     * Called when the user selects a menu item; the item's action command corresponds to
the command table for XMLSpy
62     */
63     public void actionPerformed( ActionEvent e )
64     {
65         try
66         {
67             int iCmd = Integer.parseInt( e.getActionCommand() );
68             // Handle explicitly the Close commands
69             switch ( iCmd )
70             {
71                 case 57602:         // Close
72                 case 34050:        // Close All
73                     StyleVisionContainer.initStyleVisionDocument();

```

```
74         break;
75     default:
76         StyleVisionContainer.styleVisionControl.exec( iCmd );
77         break;
78     }
79 }
80 catch ( Exception ex )
81 {
82     ex.printStackTrace();
83 }
84
85 }
```

17.3.5.2.6 UI Update Event Handling

The code listing below shows how a UI-Update event handler can be created.

```
01  /**
02   * Call-back from the StyleVisionControl.
03   * Called to enable/disable commands
04   */
05  @Override
06  public void onUpdateCmdUI() throws AutomationException
07  {
08      // A command should be enabled if the result of queryStatus contains the Supported
(1) and Enabled (2) flags
09      for ( java.util.Map.Entry<Integer, MenuItem> pair : menuMap.entrySet() )
10
pair.getValue().setEnabled( StyleVisionContainer.styleVisionControl.queryStatus( pair.getKe
y() ) > 2 );
11  }
12
13  /**
14   * Call-back from the StyleVisionControl.
15   * Usually called while enabling/disabling commands due to UI updates
16   */
17  @Override
18  public boolean onIsActiveEditor( String i_strFilePath ) throws AutomationException
19  {
20      try {
21          return
StyleVisionContainer.styleVisionDocument.getDocument().getFullName().equalsIgnoreCase( i_st
rFilePath );
22      } catch ( Exception e ) {
23          return false;
24      }
25  }
```

17.3.5.2.7 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.

```
001 //access StyleVision Java-COM bridge
002 import com.altova.automation.StyleVision.*;
003
004 // access AWT/Swing components
005 import java.awt.*;
006 import javax.swing.*;
007 import javax.swing.tree.*;
008
009 /**
010  * A simple example of a tree control loading the structure from a StyleVision Document
011  * object.
012  * The class receives a Document object, loads its nodes in a JTree, and prepares
013  * for modal activation.
014  *
015  * Feel free to modify and extend this sample.
016  *
017  * @author Altova GmbH
018  */
019 class StyleVisionDialog extends JDialog
020 {
021     /**
022      * The tree control
023      */
024     private JTree myTree;
025
026     /**
027      * Root node of the tree control
028      */
029     private DefaultMutableTreeNode top ;
030
031     /**
032      * Constructor that prepares the modal dialog containing the filled tree control
033      * @param xml The data to be displayed in the tree
034      * @param parent Parent frame
035      */
036     public StyleVisionDialog( Document doc, Frame parent )
037     {
038         // Construct the modal dialog
039         super( parent, "Data tree", true );
040         // Arrange controls in the dialog
041         top = new DefaultMutableTreeNode("root");
042         myTree = new JTree(top);
043         setContentPane( new JScrollPane( myTree ) );
044         // Build up the tree; expand all nodes, hide root node
045         fillTree( top, doc );
046         for(int i = 0 ; i < myTree.getRowCount() ; ++i )
047             myTree.expandRow( i );
048         myTree.setRootVisible( false );
049     }
050 }
```

```
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
DefaultMutableTreeNode( "SchemaSources" );
060         for (SchemaSource schema : doc.getSchemaSources())
061         {
062             String nodeText = "$" + schema.getName() ;
063             if ( schema.getIsMainSchemaSource() )
064                 nodeText += " (main)";
065             DefaultMutableTreeNode newNode = new DefaultMutableTreeNode( nodeText ) ;
066             node.add( titleNode ) ;
067             titleNode.add( newNode ) ;
068             String attribute = schema.getSchemaFileName() ;
069             if ( attribute.length() > 0 )
070             {
071                 DefaultMutableTreeNode subNode = new DefaultMutableTreeNode( "SchemaFile" ) ;
072                 subNode.add( new DefaultMutableTreeNode( attribute ) );
073                 newNode.add( subNode );
074             }
075             attribute = schema.getWorkingXMLFileName() ;
076             if ( attribute.length() > 0 )
077             {
078                 DefaultMutableTreeNode subNode = new DefaultMutableTreeNode( "Working
XML" ) ;
079                 subNode.add( new DefaultMutableTreeNode( attribute ) );
080                 newNode.add( subNode );
081             }
082             attribute = schema.getTemplateFileName() ;
083             if ( attribute.length() > 0 )
084             {
085                 DefaultMutableTreeNode subNode = new DefaultMutableTreeNode( "Template" ) ;
086                 subNode.add( new DefaultMutableTreeNode( attribute ) );
087                 newNode.add( subNode );
088             }
089         }
090
091         titleNode = new DefaultMutableTreeNode( "Parameters" ) ;
092         for (Parameter param : doc.getParameters())
093         {
094             DefaultMutableTreeNode newNode = new DefaultMutableTreeNode( "$" +
param.getName() );
095             node.add( titleNode ) ;
096             titleNode.add( newNode ) ;
097             DefaultMutableTreeNode subNode = new DefaultMutableTreeNode( "Value" ) ;
098             subNode.add( new DefaultMutableTreeNode( param.getValue() ) );
099             newNode.add( subNode );
100         }
}
```

```

101     }
102     catch (Exception e)
103     {
104         e.printStackTrace();
105     }
106 }
107
108 }

```

17.3.6 Command Reference

This section lists the names and identifiers of all menu commands that are available within StyleVision. Every sub-section lists the commands from the corresponding top-level menu of StyleVision. The command tables are organized as follows:

- The "Menu Item" column shows the command's menu text as it appears in StyleVision, to make it easier for you to identify the functionality behind the command.
- The "Command Name" column specifies the string that can be used to get an icon with the same name from **ActiveXImages** folder of the StyleVision installation directory.
- The "ID" column shows the numeric identifier of the column that must be supplied as argument to methods which execute or query this command.

To execute a command, use the [StyleVisionControl.Exec](#)¹⁰⁴⁵ or the [StyleVisionControlDocument.Exec](#)¹⁰⁵¹ methods. To query the status of a command, use the [StyleVisionControl.QueryStatus](#)¹⁰⁴⁶ or [StyleVisionControlDocument.QueryStatus](#)¹⁰⁵² methods.

Depending on the edition of StyleVision you have installed, some of these commands might not be supported.

17.3.6.1 "File" Menu

The "File" menu has the following commands:

Menu item	Command name	ID
New from XML Schema/DTD/XML...	IDC_FILE_NEW_FROM_SCHEMA	37579
New from DB...	IDC_FILE_NEW_FROM_DB	37577
New from XML column in DB table...	IDC_FILE_NEW_FROM_DBCELL	37861
New from XBRL Taxonomy...	IDC_FILE_NEW_FROM_XBRL_TAXONOMY	37912
New from HTML file...	IDC_FILE_NEW_FROM_HTML	37578
New from XSLT file...	IDC_FILE_NEW_FROM_XSLT	39737
New from Word 2007+ file...	IDC_FILE_NEW_FROM_DOCX	39814
New from Excel 2007+ file...	IDC_FILE_NEW_FROM_EXCEL	36054

Menu item	Command name	ID
New (empty)	IDC_FILE_NEW_EMPTY	37576
Open...	ID_FILE_OPEN	57601
Reload	IDC_FILE_RELOAD	36002
Close	ID_FILE_CLOSE	57602
Close All	ID_FILE_CLOSE_ALL	37777
Save Design	ID_FILE_SAVE	57603
Save As...	ID_FILE_SAVE_AS	57604
Save All	ID_FILE_SAVE_ALL	37778
Export as MobileTogether Design file...	IDC_FILE_EXPORT_AS_MTD	36055
Save Authentic XML Data	ID_SAVE_AUTHENTIC_XML	32860
Save Authentic XML Data as ...	ID_SAVE_AUTHENTIC_XML_AS	32812
Save Generated XSLT-HTML File...	IDC_SAVE_GEN_XSLT_HTML	37642
Save Generated HTML File(s)...	IDC_SAVE_GEN_HTML	37636
Save Generated XSLT-RTF File...	IDC_SAVE_GEN_XSLT_RTF	37643
Save Generated RTF File(s)...	IDC_SAVE_GEN_RTF	37638
Save Generated XSLT-FO File...	IDC_SAVE_GEN_XSLT_FO	37641
Save Generated FO File(s)...	IDC_SAVE_GEN_FO	37635
Save Generated PDF File(s)...	IDC_SAVE_GEN_PDF	37637
Save Generated XSLT-Word 2007+ File...	IDC_SAVE_GEN_XSLT_WORDML	37523
Save Generated Word 2007+ File(s)...	IDC_SAVE_GEN_WORDML	37529
Save Generated DB Schema...	IDC_SAVE_GEN_DB_SCHEMA	37633
Save Generated DB XML Data...	IDC_SAVE_GEN_DB_XML	37634
Save Generated User-Defined Schema...	IDC_SAVE_GEN_USERDEF_SCHEMA	37639
Save Generated User-Defined XML Data...	IDC_SAVE_GEN_USERDEF_XML	37640
Deploy to FlowForce...	IDC_STYLEVISIONGUI_DEPLOY_TO_FLOWFORCE	39787
Generate ASPX Web Application...	IDC_SAVE_SCRIPT_FOR_DBEXTRACT_AND_XSLTGEN ASPX	39755
Properties...	IDC_FILE_PROPERTIES	36027

Menu item	Command name	ID
Print Preview...	ID_FILE_PRINT_PREVIEW	57609
Print...	ID_FILE_PRINT	57607
Recent File	ID_FILE_MRU_FILE1	57616
Exit	ID_APP_EXIT	57665

17.3.6.2 "Edit" Menu

The "Edit" menu has the following commands:

Menu item	Command name	ID
Undo	ID_EDIT_UNDO	57643
Redo	ID_EDIT_REDO	57644
Cut	ID_EDIT_CUT	57635
Copy	ID_EDIT_COPY	57634
Paste	ID_EDIT_PASTE	57637
Delete	ID_EDIT_CLEAR	57632
Find	ID_EDIT_FIND	57636
Find Next	ID_EDIT_FINDNEXT	36802
Replace...	ID_EDIT_REPLACE	57641
Stylesheet Parameters...	IDC_EDIT_PARAMETERS	37573
Collapse/Expand Markup	IDC_SPSGUI_COLLAPSE_EXPAND_MARKUP	36804
Select All	ID_EDIT_SELECT_ALL	57642

17.3.6.3 "Project" Menu

The "Project" menu has the following commands:

Menu item	Command name	ID
New Project	IDC_ICPROJECTGUI_NEW	37200
Open Project...	IDC_ICPROJECTGUI_OPEN	37201

Menu item	Command name	ID
Reload Project	IDC_ICPROJECTGUI_RELOAD	37202
Close Project	IDC_ICPROJECTGUI_CLOSE	37203
Save Project	IDC_ICPROJECTGUI_SAVE	37204
Add Files to Project...	IDC_ICPROJECTGUI_ADD_FILES_TO_PROJECT	37205
Add Global Resource to Project...	IDC_ICPROJECTGUI_ADD_GLOBAL_RESOURCE_TO_PROJECT	37239
Add URL to Project...	IDC_ICPROJECTGUI_ADD_URL_TO_PROJECT	37206
Add Active File to Project	IDC_ICPROJECTGUI_ADD_ACTIVE_FILE_TO_PROJECT	37208
Add Active and Related Files to Project	IDC_ICPROJECTGUI_ADD_ACTIVE_AND_RELATED_FILES_TO_PROJECT	37209
Add Project Folder to Project...	IDC_ICPROJECTGUI_ADD_FOLDER_TO_PROJECT	37210
Add External Folder to Project...	IDC_ICPROJECTGUI_ADD_EXT_FOLDER_TO_PROJECT	37211
Add External Web Folder to Project...	IDC_ICPROJECTGUI_ADD_EXT_URL_FOLDER_TO_PROJECT	37212
Recent Project	IDC_ICPROJECTGUI_RECENT	37224

17.3.6.4 "View" Menu

The "View" menu has the following commands:

Menu item	Command name	ID
Dummy entry	ID_VIEW_TOOLBARS	37807
Project	ID_VIEW_PROJECT	57682
Design Overview	ID_VIEW_DESIGNOVERVIEW	37883
Schema Tree	ID_VIEW_SCHEMASOURCES	37805
Design Tree	ID_VIEW_DESIGNTREE	37804
Style Repository	ID_VIEW_STYLEREPOSITORY	37806

Menu item	Command name	ID
Context Properties	ID_VIEW_CONTEXTPROPERTY_COMMON	37802
Context Styles	ID_VIEW_CONTEXTPROPERTY_STYLES	37803
Show only one template at once	IDC_DESIGN_FILTER_ONE	37704
Show all template types	IDC_DESIGN_FILTER_ALL	37700
Show imported templates	IDC_DESIGN_FILTER_IMPORTED	37870
Show/Hide main template	IDC_DESIGN_FILTER_MAIN	37701
Show/Hide global templates	IDC_DESIGN_FILTER_MATCH	37702
Show/Hide design fragments	IDC_DESIGN_FILTER_NAMED	37703
Zoom In	IDC_SPSGUI_ZOOM_IN	36805
Zoom Out	IDC_SPSGUI_ZOOM_OUT	36806
Zoom 500%	IDC_SPSGUI_ZOOM_500	36807
Zoom 400%	IDC_SPSGUI_ZOOM_400	36808
Zoom 200%	IDC_SPSGUI_ZOOM_200	36809
Zoom 150%	IDC_SPSGUI_ZOOM_150	36810
Zoom 100%	IDC_SPSGUI_ZOOM_100	36811
Zoom 75%	IDC_SPSGUI_ZOOM_75	36812
Zoom 50%	IDC_SPSGUI_ZOOM_50	36813
Status Bar	ID_VIEW_STATUS_BAR	59393

17.3.6.5 "Insert" Menu

The "Insert" menu has the following commands:

Menu item	Command name	ID
Insert Contents	IDC_INSERT_CONTENTS	37846
Insert Rest of Contents	IDC_INSERT_REST_OF_CONTENTS	37617
Insert RichEdit	IDC_INSERT_RICHEDIT	39773
Input Field	IDC_INSERT_EDITFIELD	37854
Multiline Input Field	IDC_INSERT_MULTILINEEDITFIELD	37855

Menu item	Command name	ID
Check Box...	IDC_INSERT_CHECKBOX	37857
Combo Box...	IDC_INSERT_COMBOBOX	37856
Radio Button...	IDC_INSERT_RADIOBUTTON	37858
Button	IDC_INSERT_BUTTON	37859
Navigation	IDC_INSERT_DBNAVIGATION	37603
Navigation + Goto	IDC_INSERT_DBNAVIGATIONGOTO	37604
Query Button	IDC_INSERT_DBQUERY	37605
Value...	IDC_INSERT_AUTOCALC_VALUE	37596
Input Field...	IDC_INSERT_AUTOCALC_FIELD	37594
Multiline Input Field...	IDC_INSERT_AUTOCALC_MULTILINE_FIELD	37595
Insert Date Picker	IDC_INSERT_DATEPICKER	37602
Insert Paragraph	IDC_INSERT_PARAGRAPH	37847
Address	IDC_INSERT_ADDRESS	37843
Block (div)	IDC_INSERT_BLOCK	37528
Blockquote	IDC_INSERT_BLOCKQUOTE	37527
Center	IDC_INSERT_CENTER	37534
Fieldset	IDC_INSERT_FIELDSET	37574
Preformatted	IDC_INSERT_FORMATTED	37580
Preformatted, wrapping	IDC_INSERT_FORMATTED_WRAP	37876
Heading 1 (h1)	IDC_INSERT_HEADING1	37585
Heading 2 (h2)	IDC_INSERT_HEADING2	37586
Heading 3 (h3)	IDC_INSERT_HEADING3	37587
Heading 4 (h4)	IDC_INSERT_HEADING4	37588
Heading 5 (h5)	IDC_INSERT_HEADING5	37589
Heading 6 (h6)	IDC_INSERT_HEADING6	37590
Insert Barcode...	IDC_INSERT_BARCODE	40221
Insert Chart...	IDC_INSERT_CHART	40223
Insert Image...	IDC_INSERT_IMAGE	37593

Menu item	Command name	ID
Insert Horizontal Line	IDC_INSERT_HORIZONTAL_LINE	37606
Insert Table...	IDC_INSERT_TABLE	40212
Insert Bullets and Numbering...	IDC_INSERT_FORMAT_BULLETS	37848
Insert Bookmark...	IDC_INSERT_BOOKMARK	37530
Insert Hyperlink...	IDC_INSERT_HYPERLINK	37849
Insert Footnote	IDC_INSERT_FOOTNOTE	39797
Insert Condition...	IDC_INSERT_CONDITION	37850
Insert Output-based Condition	IDC_INSERT_CONDITION_PER_OUTPUT	37851
Insert Template...	IDC_INSERT_TEMPLATE	40216
Insert User-Defined Template...	IDC_INSERT_USER_DEFINED_TEMPLATE	40184
Insert Variable Template...	IDC_INSERT_VARIABLE_TEMPLATE	37531
Dummy entry	IDC_INSERT_NAMED_TEMPLATE	39300
Insert Layout Container	IDC_INSERT_LAYOUT_CONTAINER	40213
Insert Layout Box	IDC_INSERT_LAYOUT_BOX	40214
Insert Line	IDC_INSERT_SHAPE_LINE	40215
Table of Contents...	IDC_INSERT_TOC_WIZARD	37618
TOC Bookmark	IDC_INSERT_MARKER	37608
TOC Bookmark (Wizard)...	IDC_INSERT_MARKER_WIZARD	37609
TOC Reference	IDC_INSERT_REF	37613
Entry Text	IDC_INSERT_REF_ENTRY_TEXT	37614
Leader	IDC_INSERT_REF_LEADER	37615
Page Reference	IDC_INSERT_REF_PAGE_REFERENCE	37616
Hierarchical Numbering	IDC_INSERT_AUTO_NUMBER_LEVEL	37598
Sequential Numbering	IDC_INSERT_AUTO_NUMBER_FLAT	37597
TOC Level	IDC_INSERT_LEVEL	37607
TOC Level Reference	IDC_INSERT_REFLEVEL	37860
Template Serves as Level	IDC_ASSIGN_LEVEL	37524
Insert New Document	IDC_INSERT_RESULT_DOCUMENT	39756

Menu item	Command name	ID
New Page	IDC_INSERT_PAGE_BREAK	37610
Page Number	IDC_INSERT_PAGE_NUMBER	37611
Page Total	IDC_INSERT_PAGE_TOTAL	37612
New Column	IDC_INSERT_COLUMN_BREAK	37864
New Document Section	IDC_INSERT_DOCUMENT_SECTION	37951
Insert XBRL Table from Taxonomy...	IDC_INSERT_XBRL_TABLE_FROM_TAXONOMY	39813
Context	IDC_INSERT_XBRL_CONTEXT	37942
Item	IDC_INSERT_XBRL_ITEM	39793
Period	IDC_INSERT_XBRL_PERIOD	37909
Period Start	IDC_INSERT_XBRL_PERIOD_START	37947
Period End	IDC_INSERT_XBRL_PERIOD_END	37948
Identifier	IDC_INSERT_XBRL_IDENTIFIER	37910
Unit	IDC_INSERT_XBRL_UNITASPECT	39789
UnitRef	IDC_INSERT_XBRL_UNITREF	37946
Footnote	IDC_INSERT_XBRL_FOOTNOTE	37911
Non-XDT Segment	IDC_INSERT_XBRL_NONXDTSEGMENT_ASPECT	39790
Non-XDT Scenario	IDC_INSERT_XBRL_NONXDTSCENARIO_ASPECT	39791
XML Node	IDC_INSERT_XBRL_XMLNODE	39792
iXBRL Hidden Data	IDC_INSERT_IXBRL_HIDE	39721
iXBRL Fraction	IDC_INSERT_IXBRL_FRACTION	39723
User-Defined Element	IDC_STYLEVISIONGUI_INSERT_USERXMLELEMENT	37908
User-Defined Block	IDC_STYLEVISIONGUI_INSERT_USERXMLTEXT	37920

17.3.6.6 "Enclose with" Menu

The "Enclose with" menu has the following commands:

Menu item	Command name	ID
Template...	IDC_ENCLOSE_WITH_TEMPLATES	40180
User-Defined Template...	IDC_ENCLOSE_WITH_USER_DEFINED_TEMPLATES	40217
Variable Template...	IDC_ENCLOSE_WITH_VARIABLE_TEMPLATES	37520
Paragraph	IDC_ENCLOSE_PARAGRAPH	37627
Address	IDC_ENCLOSE_ADDRESS	40194
Block (div)	IDC_ENCLOSE_BLOCK	40195
Blockquote	IDC_ENCLOSE_BLOCKQUOTE	40196
Center	IDC_ENCLOSE_CENTER	40197
Fieldset	IDC_ENCLOSE_FIELDSET	40198
Preformatted	IDC_ENCLOSE_FORMATTED	40199
Preformatted, wrapping	IDC_ENCLOSE_FORMATTED_WRAP	40200
Heading 1 (h1)	IDC_ENCLOSE_HEADING1	40201
Heading 2 (h2)	IDC_ENCLOSE_HEADING2	40202
Heading 3 (h3)	IDC_ENCLOSE_HEADING3	40203
Heading 4 (h4)	IDC_ENCLOSE_HEADING4	40204
Heading 5 (h5)	IDC_ENCLOSE_HEADING5	40205
Heading 6 (h6)	IDC_ENCLOSE_HEADING6	40206
Bullets and Numbering...	IDC_ENCLOSE_FORMAT_BULLETS	37581
Bookmark...	IDC_ENCLOSE_BOOKMARK	40207
Hyperlink...	IDC_ENCLOSE_HYPERLINK	37620
Condition...	IDC_ENCLOSE_CONDITION	37599
Output-based Condition	IDC_ENCLOSE_CONDITION_PER_OUTPUT	37600
TOC Bookmark	IDC_ENCLOSE_MARKER	40210
TOC Bookmark (Wizard)...	IDC_ENCLOSE_MARKER_WIZARD	40211

Menu item	Command name	ID
TOC Level	IDC_ENCLOSE_LEVEL	40208
TOC Level Reference	IDC_ENCLOSE_REFLEVEL	40209
New Document	IDC_ENCLOSE_RESULT_DOCUMENT	39758
User-Defined Element...	IDC_ENCLOSE_WITH_USER_DEFINED_XML_ELEMENT	40222

17.3.6.7 "Table" Menu

The "Table" menu has the following commands:

Menu item	Command name	ID
Insert Table...	IDC_INSERT_TABLE	40212
Delete Table	IDC_TABLE_DELETE	37658
Add Table Header Column	IDC_TABLE_ADD_HEADERCOL	37888
Add Table Footer Column	IDC_TABLE_ADD_FOOTERCOL	37889
Add Table Header Row	IDC_TABLE_ADD_HEADERROW	37900
Add Table Footer Row	IDC_TABLE_ADD_FOOTERROW	37901
Append Row	IDC_TABLE_APPEND_ROW	37657
Append Column	IDC_TABLE_APPEND_COL	37656
Insert Row	IDC_TABLE_INSERT_ROW	37664
Insert Column	IDC_TABLE_INSERT_COL	37662
Delete Row	IDC_TABLE_DELETE_ROW	37660
Delete Column	IDC_TABLE_DELETE_COL	37659
Join Cell Left	IDC_TABLE_JOIN_LEFT	37666
Join Cell Right	IDC_TABLE_JOIN_RIGHT	37667
Join Cell Below	IDC_TABLE_JOIN_DOWN	37665
Join Cell Above	IDC_TABLE_JOIN_UP	37668
Split Cell Horizontally	IDC_TABLE_SPLIT_HORZ	37670
Split Cell Vertically	IDC_TABLE_SPLIT_VERT	37671

Menu item	Command name	ID
View Cell Bounds	IDC_TABLE_SHOW_ZERO_BORDER	37669
View Table Markup	IDC_TABLE_SHOW_UICELLS	37887
Table Properties...	IDC_TABLE_EDIT_PROPERTIES	37661
Edit CALS/HTML Tables...	IDC_EDIT_XMLTABLES	37533

17.3.6.8 "Authentic" Menu

The "Authentic" menu has the following commands:

Menu item	Command name	ID
Edit Authentic Scripts...	IDC_EDIT_AUTHENTIC_SCRIPTPROJECT	37601
Custom Toolbar Buttons...	IDC_TEXT_STATE_ICONS	37672
Check Macro References...	IDC_AUTHENTIC_CHECK_MACRO_REFERENCES	39751
Auto-add Date Picker	IDC_TOGGLE_DATEPICKER_AUTOINSERT	37674
Auto-add DB Controls	IDC_TOGGLE_DBCONTROL_AUTOINSERT	37675
Reload Authentic View	IDC_AUTHENTICGUI_RELOAD	32800
Validate XML	IDC_VALIDATE	32954
Select New Row with XML Data for Editing...	IDC_CHANGE_WORKING_DB_XML_CELL	32861
XML Signature...	IDC_AUTHENTICGUI_XMLSIGNATURE	32862
Define XML Entities...	IDC_DEFINE_ENTITIES	32805
Hide Markup	IDC_MARKUP_HIDE	32855
Show Small Markup	IDC_MARKUP_SMALL	32858
Show Large Markup	IDC_MARKUP_LARGE	32856
Show Mixed Markup	IDC_MARKUP_MIXED	32857
Toggle Bold	IDC_AUTHENTICGUI_RICHEDIT_TOGGLEBOLD	32813
Toggle Italic	IDC_AUTHENTICGUI_RICHEDIT_TOGGLEITALIC	32814

Menu item	Command name	ID
Toggle Underline	IDC_AUTHENTICGUI_RICHEDIT_TOGGLEUNDERLINE	32815
Toggle Strikethrough	IDC_AUTHENTICGUI_RICHEDIT_TOGGLESTRIKETHROUGH	32816
Foreground Color	IDC_AUTHENTICGUI_RICHEDIT_COLOR_FOREGROUND	32824
Background Color	IDC_AUTHENTICGUI_RICHEDIT_COLOR_BACKGROUND	32830
Align Left	IDC_AUTHENTICGUI_RICHEDIT_ALIGN_LEFT	32818
Center	IDC_AUTHENTICGUI_RICHEDIT_ALIGN_CENTER	32819
Align Right	IDC_AUTHENTICGUI_RICHEDIT_ALIGN_RIGHT	32820
Justify	IDC_AUTHENTICGUI_RICHEDIT_ALIGN_JUSTIFY	32821
Append Row	IDC_ROW_APPEND	32806
Insert Row	IDC_ROW_INSERT	32809
Duplicate Row	IDC_ROW_DUPLICATE	32808
Move Row Up	IDC_ROW_MOVE_UP	32811
Move Row Down	IDC_ROW_MOVE_DOWN	32810
Delete Row	IDC_ROW_DELETE	32807

17.3.6.9 "Database" Menu

The "Database" menu has the following commands:

Menu item	Command name	ID
Query Database	ID_VIEW_DBQUERY	37818
Edit DB Filter...	IDC_EDIT_DBFILTERS	37571
Clear DB Filter	IDC_CLEAR_DBFILTERS	37547

17.3.6.10 "Properties" Menu

The "Properties" menu has the following commands:

Menu item	Command name	ID
Edit Bullets and Numbering...	IDC_EDIT_BULLETS_AND_NUMBERING	37839
Predefined Value Formatting Strings...	IDC_FORMAT_PREDEFINES	37582

17.3.6.11 "Tools" Menu

The "Tools" menu has the following commands:

Menu item	Command name	ID
Spelling...	IDC_SPSGUI_SPELL_CHECK	36800
Spelling Options...	IDC_SPSGUI_SPELL_OPTIONS	36801
Global Resources	IDC_GLOBALRESOURCES	37401
<plugin not loaded>	IDC_GLOBALRESOURCES_SUBMENUENTR Y1	37408
Customize...	IDC_APP_TOOLS_CUSTOMIZE	32959
Options...	IDC_TOOLS_OPTIONS	37676

17.3.6.12 "Window" Menu

The "Window" menu has the following commands:

Menu item	Command name	ID
Cascade	ID_WINDOW_CASCADE	57650
Tile Horizontal	ID_WINDOW_TILE_HORZ	57651
Tile Vertical	ID_WINDOW_TILE_VERT	57652
Arrange Icons	ID_WINDOW_ARRANGE	57649

17.3.6.13 "Help" Menu

The "Help" menu has the following commands:

Menu item	Command name	ID
Table of Contents	IDC_HELP_CONTENTS	32966
Index...	IDC_HELP_INDEX	32967
Search...	IDC_HELP_SEARCH	32969
Software Activation...	IDC_ACTIVATION	32970
Order form...	IDC_OPEN_ORDER_PAGE	32971
Registration...	IDC_REGISTRATION	32972
Check for Updates...	IDC_CHECK_FOR_UPDATES	32973
StyleVision Product Comparison...	IDC_PRODUCT_COMPARISON	32955
Support Center...	IDC_OPEN_SUPPORT_PAGE	32961
FAQ on the Web...	IDC_OPEN_FAQ_PAGE	32962
Download Components and Free Tools...	IDC_OPEN_COMPONENTS_PAGE	32963
StyleVision on the Internet...	IDC_OPEN_HOME_PAGE	32964
StyleVision Training...	IDC_OPEN_TRAINING_PAGE	32965
About StyleVision...	ID_APP_ABOUT	57664

17.3.7 Object Reference

Objects:

[StyleVisionCommand](#) ¹⁰³⁸

[StyleVisionCommands](#) ¹⁰⁴⁰

[StyleVisionControl](#) ¹⁰⁴¹

[StyleVisionControlDocument](#) ¹⁰⁴⁸

[StyleVisionControlPlaceHolder](#) ¹⁰⁵⁵

To give access to standard StyleVision functionality, objects of the **StyleVision automation interface** can be accessed as well. See [StyleVisionControl.Application](#) ¹⁰⁴², [StyleVisionControlDocument.Document](#) ¹⁰⁵⁰ and [StyleVisionControlPlaceHolder.Project](#) ¹⁰⁵⁶ for more information.

17.3.7.1 StyleVisionCommand

Properties:

[ID](#)¹⁰³⁸
[Label](#)¹⁰³⁹
[Name](#)¹⁰³⁹
[IsSeparator](#)¹⁰³⁹
[ToolTip](#)¹⁰⁴⁰
[StatusText](#)¹⁰³⁹
[Accelerator](#)¹⁰³⁸
[SubCommands](#)¹⁰⁴⁰

Description:

A command object can be one of the following: an executable command, a command container (for example, a menu, submenu, or toolbar), or a menu separator. To determine what kind of information is stored in the current Command object, query its `ID`, `IsSeparator`, and `SubCommands` properties, as follows.

The Command object is...	When...
An executable command	<ul style="list-style-type: none"> • <code>ID</code> is greater than zero • <code>IsSeparator</code> is false • <code>SubCommands</code> is empty
A command container	<ul style="list-style-type: none"> • <code>ID</code> is zero • <code>IsSeparator</code> is false • <code>SubCommands</code> contains a collection of Command objects.
Separator	<ul style="list-style-type: none"> • <code>ID</code> is zero • <code>IsSeparator</code> is true

17.3.7.1.1 Accelerator

Property: `Accelerator` as `string`

Description:

Returns the accelerator key defined for the command. If the command has no accelerator key assigned, this property returns the empty string. The string representation of the accelerator key has the following format:

[ALT+] [CTRL+] [SHIFT+] key

Where `key` is converted using the Windows Platform SDK function `GetKeyNameText`.

17.3.7.1.2 ID

Property: `ID` as `long`

Description:

This property gets the unique identifier of the command. A command's ID is required to execute the command (using [Exec](#)¹⁰⁴⁵) or query its status (using [QueryStatus](#)¹⁰⁴⁶). If the command is a container for other commands (for example, a top-level menu), or a separator, the ID is 0.

17.3.7.1.3 IsSeparator

Property: IsSeparator as `boolean`

Description:

The property returns `true` if the command object is a menu separator; `false` otherwise. See also [Command](#)¹⁰³⁸.

17.3.7.1.4 Label

Property: Label as `string`

Description:

This property gets the text of the command as it is displayed in the graphical user interface of StyleVision. If the command is a separator, "Label" is an empty string. This property may also return an empty string for some toolbar commands that do not have any GUI text associated with them.

17.3.7.1.5 Name

Property: Name as `string`

Description:

This property gets the unique name of the command. This value can be used to get the icon file of the command, where it is available. The available icon files can be found in the folder `<ApplicationFolder>\Examples\ActiveX\Images` of your StyleVision installation.

17.3.7.1.6 StatusText

Property: Label as `string`

Description:

The status text is the text shown in the status bar of StyleVision when the command is selected. It applies only to command objects that are not separators or containers of other commands; otherwise, the property is an empty string.

17.3.7.1.7 SubCommands

Property: SubCommands as [Commands](#)¹⁰⁴⁰

Description:

The SubCommands property gets the collection of [Command](#)¹⁰³⁸ objects that are sub-commands of the current command. The property is applicable only to commands that are containers for other commands (menus, submenus, or toolbars). Such container commands have the ID set to 0, and the IsSeparator property set to false.

17.3.7.1.8 ToolTip

Property: ToolTip as `string`

Description:

This property gets the text that is shown as a tool-tip for each command. If the command does not have a tooltip text, the property returns an empty string.

17.3.7.2 StyleVisionCommands

Properties:

[Count](#)¹⁰⁴⁰

[Item](#)¹⁰⁴⁰

Description:

Collection of [Command](#)¹⁰³⁸ objects to get access to command labels and IDs of the StyleVisionControl. Those commands can be executed with the [Exec](#)¹⁰⁴⁵ method and their status can be queried with [QueryStatus](#)¹⁰⁴⁶.

17.3.7.2.1 Count

Property: Count as `long`

Description:

Number of [Command](#)¹⁰³⁸ objects on this level of the collection.

17.3.7.2.2 Item

Property: Item (n as `long`) as [Command](#)¹⁰³⁸

Description:

Gets the command with the index n in this collection. Index is 1-based.

17.3.7.3 StyleVisionControl

Properties:

[IntegrationLevel](#) ¹⁰⁴³
[Appearance](#) ¹⁰⁴²
[Application](#) ¹⁰⁴²
[BorderStyle](#) ¹⁰⁴²
[CommandsList](#) ¹⁰⁴²
[EnableUserPrompts](#) ¹⁰⁴³
[MainMenu](#) ¹⁰⁴³
[Toolbars](#) ¹⁰⁴⁴

Methods:

[Open](#) ¹⁰⁴⁵
[Exec](#) ¹⁰⁴⁵
[QueryStatus](#) ¹⁰⁴⁶

Events:

[OnUpdateCmdUI](#) ¹⁰⁴⁸
[OnOpenedOrFocused](#) ¹⁰⁴⁷
[OnCloseEditingWindow](#) ¹⁰⁴⁶
[OnFileChangedAlert](#) ¹⁰⁴⁷
[OnDocumentOpened](#) ¹⁰⁴⁷
[OnValidationWindowUpdated](#) ¹⁰⁴⁸

This object is a complete ActiveX control and should only be visible if the StyleVision library is used in the Application Level mode.

17.3.7.3.1 Properties

The following properties are defined:

[IntegrationLevel](#) ¹⁰⁴³
[EnableUserPrompts](#) ¹⁰⁴³
[Appearance](#) ¹⁰⁴²
[BorderStyle](#) ¹⁰⁴²

Command related properties:

[CommandsList](#) ¹⁰⁴²
[MainMenu](#) ¹⁰⁴³
[Toolbars](#) ¹⁰⁴⁴

Access to StyleVisionAPI:

[Application](#) ¹⁰⁴²

17.3.7.3.1.1 Appearance

Property: Appearance as `short`

Dispatch Id: -520

Description:

A value not equal to 0 displays a client edge around the control. Default value is 0.

17.3.7.3.1.2 Application

Property: Application as `Application`

Dispatch Id: 1

Description:

The `Application` property gives access to the `Application` object of the complete StyleVision automation server API. The property is read-only.

17.3.7.3.1.3 BorderStyle

Property: BorderStyle as `short`

Dispatch Id: -504

Description:

A value of 1 displays the control with a thin border. Default value is 0.

17.3.7.3.1.4 CommandsList

Property: CommandList as `Commands`¹⁰⁴⁰ (read-only)

Dispatch Id: 1004

Description:

This property returns a flat list of all commands defined available with StyleVisionControl. To get commands organized according to their menu structure, use `MainMenu`¹⁰⁴³. To get toolbar commands, use `Toolbars`¹⁰⁴⁴.

```
public void GetAllStyleVisionCommands()
{
    // Get all commands from the StyleVision ActiveX control assigned to the current form
    StyleVisionControlLib.StyleVisionCommands commands =
```

```
this.axStyleVisionControl1.CommandList;
// Iterate through all commands
for (int i = 0; i < commands.Count; i++)
{
    // Get each command by index and output it to the console
    StyleVisionControlLib.StyleVisionCommand cmd =
axStyleVisionControl1.CommandList[i];
    Console.WriteLine("{0} {1} {2}", cmd.ID, cmd.Name, cmd.Label.Replace("&", ""));
}
}
```

C# example

17.3.7.3.1.5 *EnableUserPrompts*

Property: EnableUserPrompts as `boolean`

Dispatch Id: 1006

Description:

Setting this property to *false*, disables user prompts in the control. The default value is *true*.

17.3.7.3.1.6 *IntegrationLevel*

Property: IntegrationLevel as [ICActiveXIntegrationLevel](#)¹⁰⁵⁸

Dispatch Id: 1000

Description:

The `IntegrationLevel` property determines the operation mode of the control. See also [Integration at Application Level](#)¹⁰⁰⁵ and [Integration at Document Level](#)¹⁰⁰⁷ for more information.

Note: It is important to set this property immediately after the creation of the `StyleVisionControl` object.

17.3.7.3.1.7 *MainMenu*

Property: MainMenu as [Command](#)¹⁰³³ (read-only)

Dispatch Id: 1003

Description:

This property provides information about the structure and commands available in the `StyleVisionControl` main menu, as a `Command` object. The `Command` object contains all available submenus of `StyleVision` (for example "File", "Edit", "View" etc.). To access the submenu objects, use the `SubCommands` property of the `MainMenu` property. Each submenu is also a `Command` object. For each submenu, you can then further iterate through their

`SubCommands` property in order to get their corresponding child commands and separators (this technique may be used, for example, to create the application menu programmatically). Note that some menu commands act as containers ("parents") for other menu commands, in which case they also have a `SubCommands` property. To get the structure of all menu commands programmatically, you will need a recursive function.

```
public void GetStyleVisionMenus ()
{
    // Get the main menu from the StyleVision ActiveX control assigned to the current
    form
    StyleVisionControlLib.StyleVisionCommand mainMenu =
    this.axStyleVisionControl1.MainMenu;

    // Loop through entries of the main menu (e.g. File, Edit, etc.)
    for (int i = 0; i < mainMenu.SubCommands.Count; i++)
    {
        StyleVisionControlLib.StyleVisionCommand menu = mainMenu.SubCommands[i];
        Console.WriteLine("{0} menu has {1} children items (including separators)",
        menu.Label.Replace("&", ""), menu.SubCommands.Count);
    }
}
```

C# example

17.3.7.3.1.8 Toolbars

Property: Toolbars as [Commands](#)¹⁰⁴⁰ (read-only)

Dispatch Id: 1005

Description:

This property provides information about the structure of StyleVisionControl toolbars, as a `Command` object. The `Command` object contains all available toolbars of StyleVision. To access the toolbars, use the `SubCommands` property of the `Toolbars` property. Each toolbar is also a `Command` object. For each toolbar, you can then further iterate through their `SubCommands` property in order to get their commands (this technique may be used, for example, to create the application's toolbars programmatically).

```
public void GetStyleVisionToolbars ()
{
    // Get the application toolbars from the StyleVision ActiveX control assigned to the
    current form
    StyleVisionControlLib.StyleVisionCommands toolbars =
    this.axStyleVisionControl1.Toolbars;

    // Iterate through all toolbars
    for (int i = 0; i < toolbars.Count; i++)
    {
        StyleVisionControlLib.StyleVisionCommand toolbar = toolbars[i];
        Console.WriteLine ();
        Console.WriteLine ("The toolbar \"{0}\" has the following commands:",
```

```

toolbar.Label);

    // Iterate through all commands of this toolbar
    for (int j = 0; j < toolbar.SubCommands.Count; j++)
    {
        StyleVisionControlLib.StyleVisionCommand cmd = toolbar.SubCommands[j];
        // Output only command objects that are not separators
        if (!cmd.IsSeparator)
        {
            Console.WriteLine("{0}, {1}, {2}", cmd.ID, cmd.Name, cmd.Label.Replace("&",
""));
        }
    }
}
}
}

```

C# example

17.3.7.3.2 Methods

The following methods are defined:

[Open](#)¹⁰⁴⁵
[Exec](#)¹⁰⁴⁵
[QueryStatus](#)¹⁰⁴⁶

17.3.7.3.2.1 Exec

Method: `Exec (nCmdID as long) as boolean`

Dispatch Id: 6

Description:

This method calls the StyleVision command with the ID `nCmdID`. If the command can be executed, the method returns `true`. To get a list of all available commands, use [CommandsList](#)¹⁰⁴². To retrieve the status of any command, use [QueryStatus](#)¹⁰⁴⁶.

17.3.7.3.2.2 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](#)¹⁰⁵² and [StyleVisionControlPlaceholder.OpenProject](#)¹⁰⁵⁶.

17.3.7.3.2.3 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.

Bit	Value	Name	Meaning
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.

17.3.7.3.3 Events

The StyleVisionControl ActiveX control provides the following connection point events:

[OnUpdateCmdUI](#)¹⁰⁴⁸
[OnOpenedOrFocused](#)¹⁰⁴⁷
[OnCloseEditingWindow](#)¹⁰⁴⁶
[OnFileChangedAlert](#)¹⁰⁴⁷
[OnDocumentOpened](#)¹⁰⁴⁷
[OnValidationWindowUpdated](#)¹⁰⁴⁸

17.3.7.3.3.1 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.

17.3.7.3.3.2 *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](#)¹⁰⁵⁴ instead.

17.3.7.3.3.3 *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 hard disk 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.

17.3.7.3.3.4 *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).

17.3.7.3.3.5 *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.

17.3.7.3.3.6 *OnToolWindowUpdated*

Event: `OnToolWindowUpdated(pToolWnd as long)`

Dispatch Id: 1006

Description:

This event is triggered when the tool window is updated.

17.3.7.3.3.7 *OnUpdateCmdUI*

Event: `OnUpdateCmdUI ()`

Dispatch Id: 1003

Description:

Called frequently to give integrators a good opportunity to check status of StyleVision commands using [StyleVisionControl.QueryStatus](#)¹⁰⁴⁶. Do not perform long operations in this callback.

17.3.7.3.3.8 *OnValidationWindowUpdated*

Event: `OnValidationWindowUpdated ()`

Dispatch Id: 3

Description:

This event is triggered whenever the validation output window is updated with new information.

17.3.7.4 StyleVisionControlDocument

Properties:

[Appearance](#)¹⁰⁴⁹

[BorderStyle](#)¹⁰⁵⁰
[Document](#)¹⁰⁵⁰
[IsModified](#)¹⁰⁵⁰
[Path](#)¹⁰⁵⁰
[ReadOnly](#)¹⁰⁵¹

Methods:

[Exec](#)¹⁰⁵¹
[New](#)¹⁰⁵¹
[Open](#)¹⁰⁵²
[QueryStatus](#)¹⁰⁵²
[Reload](#)¹⁰⁵²
[Save](#)¹⁰⁵²
[SaveAs](#)¹⁰⁵³

Events:

[OnDocumentOpened](#)¹⁰⁵⁴
[OnDocumentClosed](#)¹⁰⁵³
[OnModifiedFlagChanged](#)¹⁰⁵⁴
[OnFileChangedAlert](#)¹⁰⁵⁴
[OnActivate](#)¹⁰⁵³

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.

17.3.7.4.1 Properties

The following properties are defined:

[ReadOnly](#)¹⁰⁵¹
[IsModified](#)¹⁰⁵⁰
[Path](#)¹⁰⁵⁰
[Appearance](#)¹⁰⁴⁹
[BorderStyle](#)¹⁰⁵⁰

Access to StyleVisionAPI:

[Document](#)¹⁰⁵⁰

17.3.7.4.1.1 Appearance

Property: Appearance as `short`

Dispatch Id: -520

Description:

A value not equal to 0 displays a client edge around the document control. Default value is 0.

17.3.7.4.1.2 *BorderStyle*

Property: `BorderStyle` as `short`

Dispatch Id: -504

Description:

A value of 1 displays the control with a thin border. Default value is 0.

17.3.7.4.1.3 *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 functionality which can be used with the document loaded in the control. The property is read-only.

17.3.7.4.1.4 *IsModified*

Property: `IsModified` as `boolean` (read-only)

Dispatch Id: 1006

Description:

`IsModified` is `true` if the document content has changed since the last open, reload or save operation. It is `false`, otherwise.

17.3.7.4.1.5 *Path*

Property: `Path` as `string`

Dispatch Id: 1005

Description:

Sets or gets the full path name of the document loaded into the control.

17.3.7.4.1.6 *ReadOnly*

Property: `ReadOnly` as `boolean`

Dispatch Id: 1007

Description:

Using this property you can turn on and off the read-only mode of the document. If `ReadOnly` is `true` it is not possible to do any modifications.

17.3.7.4.2 *Methods*

The following methods are defined:

Document handling:

[New](#)¹⁰⁵¹

[Open](#)¹⁰⁵²

[Reload](#)¹⁰⁵²

[Save](#)¹⁰⁵²

[SaveAs](#)¹⁰⁵³

Command Handling:

[Exec](#)¹⁰⁵¹

[QueryStatus](#)¹⁰⁵²

17.3.7.4.2.1 *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`. This method should be called only if there is currently an active document available in the application.

To get commands organized according to their menu structure, use the [MainMenu](#)¹⁰⁴³ property of `StyleVisionControl`. To get toolbar commands, use the [Toolbars](#)¹⁰⁴⁴ property of the `StyleVisionControl`.

17.3.7.4.2.2 *New*

Method: `New` () as `boolean`

Dispatch Id: 1000

Description:

This method initializes a new document inside the control.

17.3.7.4.2.3 *Open*

Method: `Open (strFileName as string) as boolean`

Dispatch Id: 1001

Description:

`Open` loads the file `strFileName` as the new document into the control.

17.3.7.4.2.4 *QueryStatus*

Method: `QueryStatus (nCmdID as long) as long`

Dispatch Id: 9

Description:

`QueryStatus` returns the enabled/disabled and checked/unchecked status of the command specified by `nCmdID`. The status is returned as a bit mask.

Bit	Value	Name	Meaning
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.

17.3.7.4.2.5 *Reload*

Method: `Reload() as boolean`

Dispatch Id: 1002

Description:

`Reload` updates the document content from the file system.

17.3.7.4.2.6 *Save*

Method: `Save() as boolean`

Dispatch Id: 1003

Description:

`Save` saves the current document at the location [Path](#)¹⁰⁵⁰.

17.3.7.4.2.7 *SaveAs*

Method: `SaveAs (strFileName as string) as boolean`

Dispatch Id: 1004

Description:

`SaveAs` sets [Path](#)¹⁰⁵⁰ to `strFileName` and then saves the document to this location.

17.3.7.4.3 Events

The `StyleVisionControlDocument` ActiveX control provides following connection point events:

[OnDocumentOpened](#)¹⁰⁵⁴
[OnDocumentClosed](#)¹⁰⁵³
[OnModifiedFlagChanged](#)¹⁰⁵⁴
[OnFileChangedAlert](#)¹⁰⁵⁴
[OnActivate](#)¹⁰⁵³
[OnSetEditorTitle](#)¹⁰⁵⁵

17.3.7.4.3.1 *OnActivate*

Event: `OnActivate ()`

Dispatch Id: 1005

Description:

This event is triggered when the document control is activated, has the focus, and is ready for user input.

17.3.7.4.3.2 *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.

17.3.7.4.3.3 *OnDocumentOpened*

Event: `OnDocumentOpened (objDocument as Document)`

Dispatch Id: 1000

Description:

This event is triggered whenever a document is opened in this control. The argument `objDocument` is a `Document` object from the StyleVision automation interface, and can be used to query for more details about the document, or perform additional operations.

17.3.7.4.3.4 *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.

17.3.7.4.3.5 *OnFileChangedAlert*

Event: `OnFileChangedAlert () as bool`

Dispatch Id: 1003

Description:

This event is triggered when the file loaded into this document control is changed on the hard disk by another application. Clients should return `true`, if they handled the event, or `false`, if StyleVision should handle it in its customary way, i.e. prompting the user for reload.

17.3.7.4.3.6 *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.

17.3.7.4.3.7 *OnSetEditorTitle*

Event: OnSetEditorTitle ()

Dispatch Id: 1006

Description:

This event is being raised when the contained document is being internally renamed.

17.3.7.5 StyleVisionControlPlaceholder

Properties available for all kinds of placeholder windows:

[PlaceholderWindowID](#)¹⁰⁵⁶

Properties for project placeholder window:

[Project](#)¹⁰⁵⁶

Methods for project placeholder window:

[OpenProject](#)¹⁰⁵⁶

[CloseProject](#)¹⁰⁵⁷

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.

17.3.7.5.1 Properties

The following properties are defined:

[PlaceholderWindowID](#)¹⁰⁵⁶

Access to StyleVisionAPI:

[Project](#)¹⁰⁵⁶

17.3.7.5.1.1 *Label*

Property: Label as `String` (read-only)

Dispatch Id: 1001

Description:

This property gives access to the title of the placeholder. The property is read-only.

17.3.7.5.1.2 PlaceholderWindowID

Property: PlaceholderWindowID as [StyleVisionControlPlaceholderWindow](#)¹⁰⁵⁸

Dispatch Id: 1

Description:

This property specifies 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 [StyleVisionControlPlaceholderWindow](#)¹⁰⁵⁸ enumeration. The control changes its state immediately and shows the new StyleVision window.

17.3.7.5.1.3 Project

Property: Project as Project (read-only)

Dispatch Id: 2

Description:

The Project property gives access to the Project object of the StyleVision automation server API. This interface provides additional functionality which can be used with the project loaded into the control. The property will return a valid project interface only if the placeholder window has [PlaceholderWindowID](#)¹⁰⁵⁶ with a value of StyleVisionXProjectWindow (=3). The property is read-only.

17.3.7.5.2 Methods

The following method is defined:

[OpenProject](#)¹⁰⁵⁶
[CloseProject](#)¹⁰⁵⁷

17.3.7.5.2.1 OpenProject

Method: OpenProject (strFileName as string) as boolean

Dispatch Id: 3

Description:

OpenProject loads the file strFileName as the new project into the control. The method will fail if the placeholder window has a [PlaceholderWindowID](#)¹⁰⁵⁶ different to StyleVisionXProjectWindow (=3).

17.3.7.5.2.2 *CloseProject*

Method: `CloseProject ()`

Dispatch Id: 4

Description:

`CloseProject` closes the project loaded by the control. The method will fail if the placeholder window has a [PlaceholderWindowID](#)¹⁰⁵⁶ different to `StyleVisionXProjectWindow (=3)`.

17.3.7.5.3 Events

The `StyleVisionControlPlaceholder` ActiveX control provides following connection point events:

[OnModifiedFlagChanged](#)¹⁰⁵⁷

17.3.7.5.3.1 *OnModifiedFlagChanged*

Event: `OnModifiedFlagChanged (i_bIsModified as boolean)`

Dispatch Id: 1

Description:

This event gets triggered only for placeholder controls with a [PlaceholderWindowID](#)¹⁰⁵⁶ of `StyleVisionXProjectWindow (=3)`. The event is fired whenever the project content changes between modified and unmodified state. The parameter `i_bIsModified` is `true` if the project contents differs from the original content, and `false`, otherwise.

17.3.7.5.3.2 *OnSetLabel*

Event: `OnSetLabel (i_strNewLabel as string)`

Dispatch Id: 1000

Description:

Raised when the title of the placeholder window is changed.

17.3.7.6 Enumerations

The following enumerations are defined:

[ICActiveXIntegrationLevel](#)¹⁰⁵⁸

[StyleVisionControlPlaceholderWindow](#)¹⁰⁵⁸

17.3.7.6.1 IActiveXIntegrationLevel

Possible values for the [IntegrationLevel](#) ¹⁰⁴³ property of the StyleVisionControl.

```
IActiveXIntegrationOnApplicationLevel = 0  
IActiveXIntegrationOnDocumentLevel = 1
```

17.3.7.6.2 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  
StyleVisionControlDBQueryWnd = 8
```

18 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](#) ¹⁰⁶⁰

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](#) ¹¹⁶⁶

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](#) ¹¹⁷²

Provides technical information about StyleVision.

- OS and memory requirements
- Altova XML Parser
- Altova XSLT and XQuery Engines
- Unicode support
- Internet usage

[License Information](#) ¹¹⁷⁴

Contains information about the way StyleVision is distributed and about its licensing.

- Electronic software distribution
- License metering
- Copyright
- End User License Agreement

18.1 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](#) ¹⁰⁶⁰
- [XSLT 2.0](#) ¹⁰⁶⁰
- [XSLT 3.0](#) ¹⁰⁶²
- [XQuery 1.0](#) ¹⁰⁶³
- [XQuery 3.1](#) ¹⁰⁶⁶

18.1.1 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 U+00A0 (the hexadecimal character reference for a non-breaking space) is inserted in the HTML code either as a character reference (` ` or ` `) or as an entity reference, ` `.

18.1.2 XSLT 2.0

This section:

- [Engine conformance](#) ¹⁰⁶¹
- [Backward compatibility](#) ¹⁰⁶¹
- [Namespaces](#) ¹⁰⁶¹
- [Schema awareness](#) ¹⁰⁶¹
- [Implementation-specific behavior](#) ¹⁰⁶²

Conformance

The XSLT 2.0 engine of StyleVision conforms to the World Wide Web Consortium's (W3C's) [XSLT 2.0 Recommendation of 23 January 2007](#) and [XPath 2.0 Recommendation of 14 December 2010](#).

Backwards Compatibility

The XSLT 2.0 engine is backwards compatible. Typically, the backwards compatibility of the XSLT 2.0 engine comes into play when using the XSLT 2.0 engine to process an XSLT 1.0 stylesheet or instruction. 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.

Namespace Name	Prefix	Namespace URI
XML Schema types	xs:	http://www.w3.org/2001/XMLSchema
XPath 2.0 functions	fn:	http://www.w3.org/2005/xpath-functions

Typically, these namespaces will be declared on the `xsl:stylesheet` or `xsl:transform` element, as shown in the following listing:

```
<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 certain XSLT 2.0 functions.

xsl:result-document

Additionally supported encodings are (the Altova-specific): `x-base16tobinary` and `x-base64tobinary`.

function-available

The function tests for the availability of in-scope functions (XSLT, XPath, and extension functions).

unparsed-text

The `href` argument 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`. Example: `xs:base64Binary(unparsed-text('chart.png', 'x-binarytobase64'))`.

unparsed-text-available

The `href` argument 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`.

18.1.3 XSLT 3.0

The XSLT 3.0 Engine of StyleVision conforms to the World Wide Web Consortium's (W3C's) [XSLT 3.0 Recommendation of 8 June 2017](#) and [XPath 3.1 Recommendation of 21 March 2017](#).

The XSLT 3.0 engine has the [same implementation-specific characteristics as the XSLT 2.0 engine](#)¹⁰⁶⁰. Additionally, it includes support for a number of new XSLT 3.0 features: XPath/XQuery 3.1 functions and operators, and the [XPath 3.1 specification](#).

Note: The optional [streaming feature](#) is not supported currently. The entire document will be loaded into memory regardless of the value of the `streamable` attribute. If enough memory is available, then: (i) the entire document will be processed—without streaming, (ii) [guaranteed-streamable constructs](#) will be processed correctly, as if the execution used streaming, and (iii) streaming errors will not be detected. In 64-bit apps, non-streaming execution should not be a problem. If memory does turn out to be an issue, a solution would be to add more memory to the system.

Namespaces

Your XSLT 3.0 stylesheet should declare the following namespaces in order for you to be able to use all the type constructors and functions available in XSLT 3.0. The prefixes given below are conventionally used; you could use alternative prefixes if you wish.

Namespace Name	Prefix	Namespace URI
----------------	--------	---------------

XML Schema types	xs:	http://www.w3.org/2001/XMLSchema
XPath/XQuery 3.1 functions	fn:	http://www.w3.org/2005/xpath-functions
Math functions	math:	http://www.w3.org/2005/xpath-functions/math
Map functions	map:	http://www.w3.org/2005/xpath-functions/map
Array functions	array:	http://www.w3.org/2005/xpath-functions/array
XQuery, XSLT, and XPath Error Codes	err:	http://www.w3.org/2005/xpath-functions/xqt-errors
Serialization functions	output	http://www.w3.org/2010/xslt-xquery-serialization

Typically, these namespaces will be declared on the `xsl:stylesheet` or `xsl:transform` element, as shown in the following listing:

```
<xsl:stylesheet version="3.0"
  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 3.0 engine uses the XPath and XQuery Functions and Operators 3.1 namespace (listed in the table above) as its **default functions namespace**. So you can use the functions of this namespace in your stylesheet without any prefix. If you declare the 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/XQuery 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')`.

18.1.4 XQuery 1.0

This section:

- [Engine conformance](#)¹⁰⁶⁴
- [Schema awareness](#)¹⁰⁶⁴
- [Encoding](#)¹⁰⁶⁴
- [Namespaces](#)¹⁰⁶¹
- [XML source and validation](#)¹⁰⁶⁵
- [Static and dynamic type checking](#)¹⁰⁶⁵
- [Library modules](#)¹⁰⁶⁵
- [External functions](#)¹⁰⁶⁵

- [Collations](#) ¹⁰⁶⁵
- [Precision of numeric data](#) ¹⁰⁶⁶
- [XQuery instructions support](#) ¹⁰⁶⁶
- [Implementation-specific behavior](#) ¹⁰⁶⁶

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.

Namespace Name	Prefix	Namespace URI
XML Schema types	xs:	http://www.w3.org/2001/XMLSchema
Schema instance	xsi:	http://www.w3.org/2001/XMLSchema-instance
Built-in functions	fn:	http://www.w3.org/2005/xpath-functions
Local functions	local:	http://www.w3.org/2005/xquery-local-functions

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 (see `fn:`) is the default functions namespace in XQuery, the `fn:` prefix does not need to be used when built-in functions are invoked (for example, `string("Hello")` will call the `fn:string` function). However, the prefix `fn:` can be used to call a built-in function without having to declare the namespace in the query prolog (for example: `fn:string("Hello")`).
- You can change the default functions namespace by declaring the `default function namespace` expression in the query prolog.
- When using types from the XML Schema namespace, the prefix `xs:` may be used without having to explicitly declare the namespaces and bind these prefixes to them in the query prolog. (Example: `xs:date` and `xs:yearMonthDuration`.) If you wish to use some other prefix for the XML Schema namespace, this must be explicitly declared in the query prolog. (Example: `declare namespace alt = "http://www.w3.org/2001/XMLSchema"; alt:date("2004-10-04")`.)
- Note that the `untypedAtomic`, `dayTimeDuration`, and `yearMonthDuration` datatypes have been moved, with the CRs of 23 January 2007, from the XPath Datatypes namespace to the XML Schema namespace, so: `xs:yearMonthDuration`.

If namespaces for functions, type constructors, node tests, etc are wrongly assigned, an error is reported. Note, however, that some functions have the same name as schema datatypes, e.g. `fn:string` and `fn:boolean`. (Both `xs:string` and `xs:boolean` are defined.) The namespace prefix determines whether the function or type constructor is used.

XML source document and validation

XML documents used in executing an XQuery document with the XQuery 1.0 Engine must be well-formed. However, they do not need to be valid according to an XML Schema. If the file is not valid, the invalid file is loaded without schema information. If the XML file is associated with an external schema and is valid according to it, then post-schema validation information is generated for the XML data and will be used for query evaluation.

Static and dynamic type checking

The static analysis phase checks aspects of the query such as syntax, whether external references (e.g. for modules) exist, whether invoked functions and variables are defined, and so on. If an error is detected in the static analysis phase, it is reported and the execution is stopped.

Dynamic type checking is carried out at run-time, when the query is actually executed. If a type is incompatible with the requirement of an operation, an error is reported. For example, the expression `xs:string("1") + 1` returns an error because the addition operation cannot be carried out on an operand of type `xs:string`.

Library Modules

Library modules store functions and variables so they can be reused. The XQuery 1.0 Engine supports modules that are stored in **a single external XQuery file**. Such a module file must contain a `module` declaration in its prolog, which associates a target namespace. Here is an example module:

```
module namespace libns="urn:module-library";
declare variable $libns:company := "Altova";
declare function libns:webaddress() { "http://www.altova.com" };
```

All functions and variables declared in the module belong to the namespace associated with the module. The module is used by importing it into an XQuery file with the `import module` statement in the query prolog. The `import module` statement only imports functions and variables declared directly in the library module file. As follows:

```
import module namespace modlib = "urn:module-library" at "modulefilename.xq";
if      ($modlib:company = "Altova")
then    modlib:webaddress()
else    error("No match found.")
```

External functions

External functions are not supported, i.e. in those expressions using the `external` keyword, as in:

```
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](#)¹⁰⁶⁸. To use a specific collation, supply

its URI as given in the [list of supported collations](#)¹⁰⁶⁸. 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.

Implementation-specific behavior

Given below is a description of how the XQuery and XQuery Update 1.0 engines handle implementation-specific aspects of certain functions.

unparsed-text

The `href` argument 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`. Example: `xs:base64Binary(unparsed-text('chart.png', 'x-binarytobase64'))`.

unparsed-text-available

The `href` argument 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`.

18.1.5 XQuery 3.1

The XQuery 3.1 Engine of StyleVision conforms to the World Wide Web Consortium's (W3C's) [XQuery 3.1 Recommendation of 21 March 2017](#) and includes support for XPath and XQuery Functions 3.1. The XQuery 3.1 specification is a superset of the 3.0 specification. The XQuery 3.1 engine therefore supports XQuery 3.0 features.

Namespaces

Your XQuery 3.1 document should declare the following namespaces in order for you to be able to use all the type constructors and functions available in XQuery 3.1. The prefixes given below are conventionally used; you could use alternative prefixes if you wish.

Namespace Name	Prefix	Namespace URI
----------------	--------	---------------

XML Schema types	<code>xs:</code>	<code>http://www.w3.org/2001/XMLSchema</code>
XPath/XQuery 3.1 functions	<code>fn:</code>	<code>http://www.w3.org/2005/xpath-functions</code>
Math functions	<code>math:</code>	<code>http://www.w3.org/2005/xpath-functions/math</code>
Map functions	<code>map:</code>	<code>http://www.w3.org/2005/xpath-functions/map</code>
Array functions	<code>array:</code>	<code>http://www.w3.org/2005/xpath-functions/array</code>
XQuery, XSLT, and XPath Error Codes	<code>err:</code>	<code>http://www.w3.org/2005/xpath-functions/xqt-errors</code>
Serialization functions	<code>output</code>	<code>http://www.w3.org/2010/xslt-xquery-serialization</code>

The following points should be noted:

- The XQuery 3.1 Engine recognizes the prefixes listed above as being bound to the corresponding namespaces.
- Since the built-in functions namespace listed above (see `fn:`) 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")`.)

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.

Implementation-specific behavior

Implementation-specific characteristics are the same as for [XQuery 1.0](#)¹⁰⁶³.

Additionally, the Altova-specific encoding `x-base64tobinary` can be used to create a binary result document, such as an image.

18.2 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.

This section mainly describes XPath/XQuery extension functions that have been created by Altova to provide additional operations. [These functions](#)¹⁰⁶⁹ can be computed by Altova's XSLT and XQuery engines according to the rules described in this section. For information about the regular XPath/XQuery functions, see [Altova's XPath/XQuery Function Reference](#).

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 Altova XSLT and XQuery engines read non-prefixed functions as belonging to the namespace <http://www.w3.org/2005/xpath-functions>, which is the default functions namespace specified in the XPath/XQuery functions specifications. 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 timezones of the values being compared need to be known. When the timezone is not explicitly given in such a value, the implicit timezone is used. The implicit timezone is taken from the system clock, and its value can be checked with the `implicit-timezone()` function.

Collations

The default collation is the Unicode codepoint collation, which compares strings on the basis of their Unicode codepoint. The engine uses the Unicode Collation Algorithm. Other supported collations are the [ICU collations](#) listed below; to use one of these, supply its URI as given in the table below. Any string comparisons, including for the `max` and `min` functions, will be made according to the specified collation. If the collation option is not specified, the default Unicode-codepoint collation is used.

Language	URIs
da: Danish	da_DK

de: German	de_AT, de_BE, de_CH, de_DE, de_LI, de_LU
en: English	en_AS, en_AU, en_BB, en_BE, en_BM, en_BW, en_BZ, en_CA, en_GB, en_GU, en_HK, en_IE, en_IN, 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
es: Spanish	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
fr: French	fr_BE, fr_BF, fr_BI, fr_BJ, fr_BL, fr_CA, fr_CD, fr_CF, fr_CG, fr_CH, fr_CI, fr_CM, fr_DJ, fr_FR, fr_GA, fr_GN, fr_GP, fr_GQ, fr_KM, fr_LU, fr_MC, fr_MF, fr_MG, fr_ML, fr_MQ, fr_NE, fr_RE, fr_RW, fr_SN, fr_TD, fr_TG
it: Italian	it_CH, it_IT
ja: Japanese	ja_JP
nb: Norwegian Bokmal	nb_NO
nl: Dutch	nl_AW, nl_BE, nl_NL
nn: Nynorsk	nn_NO
pt: Portuguese	pt_AO, pt_BR, pt_GW, pt_MZ, pt_PT, pt_ST
ru: Russian	ru_MD, ru_RU, ru_UA
sv: Swedish	sv_FI, sv_SE

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.

18.2.1 Altova Extension Functions

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace**, <http://www.altova.com/xslt-extensions>, and are indicated in this section with the prefix `altova:`, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

Functions defined in the W3C's XPath/XQuery Functions specifications can be used in: (i) XPath expressions in an XSLT context, and (ii) in XQuery expressions in an XQuery document. In this documentation we indicate the functions that can be used in the former context (XPath in XSLT) with an **xp** symbol and call them XPath functions; those functions that can be used in the latter (XQuery) context are indicated with an **xq** symbol; they work as XQuery functions. The W3C's XSLT specifications—not XPath/XQuery Functions specifications—also

define functions that can be used in XPath expressions in XSLT documents. These functions are marked with an **XSLT** symbol and are called XSLT functions. The XPath/XQuery and XSLT versions in which a function can be used are indicated in the description of the function (see *symbols below*). Functions from the XPath/XQuery and XSLT function libraries are listed without a prefix. Extension functions from other libraries, such as Altova extension functions, are listed with a prefix.

<i>XPath functions (used in XPath expressions in XSLT):</i>	XP1 XP2 XP3.1
<i>XSLT functions (used in XPath expressions in XSLT):</i>	XSLT1 XSLT2 XSLT3
<i>XQuery functions (used in XQuery expressions in XQuery):</i>	XQ1 XQ3.1

Usage of Altova extension functions

In order to use Altova extension functions, you must declare the Altova extension functions namespace (*first highlight in code listing below*) and then use the extension functions so that they are resolved as belonging to this namespace (see *second highlight*). The example below uses the Altova extension function named **age**.

```
<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/xslt-extensions">
  <xsl:output method="text" encoding="ISO-8859-1"/>
  <xsl:template match="Persons">
    <xsl:for-each select="Person">
      <xsl:value-of select="concat(Name, ': ')" />
      <xsl:value-of select="altova:age(xs:date(BirthDate))" />
      <xsl:value-of select="' years&#x0A;'" />
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

XSLT functions ¹⁰⁷¹

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](#) ¹⁰⁷³
- [Geolocation](#) ¹⁰⁹⁰
- [Image-related](#) ¹¹⁰²
- [Numeric](#) ¹¹⁰⁷
- [Sequence](#) ¹¹²⁸
- [String](#) ¹¹³⁶

- [Miscellaneous](#)¹¹⁴³

Barcode functions

[Altova's barcode extension functions](#)¹¹⁴⁵ enable barcodes to be generated and placed in output generated via XSLT stylesheets.

18.2.1.1 XSLT Functions

XSLT extension functions can be used in XPath expressions in an XSLT context. They will not work in a non-XSLT context (for instance, in an XQuery context).

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace**, <http://www.altova.com/xslt-extensions>, and are indicated in this section with the prefix **altova:**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

<i>XPath functions (used in XPath expressions in XSLT):</i>	XP1 XP2 XP3.1
<i>XSLT functions (used in XPath expressions in XSLT):</i>	XSLT1 XSLT2 XSLT3
<i>XQuery functions (used in XQuery expressions in XQuery):</i>	XQ1 XQ3.1

General 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 $p_1, p_2, p_3 \dots p_N$. Note the following points about usage: (i) The variables must be defined with names of the form p_x , 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: p_1 to p_N : The second argument will be the value of the variable p_1 , the third argument that of the variable p_2 , and so on; (iii) The variable values must be of type `item*`.

Example

```
<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 $\$p_1$, the third argument that assigned to the variable $\$p_2$, 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.

[[Top](#)¹⁰⁷¹]

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.

[[Top](#)¹⁰⁷¹]

18.2.1.2 XPath/XQuery Functions: Date and Time

Altova's date/time extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data held as XML Schema's various date and time datatypes. The functions in

this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace**, <http://www.altova.com/xslt-extensions>, and are indicated in this section with the prefix **altova:**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

<i>XPath functions (used in XPath expressions in XSLT):</i>	XP1 XP2 XP3.1
<i>XSLT functions (used in XPath expressions in XSLT):</i>	XSLT1 XSLT2 XSLT3
<i>XQuery functions (used in XQuery expressions in XQuery):</i>	XQ1 XQ3.1

▼ Grouped by functionality

- [Add a duration to xs:dateTime and return xs:dateTime](#)¹⁰⁷⁵
- [Add a duration to xs:date and return xs:date](#)¹⁰⁷⁷
- [Add a duration to xs:time and return xs:time](#)¹⁰⁷⁸
- [Format and retrieve durations](#)¹⁰⁷⁹
- [Remove timezone from functions that generate current date/time](#)¹⁰⁷⁹
- [Return days, hours, minutes, and seconds from durations](#)¹⁰⁸¹
- [Return weekday as integer from date](#)¹⁰⁸²
- [Return week number as integer from date](#)¹⁰⁸²
- [Build date, time, or duration type from lexical components of each type](#)¹⁰⁸⁴
- [Construct date, dateTime, or time type from string input](#)¹⁰⁸⁶
- [Age-related functions](#)¹⁰⁸⁷
- [Epoch time \(Unix time\) functions](#)¹⁰⁸⁹

▼ Listed alphabetically

[altova:add-days-to-date](#)¹⁰⁷⁷
[altova:add-days-to-dateTime](#)¹⁰⁷⁵
[altova:add-hours-to-dateTime](#)¹⁰⁷⁵
[altova:add-hours-to-time](#)¹⁰⁷⁸
[altova:add-minutes-to-dateTime](#)¹⁰⁷⁵
[altova:add-minutes-to-time](#)¹⁰⁷⁸
[altova:add-months-to-date](#)¹⁰⁷⁷
[altova:add-months-to-dateTime](#)¹⁰⁷⁵
[altova:add-seconds-to-dateTime](#)¹⁰⁷⁵
[altova:add-seconds-to-time](#)¹⁰⁷⁸
[altova:add-years-to-date](#)¹⁰⁷⁷
[altova:add-years-to-dateTime](#)¹⁰⁷⁵
[altova:age](#)¹⁰⁸⁷
[altova:age-details](#)¹⁰⁸⁷
[altova:build-date](#)¹⁰⁸⁴
[altova:build-duration](#)¹⁰⁸⁴
[altova:build-time](#)¹⁰⁸⁴
[altova:current-dateTime-no-TZ](#)¹⁰⁷⁹
[altova:current-date-no-TZ](#)¹⁰⁷⁹

[altova:current-time-no-TZ](#) ¹⁰⁷⁹
[altova:date-no-TZ](#) ¹⁰⁷⁹
[altova:dateTime-from-epoch](#) ¹⁰⁸⁹
[altova:dateTime-from-epoch-no-TZ](#) ¹⁰⁸⁹
[altova:dateTime-no-TZ](#) ¹⁰⁷⁹
[altova:days-in-month](#) ¹⁰⁸¹
[altova:epoch-from-dateTime](#) ¹⁰⁸⁹
[altova:hours-from-dateTimeDuration-accumulated](#) ¹⁰⁸¹
[altova:minutes-from-dateTimeDuration-accumulated](#) ¹⁰⁸¹
[altova:seconds-from-dateTimeDuration-accumulated](#) ¹⁰⁸¹
[altova:format-duration](#) ¹⁰⁷⁸
[altova:parse-date](#) ¹⁰⁸⁶
[altova:parse-dateTime](#) ¹⁰⁸⁶
[altova:parse-duration](#) ¹⁰⁷⁸
[altova:parse-time](#) ¹⁰⁸⁶
[altova:time-no-TZ](#) ¹⁰⁷⁹
[altova:weekday-from-date](#) ¹⁰⁸²
[altova:weekday-from-dateTime](#) ¹⁰⁸²
[altova:weeknumber-from-date](#) ¹⁰⁸³
[altova:weeknumber-from-dateTime](#) ¹⁰⁸³

[[Top](#) ¹⁰⁷³]

Add a duration to xs:dateTime **XP3.1 XQ3.1**

These functions add a duration to `xs:dateTime` and return `xs:dateTime`. The `xs:dateTime` type has a format of `CCYY-MM-DDThh:mm:ss.sss`. This is a concatenation of the `xs:date` and `xs:time` formats separated by the letter `T`. A timezone suffix (`+01:00`, for example) is optional.

▼ add-years-to-dateTime [altova:]

altova:add-years-to-dateTime (`DateTime` as `xs:dateTime`, `Years` as `xs:integer`) as `xs:dateTime` **XP3.1 XQ3.1**

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` **XP3.1 XQ3.1**

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
XP3.1 XQ3.1

Adds a duration in days to an xs:dateTime (see examples below). The second argument is the number of days to be added to the xs:dateTime supplied as the first argument. The result is of type xs:dateTime.

☐ Examples

- **altova:add-days-to-dateTime** (xs:dateTime("2014-01-15T14:00:00"), 10) returns 2014-01-25T14:00:00
- **altova:add-days-to-dateTime** (xs:dateTime("2014-01-15T14:00:00"), -8) returns 2014-01-07T14:00:00

▼ add-hours-to-dateTime [altova:]

altova:add-hours-to-dateTime (DateTime as xs:dateTime, Hours as xs:integer) as xs:dateTime
XP3.1 XQ3.1

Adds a duration in hours to an xs:dateTime (see examples below). The second argument is the number of hours to be added to the xs:dateTime supplied as the first argument. The result is of type xs:dateTime.

☐ Examples

- **altova:add-hours-to-dateTime** (xs:dateTime("2014-01-15T13:00:00"), 10) returns 2014-01-15T23:00:00
- **altova:add-hours-to-dateTime** (xs:dateTime("2014-01-15T13:00:00"), -8) returns 2014-01-15T05:00:00

▼ add-minutes-to-dateTime [altova:]

altova:add-minutes-to-dateTime (DateTime as xs:dateTime, Minutes as xs:integer) as xs:dateTime
XP3.1 XQ3.1

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"), 45) returns 2014-01-15T14:55:00
- **altova:add-minutes-to-dateTime** (xs:dateTime("2014-01-15T14:10:00"), -5) returns 2014-01-15T14:05:00

▼ add-seconds-to-dateTime [altova:]

altova:add-seconds-to-dateTime (DateTime as xs:dateTime, Seconds as xs:integer) as xs:dateTime
XP3.1 XQ3.1

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`

[[Top](#)¹⁰⁷³]

Add a duration to `xs:date` [XP3.1](#) [XQ3.1](#)

These functions add a duration to `xs:date` and return `xs:date`. The `xs:date` type has a format of CCYY-MM-DD.

▼ add-years-to-date [altova:]

`altova:add-years-to-date`(*Date as xs:date, Years as xs:integer*) as `xs:date` [XP3.1](#) [XQ3.1](#)

Adds a duration in years to a date. The second argument is the number of years to be added to the `xs:date` supplied as the first argument. The result is of type `xs:date`.

▣ Examples

- `altova:add-years-to-date`(`xs:date("2014-01-15")`, 10) returns `2024-01-15`
- `altova:add-years-to-date`(`xs:date("2014-01-15")`, -4) returns `2010-01-15`

▼ add-months-to-date [altova:]

`altova:add-months-to-date`(*Date as xs:date, Months as xs:integer*) as `xs:date` [XP3.1](#) [XQ3.1](#)

Adds a duration in months to a date. The second argument is the number of months to be added to the `xs:date` supplied as the first argument. The result is of type `xs:date`.

▣ Examples

- `altova:add-months-to-date`(`xs:date("2014-01-15")`, 10) returns `2014-11-15`
- `altova:add-months-to-date`(`xs:date("2014-01-15")`, -2) returns `2013-11-15`

▼ add-days-to-date [altova:]

`altova:add-days-to-date`(*Date as xs:date, Days as xs:integer*) as `xs:date` [XP3.1](#) [XQ3.1](#)

Adds a duration in days to a date. The second argument is the number of days to be added to the `xs:date` supplied as the first argument. The result is of type `xs:date`.

▣ Examples

- `altova:add-days-to-date`(`xs:date("2014-01-15")`, 10) returns `2014-01-25`
- `altova:add-days-to-date`(`xs:date("2014-01-15")`, -8) returns `2014-01-07`

[[Top](#)¹⁰⁷³]

Format and retrieve durations [XP3.1](#) [XQ3.1](#)

These functions parse an input `xs:duration` or `xs:string` and return, respectively, an `xs:string` or `xs:duration`.

▼ format-duration [altova:]

`altova:format-duration`(Duration as `xs:duration`, Picture as `xs:string`) as `xs:string` [XP3.1](#) [XQ3.1](#)

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` [XP3.1](#) [XQ3.1](#)

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"
- `altova:parse-duration`("Months:03 Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7", "Months:[M01] Days:[D01] Hours:[H01] Minutes:[m01]") returns "P3M2DT2H53M"

[[Top](#)¹⁰⁷³]

Add a duration to xs:time [XP3.1](#) [XQ3.1](#)

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` [XP3.1](#) [XQ3.1](#)

Adds a duration in hours to a time. The second argument is the number of hours to be added to the `xs:time` supplied as the first argument. The result is of type `xs:time`.

▣ Examples

- `altova:add-hours-to-time`(`xs:time("11:00:00")`, 10) returns 21:00:00
- `altova:add-hours-to-time`(`xs:time("11:00:00")`, -7) returns 04:00:00

▼ add-minutes-to-time [altova:]

altova:add-minutes-to-time(Time as *xs:time*, Minutes as *xs:integer*) as *xs:time* **XP3.1 XQ3.1**

Adds a duration in minutes to a time. The second argument is the number of minutes to be added to the *xs:time* supplied as the first argument. The result is of type *xs:time*.

☐ Examples

- **altova:add-minutes-to-time**(*xs:time*("14:10:00"), 45) returns 14:55:00
- **altova:add-minutes-to-time**(*xs:time*("14:10:00"), -5) returns 14:05:00

▼ add-seconds-to-time [altova:]

altova:add-seconds-to-time(Time as *xs:time*, Minutes as *xs:integer*) as *xs:time* **XP3.1 XQ3.1**

Adds a duration in seconds to a time. The second argument is the number of seconds to be added to the *xs:time* supplied as the first argument. The result is of type *xs:time*. The Seconds component can be in the range of 0 to 59.999.

☐ Examples

- **altova:add-seconds-to-time**(*xs:time*("14:00:00"), 20) returns 14:00:20
- **altova:add-seconds-to-time**(*xs:time*("14:00:00"), 20.895) returns 14:00:20.895

[[Top](#)¹⁰⁷³]

Remove the timezone part from date/time datatypes **XP3.1 XQ3.1**

These functions remove the timezone from the current *xs:dateTime*, *xs:date*, or *xs:time* values, respectively. Note that the difference between *xs:dateTime* and *xs:dateTimeStamp* is that in the case of the latter the timezone part is required (while it is optional in the case of the former). So the format of an *xs:dateTimeStamp* value is: CCYY-MM-DDThh:mm:ss.sss±hh:mm. or CCYY-MM-DDThh:mm:ss.sssZ. If the date and time is read from the system clock as *xs:dateTimeStamp*, the `current-dateTime-no-TZ()` function can be used to remove the timezone if so required.

▼ current-date-no-TZ [altova:]

altova:current-date-no-TZ() as *xs:date* **XP3.1 XQ3.1**

This function takes no argument. It removes the timezone part of `current-date()` (which is the current date according to the system clock) and returns an *xs:date* value.

☐ Examples

If the current date is 2014-01-15+01:00:

- **altova:current-date-no-TZ()** returns 2014-01-15

▼ current-dateTime-no-TZ [altova:]

altova:current-dateTime-no-TZ() as *xs:dateTime* **XP3.1 XQ3.1**

This function takes no argument. It removes the timezone part of `current-dateTime()` (which is the current date-and-time according to the system clock) and returns an *xs:dateTime* value.

Examples

If the current dateTime is 2014-01-15T14:00:00+01:00:

- `altova:current-dateTime-no-TZ()` returns 2014-01-15T14:00:00

current-time-no-TZ [altova:]

`altova:current-time-no-TZ()` as `xs:time` **XP3.1** **XQ3.1**

This function takes no argument. It removes the timezone part of `current-time()` (which is the current time according to the system clock) and returns an `xs:time` value.

Examples

If the current time is 14:00:00+01:00:

- `altova:current-time-no-TZ()` returns 14:00:00

date-no-TZ [altova:]

`altova:date-no-TZ(InputDate as xs:date)` as `xs:date` **XP3.1** **XQ3.1**

This function takes an `xs:date` argument, removes the timezone part from it, and returns an `xs:date` value. Note that the date is not modified.

Examples

- `altova:date-no-TZ(xs:date("2014-01-15+01:00"))` returns 2014-01-15

dateTime-no-TZ [altova:]

`altova:dateTime-no-TZ(InputDateTime as xs:dateTime)` as `xs:dateTime` **XP3.1** **XQ3.1**

This function takes an `xs:dateTime` argument, removes the timezone part from it, and returns an `xs:dateTime` value. Note that neither the date nor the time is modified.

Examples

- `altova:dateTime-no-TZ(xs:date("2014-01-15T14:00:00+01:00"))` returns 2014-01-15T14:00:00

time-no-TZ [altova:]

`altova:time-no-TZ(InputTime as xs:time)` as `xs:time` **XP3.1** **XQ3.1**

This function takes an `xs:time` argument, removes the timezone part from it, and returns an `xs:time` value. Note that the time is not modified.

Examples

- `altova:time-no-TZ(xs:time("14:00:00+01:00"))` returns 14:00:00

Return the number of days, hours, minutes, seconds from durations XP3.1 XQ3.1

These functions return the number of days in a month, and the number of hours, minutes, and seconds, respectively, from durations.

▼ days-in-month [altova:]

altova:days-in-month(Year as xs:integer, Month as xs:integer) as xs:integer XP3.1 XQ3.1

Returns the number of days in the specified month. The month is specified by means of the Year and Month arguments.

▣ Examples

- **altova:days-in-month**(2018, 10) returns 31
- **altova:days-in-month**(2018, 2) returns 28
- **altova:days-in-month**(2020, 2) returns 29

▼ hours-from-dayTimeDuration-accumulated

altova:hours-from-dayTimeDuration-accumulated(DayAndTime as xs:duration) as xs:integer XP3.1 XQ3.1

Returns the total number of hours in the duration submitted by the DayAndTime argument (which is of type xs:duration). The hours in the Day and Time components are added together to give a result that is an integer. A new hour is counted only for a full 60 minutes. Negative durations result in a negative hour value.

▣ Examples

- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5D")) returns 120, which is the total number of hours in 5 days.
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5DT2H")) returns 122, which is the total number of hours in 5 days plus 2 hours.
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5DT2H60M")) returns 123, which is the total number of hours in 5 days plus 2 hours and 60 mins.
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5DT2H119M")) returns 123, which is the total number of hours in 5 days plus 2 hours and 119 mins.
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5DT2H120M")) returns 124, which is the total number of hours in 5 days plus 2 hours and 120 mins.
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("-P5DT2H")) returns -122

▼ minutes-from-dayTimeDuration-accumulated

altova:minutes-from-dayTimeDuration-accumulated(DayAndTime as xs:duration) as xs:integer XP3.1 XQ3.1

Returns the total number of minutes in the duration submitted by the DayAndTime argument (which is of type xs:duration). The minutes in the Day and Time components are added together to give a result that is an integer. Negative durations result in a negative minute value.

▣ Examples

- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("PT60M")) returns 60
- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("PT1H")) returns 60, which is the total number of minutes in 1 hour.
- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("PT1H40M")) returns 100

- `altova:minutes-from-dayTimeDuration-accumulated(xs:duration("P1D"))` returns 1440, which is the total number of minutes in 1 day.
- `altova:minutes-from-dayTimeDuration-accumulated(xs:duration("-P1DT60M"))` returns -1500

▼ seconds-from-dayTimeDuration-accumulated

`altova:seconds-from-dayTimeDuration-accumulated(DayAndTime as xs:duration) as xs:integer XP3.1 XQ3.1`

Returns the total number of seconds in the duration submitted by the `DayAndTime` argument (which is of type `xs:duration`). The seconds in the `Day` and `Time` components are added together to give a result that is an integer. Negative durations result in a negative seconds value.

☐ Examples

- `altova:seconds-from-dayTimeDuration-accumulated(xs:duration("PT1M"))` returns 60, which is the total number of seconds in 1 minute.
- `altova:seconds-from-dayTimeDuration-accumulated(xs:duration("PT1H"))` returns 3600, which is the total number of seconds in 1 hour.
- `altova:seconds-from-dayTimeDuration-accumulated(xs:duration("PT1H2M"))` returns 3720
- `altova:seconds-from-dayTimeDuration-accumulated(xs:duration("P1D"))` returns 86400, which is the total number of seconds in 1 day.
- `altova:seconds-from-dayTimeDuration-accumulated(xs:duration("-P1DT1M"))` returns -86460

Return the weekday from xs:dateTime or xs:date XP3.1 XQ3.1

These functions return the weekday (as an integer) from `xs:dateTime` or `xs:date`. The days of the week are numbered (using the American format) from 1 to 7, with `Sunday=1`. In the European format, the week starts with Monday (=1). The American format, where `Sunday=1`, can be set by using the integer 0 where an integer is accepted to indicate the format.

▼ weekday-from-dateTime [altova:]

`altova:weekday-from-dateTime(DateTime as xs:dateTime) as xs:integer XP3.1 XQ3.1`

Takes a date-with-time as its single argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with `Sunday=1`. If the European format is required (where `Monday=1`), use the other signature of this function (see *next signature below*).

☐ Examples

- `altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00"))` returns 2, which would indicate a Monday.

`altova:weekday-from-dateTime(DateTime as xs:dateTime, Format as xs:integer) as xs:integer XP3.1 XQ3.1`

Takes a date-with-time as its first argument and returns the day of the week of this date as an integer. If the second (integer) argument is 0, then the weekdays are numbered 1 to 7 starting with `Sunday=1`. If the second argument is an integer other than 0, then `Monday=1`. If there is no second argument, the function is read as having the other signature of this function (see *previous signature*).

☐ Examples

- `altova:weekday-from-dateTime` (`xs:dateTime("2014-02-03T09:00:00")`, 1) returns 1, which would indicate a Monday
- `altova:weekday-from-dateTime` (`xs:dateTime("2014-02-03T09:00:00")`, 4) returns 1, which would indicate a Monday
- `altova:weekday-from-dateTime` (`xs:dateTime("2014-02-03T09:00:00")`, 0) returns 2, which would indicate a Monday.

▼ weekday-from-date [altova:]

`altova:weekday-from-date`(`Date` as `xs:date`) as `xs:integer` **XP3.1 XQ3.1**

Takes a date as its single argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with `Sunday=1`. If the European format is required (where `Monday=1`), use the other signature of this function (see next signature below).

▢ Examples

- `altova:weekday-from-date` (`xs:date("2014-02-03+01:00")`) returns 2, which would indicate a Monday.

`altova:weekday-from-date`(`Date` as `xs:date`, `Format` as `xs:integer`) as `xs:integer` **XP3.1 XQ3.1**

Takes a date as its first argument and returns the day of the week of this date as an integer. If the second (`Format`) argument is 0, then the weekdays are numbered 1 to 7 starting with `Sunday=1`. If the second argument is an integer other than 0, then `Monday=1`. If there is no second argument, the function is read as having the other signature of this function (see previous signature).

▢ Examples

- `altova:weekday-from-date` (`xs:date("2014-02-03")`, 1) returns 1, which would indicate a Monday
- `altova:weekday-from-date` (`xs:date("2014-02-03")`, 4) returns 1, which would indicate a Monday
- `altova:weekday-from-date` (`xs:date("2014-02-03")`, 0) returns 2, which would indicate a Monday.

[[Top](#)¹⁰⁷³]

Return the week number from `xs:dateTime` or `xs:date` **XP2 XQ1 XP3.1 XQ3.1**

These functions return the week number (as an integer) from `xs:dateTime` or `xs:date`. Week-numbering is available in the US, ISO/European, and Islamic calendar formats. Week-numbering is different in these calendar formats because the week is considered to start on different days (on Sunday in the US format, Monday in the ISO/European format, and Saturday in the Islamic format).

▼ weeknumber-from-date [altova:]

`altova:weeknumber-from-date`(`Date` as `xs:date`, `Calendar` as `xs:integer`) as `xs:integer` **XP2 XQ1 XP3.1 XQ3.1**

Returns the week number of the submitted `Date` argument as an integer. The second argument (`Calendar`) specifies the calendar system to follow.

Supported `calendar` values are:

- 0 = US calendar (*week starts Sunday*)
- 1 = ISO standard, European calendar (*week starts Monday*)
- 2 = Islamic calendar (*week starts Saturday*)

Default is 0.

Examples

- `altova:weeknumber-from-date` (`xs:date("2014-03-23")`, 0) returns 13
- `altova:weeknumber-from-date` (`xs:date("2014-03-23")`, 1) returns 12
- `altova:weeknumber-from-date` (`xs:date("2014-03-23")`, 2) returns 13
- `altova:weeknumber-from-date` (`xs:date("2014-03-23")`) returns 13

The day of the date in the examples above (`2014-03-23`) is Sunday. So the US and Islamic calendars are one week ahead of the European calendar on this day.

weeknumber-from-dateTime [altova:]

`altova:weeknumber-from-dateTime` (`DateTime` as `xs:dateTime`, `Calendar` as `xs:integer`) as `xs:integer` [XP2](#) [XQ1](#) [XP3.1](#) [XQ3.1](#)

Returns the week number of the submitted `DateTime` argument as an integer. The second argument (`Calendar`) specifies the calendar system to follow.

Supported `calendar` values are:

- 0 = US calendar (*week starts Sunday*)
- 1 = ISO standard, European calendar (*week starts Monday*)
- 2 = Islamic calendar (*week starts Saturday*)

Default is 0.

Examples

- `altova:weeknumber-from-dateTime` (`xs:dateTime("2014-03-23T00:00:00")`, 0) returns 13
- `altova:weeknumber-from-dateTime` (`xs:dateTime("2014-03-23T00:00:00")`, 1) returns 12
- `altova:weeknumber-from-dateTime` (`xs:dateTime("2014-03-23T00:00:00")`, 2) returns 13
- `altova:weeknumber-from-dateTime` (`xs:dateTime("2014-03-23T00:00:00")`) returns 13

The day of the `dateTime` in the examples above (`2014-03-23T00:00:00`) is Sunday. So the US and Islamic calendars are one week ahead of the European calendar on this day.

[[Top](#)¹⁰⁷³]

Build date, time, and duration datatypes from their lexical components [XP3.1](#) [XQ3.1](#)

The functions take the lexical components of the `xs:date`, `xs:time`, or `xs:duration` datatype as input arguments and combine them to build the respective datatype.

build-date [altova:]

`altova:build-date` (`Year` as `xs:integer`, `Month` as `xs:integer`, `Date` as `xs:integer`) as

xs:date XP3.1 XQ3.1

The first, second, and third arguments are, respectively, the year, month, and date. They are combined to build a value of `xs:date` type. The values of the integers must be within the correct range of that particular date part. For example, the second argument (for the month part) should not be greater than 12.

▣ Examples

- `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` XP3.1 XQ3.1

The first, second, and third arguments are, respectively, the hour (0 to 23), minutes (0 to 59), and seconds (0 to 59) values. They are combined to build a value of `xs:time` type. The values of the integers must be within the correct range of that particular time part. For example, the second (`Minutes`) argument should not be greater than 59. To add a timezone part to the value, use the other signature of this function (see *next signature*).

▣ Examples

- `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` XP3.1 XQ3.1

The first, second, and third arguments are, respectively, the hour (0 to 23), minutes (0 to 59), and seconds (0 to 59) values. The fourth argument is a string that provides the timezone part of the value. The four arguments are combined to build a value of `xs:time` type. The values of the integers must be within the correct range of that particular time part. For example, the second (`Minutes`) argument should not be greater than 59.

▣ Examples

- `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` XP3.1 XQ3.1

Takes two arguments to build a value of type `xs:yearMonthDuration`. The first argument provides the `Years` part of the duration value, while the second argument provides the `Months` part. If the second (`Months`) argument is greater than or equal to 12, then the integer is divided by 12; the quotient is added to the first argument to provide the `Years` part of the duration value while the remainder (of the division) provides the `Months` part. To build a duration of type `xs:dayTimeDuration`, see the next signature.

▣ Examples

- `altova:build-duration(2, 10)` returns `P2Y10M`
- `altova:build-duration(14, 27)` returns `P16Y3M`
- `altova:build-duration(2, 24)` returns `P4Y`

`altova:build-duration(Days as xs:integer, Hours as xs:integer, Minutes as xs:integer, Seconds as xs:integer) as xs:dayTimeDuration` XP3.1 XQ3.1

Takes four arguments and combines them to build a value of type `xs:dayTimeDuration`. The first argument provides the `Days` part of the duration value, the second, third, and fourth arguments provide,

respectively, the `Hours`, `Minutes`, and `Seconds` parts of the duration value. Each of the three `Time` arguments is converted to an equivalent value in terms of the next higher unit and the result is used for calculation of the total duration value. For example, 72 seconds is converted to `1M+12S` (1 minute and 12 seconds), and this value is used for calculation of the total duration value. To build a duration of type `xs:yearMonthDuration`., see the previous signature.

Examples

- `altova:build-duration(2, 10, 3, 56)` returns `P2DT10H3M56S`
- `altova:build-duration(1, 0, 100, 0)` returns `P1DT1H40M`
- `altova:build-duration(1, 0, 0, 3600)` returns `P1DT1H`

[[Top](#) ¹⁰⁷³]

Construct date, dateTime, and time datatypes from string input [XP2](#) [XQ1](#) [XP3.1](#) [XQ3.1](#)

These functions take strings as arguments and construct `xs:date`, `xs:dateTime`, or `xs:time` datatypes. The string is analyzed for components of the datatype based on a submitted pattern argument.

▼ parse-date [altova:]

`altova:parse-date(Date as xs:string, DatePattern as xs:string) as xs:date` [XP2](#) [XQ1](#) [XP3.1](#) [XQ3.1](#)

Returns the input string `Date` as an `xs:date` value. The second argument `DatePattern` specifies the pattern (sequence of components) of the input string. `DatePattern` is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

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`

▼ parse-dateTime [altova:]

`altova:parse-dateTime(DateTime as xs:string, DateTimePattern as xs:string) as xs:dateTime` [XP2](#) [XQ1](#) [XP3.1](#) [XQ3.1](#)

Returns the input string `DateTime` as an `xs:dateTime` value. The second argument `DateTimePattern` specifies the pattern (sequence of components) of the input string. `DateTimePattern` is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

D	Date
---	------

M	Month
Y	Year
H	Hour
m	minutes
s	seconds

The pattern in `DateTimePattern` must match the pattern in `DateTime`. Since the output is of type `xs:dateTime`, the output will always have the lexical format `YYYY-MM-DDTHH:mm:ss`.

Examples

- `altova:parse-dateTime(xs:string("09-12-2014 13:56:24"), "[M]-[D]-[Y] [H]:[m]:[s]")` returns `2014-09-12T13:56:24`
- `altova:parse-dateTime("time=13:56:24; date=09-12-2014", "time=[H]:[m]:[s]; date=[D]-[M]-[Y]")` returns `2014-12-09T13:56:24`

▼ parse-time [altova:]

`altova:parse-time(Time as xs:string, TimePattern as xs:string) as xs:time` [XP2](#) [XQ1](#) [XP3.1](#) [XQ3.1](#)

Returns the input string `Time` as an `xs:time` value. The second argument `TimePattern` specifies the pattern (sequence of components) of the input string. `TimePattern` is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

H	Hour
m	minutes
s	seconds

The pattern in `TimePattern` must match the pattern in `Time`. Since the output is of type `xs:time`, the output will always have the lexical format `HH:mm:ss`.

Examples

- `altova:parse-time(xs:string("13:56:24"), "[H]:[m]:[s]")` returns `13:56:24`
- `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](#)¹⁰⁷³]

Age-related functions [XP3.1](#) [XQ3.1](#)

These functions return the age as calculated (i) between one input argument date and the current date, or (ii) between two input argument dates. The `altova:age` function returns the age in terms of years, the `altova:age-details` function returns the age as a sequence of three integers giving the years, months, and days of the age.

▼ age [altova:]

altova:age(StartDate as xs:date) as xs:integer **XP3.1 XQ3.1**

Returns an integer that is the age *in years* of some object, counting from a start-date submitted as the argument and ending with the current date (taken from the system clock). If the input argument is a date anything greater than or equal to one year in the future, the return value will be negative.

▣ Examples

If the current date is 2014-01-15:

- **altova:age**(xs:date("2013-01-15")) returns 1
- **altova:age**(xs:date("2013-01-16")) returns 0
- **altova:age**(xs:date("2015-01-15")) returns -1
- **altova:age**(xs:date("2015-01-14")) returns 0

altova:age(StartDate as xs:date, EndDate as xs:date) as xs:integer **XP3.1 XQ3.1**

Returns an integer that is the age *in years* of some object, counting from a start-date that is submitted as the first argument up to an end-date that is the second argument. The return value will be negative if the first argument is one year or more later than the second argument.

▣ Examples

If the current date is 2014-01-15:

- **altova:age**(xs:date("2000-01-15"), xs:date("2010-01-15")) returns 10
- **altova:age**(xs:date("2000-01-15"), current-date()) returns 14 if the current date is 2014-01-15
- **altova:age**(xs:date("2014-01-15"), xs:date("2010-01-15")) returns -4

▼ age-details [altova:]

altova:age-details(InputDate as xs:date) as (xs:integer)* **XP3.1 XQ3.1**

Returns three integers that are, respectively, the years, months, and days between the date that is submitted as the argument and the current date (taken from the system clock). The sum of the returned `years+months+days` together gives the total time difference between the two dates (the input date and the current date). The input date may have a value earlier or later than the current date, but whether the input date is earlier or later is not indicated by the sign of the return values; the return values are always positive.

▣ Examples

If the current date is 2014-01-15:

- **altova:age-details**(xs:date("2014-01-16")) returns (0 0 1)
- **altova:age-details**(xs:date("2014-01-14")) returns (0 0 1)
- **altova:age-details**(xs:date("2013-01-16")) returns (1 0 1)
- **altova:age-details**(current-date()) returns (0 0 0)

altova:age-details(Date-1 as xs:date, Date-2 as xs:date) as (xs:integer)* **XP3.1 XQ3.1**

Returns three integers that are, respectively, the years, months, and days between the two argument dates. The sum of the returned `years+months+days` together gives the total time difference between the two input dates; it does not matter whether the earlier or later of the two dates is submitted as the first argument. The return values do not indicate whether the input date occurs earlier or later than the current date. Return values are always positive.

▣ Examples

- `altova:age-details(xs:date("2014-01-16"), xs:date("2014-01-15"))` returns (0 0 1)
- `altova:age-details(xs:date("2014-01-15"), xs:date("2014-01-16"))` returns (0 0 1)

[[Top](#) ¹⁰⁷³]

Epoch time (Unix time) functions [XP3.1](#) [XQ3.1](#)

Epoch time is a time system used on Unix systems. It defines any given point in time as being the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. Altova's Epoch time extension functions convert `xs:dateTime` values to Epoch time values and vice versa.

▼ `dateTime-from-epoch` [altova:]

`altova:dateTime-from-epoch` (Epoch as `xs:decimal` as `xs:dateTime` [XP3.1](#) [XQ3.1](#))

Epoch time is a time system used on Unix systems. It defines any given point in time as being the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. The `dateTime-from-epoch` function returns the `xs:dateTime` equivalent of an Epoch time, adjusts it for the local timezone, and includes the timezone information in the result.

The function takes an `xs:decimal` argument and returns an `xs:dateTime` value that includes a `TZ` (timezone) part. The result is obtained by calculating the UTC `dateTime` equivalent of the Epoch time, and adding to it the local timezone (taken from the system clock). For example, if the function is executed on a machine that has been set to be in a timezone of +01:00 (relative to UTC), then after the UTC `dateTime` equivalent has been calculated, one hour will be added to the result. The timezone information, which is an optional lexical part of the `xs:dateTime` result, is also reported in the `dateTime` result. Compare this result with that of `dateTime-from-epoch-no-TZ`, and also see the function `epoch-from-dateTime`.

▣ Examples

The examples below assume a local timezone of UTC +01:00. Consequently, the UTC `dateTime` equivalent of the submitted Epoch time will be incremented by one hour. The timezone is reported in the result.

- `altova:dateTime-from-epoch(34)` returns `1970-01-01T01:00:34+01:00`
- `altova:dateTime-from-epoch(62)` returns `1970-01-01T01:01:02+01:00`

▼ `dateTime-from-epoch-no-TZ` [altova:]

`altova:dateTime-from-epoch-no-TZ` (Epoch as `xs:decimal` as `xs:dateTime` [XP3.1](#) [XQ3.1](#))

Epoch time is a time system used on Unix systems. It defines any given point in time as being the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. The `dateTime-from-epoch-no-TZ` function returns the `xs:dateTime` equivalent of an Epoch time, adjusts it for the local timezone, but does not include the timezone information in the result.

The function takes an `xs:decimal` argument and returns an `xs:dateTime` value that does not include a `TZ` (timezone) part. The result is obtained by calculating the UTC `dateTime` equivalent of the Epoch time, and adding to it the local timezone (taken from the system clock). For example, if the function is executed on a machine that has been set to be in a timezone of +01:00 (relative to UTC), then after the UTC `dateTime` equivalent has been calculated, one hour will be added to the result. The timezone information,

which is an optional lexical part of the `xs:dateTime` result, is not reported in the `dateTime` result. Compare this result with that of `dateTime-from-epoch`, and also see the function `epoch-from-dateTime`.

Examples

The examples below assume a local timezone of UTC +01:00. Consequently, the UTC `dateTime` equivalent of the submitted Epoch time will be incremented by one hour. The timezone is not reported in the result.

- `altova:dateTime-from-epoch(34)` returns `1970-01-01T01:00:34`
- `altova:dateTime-from-epoch(62)` returns `1970-01-01T01:01:02`

epoch-from-dateTime [altova:]

`altova:epoch-from-dateTime(dateTimeValue as xs:dateTime) as xs:decimal XP3.1 XQ3.1`

Epoch time is a time system used on Unix systems. It defines any given point in time as being the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. The `epoch-from-dateTime` function returns the Epoch time equivalent of the `xs:dateTime` that is submitted as the argument of the function. Note that you might have to explicitly construct the `xs:dateTime` value. The submitted `xs:dateTime` value may or may not contain the optional `tz` (timezone) part.

Whether the timezone part is submitted as part of the argument or not, the local timezone offset (taken from the system clock) is subtracted from the submitted `dateTimeValue` argument. This produces the equivalent UTC time, from which the equivalent Epoch time is calculated. For example, if the function is executed on a machine that has been set to be in a timezone of +01:00 (relative to UTC), then one hour is subtracted from the submitted `dateTimeValue` before the Epoch value is calculated. Also see the function `dateTime-from-epoch`.

Examples

The examples below assume a local timezone of UTC +01:00. Consequently, one hour will be subtracted from the submitted `dateTime` before the Epoch time is calculated.

- `altova:epoch-from-dateTime(xs:dateTime("1970-01-01T01:00:34+01:00"))` returns `34`
- `altova:epoch-from-dateTime(xs:dateTime("1970-01-01T01:00:34"))` returns `34`
- `altova:epoch-from-dateTime(xs:dateTime("2021-04-01T11:22:33"))` returns `1617272553`

[[Top](#)¹⁰⁷³]

18.2.1.3 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.

<i>XPath functions (used in XPath expressions in XSLT):</i>	XP1 XP2 XP3.1
<i>XSLT functions (used in XPath expressions in XSLT):</i>	XSLT1 XSLT2 XSLT3
<i>XQuery functions (used in XQuery expressions in XQuery):</i>	XQ1 XQ3.1

▼ format-geolocation [altova:]

altova:format-geolocation(Latitude as xs:decimal, Longitude as xs:decimal, GeolocationOutputStringFormat as xs:integer) as xs:string **XP3.1 XQ3.1**

Takes the latitude and longitude as the first two arguments, and outputs the geolocation as a string. The third argument, **GeolocationOutputStringFormat**, is the format of the geolocation output string; it uses integer values from 1 to 4 to identify the output string format (see 'Geolocation output string formats' below). Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The [image-exif-data](#)¹¹⁰² function and the Exif metadata's attributes can be used to supply the input strings.

☐ Examples

- **altova:format-geolocation**(33.33, -22.22, 4) returns the xs:string "33.33 -22.22"
- **altova:format-geolocation**(33.33, -22.22, 2) returns the xs:string "33.33N 22.22W"
- **altova:format-geolocation**(-33.33, 22.22, 2) returns the xs:string "33.33S 22.22E"
- **altova:format-geolocation**(33.33, -22.22, 1) returns the xs:string "33°19'48.00"S 22°13'12.00"E"

☐ Geolocation output string formats:

The supplied latitude and longitude is formatted in one of the output formats given below. The desired format is identified by its integer ID (1 to 4). Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

1
Degrees, minutes, decimal seconds, with suffixed orientation (N/S, E/W) D°M'S.SS"N/S D°M'S.SS"E/W <i>Example:</i> 33°55'11.11"N 22°44'66.66"W

2
Decimal degrees, with suffixed orientation (N/S, E/W) D.DDN/S D.DDE/W <i>Example:</i> 33.33N 22.22W

3
Degrees, minutes, decimal seconds, with prefixed sign (+/-); plus sign for (N/E) is optional <code>+/-D°M'S.SS" +/-D°M'S.SS"</code> <i>Example:</i> <code>33°55'11.11" -22°44'66.66"</code>

4
Decimal degrees, with prefixed sign (+/-); plus sign for (N/E) is optional <code>+/-D.DD +/-D.DD</code> <i>Example:</i> <code>33.33 -22.22</code>

▣ *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*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

▼ parse-geolocation [altova:]

`altova:parse-geolocation(GeolocationInputString as xs:string) as xs:decimal+ XP3.1 XQ3.1`
Parses the supplied `GeolocationInputString` argument and returns the geolocation's latitude and longitude (in that order) as a sequence two `xs:decimal` items. The formats in which the geolocation input string can be supplied are listed below.

Note: The [image-exif-data](#)¹¹⁰² function and the Exif metadata's `@Geolocation`¹¹⁰² 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` (`48.858222222222, 24.2945`)
- `altova:parse-geolocation("48°51'29.6"N 24°17'40.2"E")` returns the sequence of two `xs:decimals` (`48.858222222222, 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, E/W)
`D°M'S.SS"N/S D°M'S.SS"W/E`
Example: 33°55'11.11"N 22°44'55.25"W
- Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (N/E) is optional
`+/-D°M'S.SS" +/-D°M'S.SS"`
Example: 33°55'11.11" -22°44'55.25"
- Degrees, decimal minutes, with suffixed orientation (N/S, E/W)
`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/E) is optional
`+/-D°M.MM' +/-D°M.MM'`
Example: +33°55.55' -22°44.44'
- Decimal degrees, with suffixed orientation (N/S, E/W)
`D.DDN/S D.DDW/E`
Example: 33.33N 22.22W
- Decimal degrees, with prefixed sign (+/-); the plus sign for (N/S E/W) is optional
`+/-D.DD +/-D.DD`
Example: 33.33 -22.22

Examples of format-combinations:

33.33N -22°44'55.25"
 33.33 22°44'55.25"W
 33.33 22.45

☐ Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute `Geolocation` from standard Exif metadata tags. `Geolocation` is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (see table below).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°

				13'11.73"E
--	--	--	--	------------

▼ geolocation-distance-km [altova:]

`altova:geolocation-distance-km(GeolocationInputString-1 as xs:string, GeolocationInputString-2 as xs:string) as xs:decimal XP3.1 XQ3.1`

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](#)¹¹⁰² function and the Exif metadata's [@Geolocation](#)¹¹⁰² 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, E/W)**
`D°M'S.SS"N/S D°M'S.SS"W/E`
Example: `33°55'11.11"N 22°44'55.25"W`
- **Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (N/E) is optional**
`+/-D°M'S.SS" +/-D°M'S.SS"`
Example: `33°55'11.11" -22°44'55.25"`
- **Degrees, decimal minutes, with suffixed orientation (N/S, E/W)**
`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/E) is optional**
`+/-D°M.MM' +/-D°M.MM'`
Example: `+33°55.55' -22°44.44'`

- **Decimal degrees, with suffixed orientation (N/S, E/W)**
`D.DDN/S D.DDW/E`
Example: 33.33N 22.22W
- **Decimal degrees, with prefixed sign (+/-); the plus sign for (N/S E/W) is optional**
`+/-D.DD +/-D.DD`
Example: 33.33 -22.22

Examples of format-combinations:

33.33N -22°44'55.25"
 33.33 22°44'55.25"W
 33.33 22.45

▣ Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute `Geolocation` from standard Exif metadata tags. `Geolocation` is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (see table below).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

▼ `geolocation-distance-mi` [altova:]

`altova:geolocation-distance-mi (GeolocationInputString-1 as xs:string, GeolocationInputString-2 as xs:string) as xs:decimal XP3.1 XQ3.1`

Calculates the distance between two geolocations in miles. The formats in which a geolocation input string can be supplied are listed below. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The [image-exif-data](#)¹¹⁰² function and the Exif metadata's [@Geolocation](#)¹¹⁰² 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, E/W)
 $D^{\circ}M'S.SS''N/S$ $D^{\circ}M'S.SS''W/E$
Example: 33°55'11.11"N 22°44'55.25"W
- Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (N/E) is optional
 $+/-D^{\circ}M'S.SS''$ $+/-D^{\circ}M'S.SS''$
Example: 33°55'11.11" -22°44'55.25"
- Degrees, decimal minutes, with suffixed orientation (N/S, E/W)
 $D^{\circ}M.MM'N/S$ $D^{\circ}M.MM'W/E$
Example: 33°55.55'N 22°44.44'W
- Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (N/E) is optional
 $+/-D^{\circ}M.MM'$ $+/-D^{\circ}M.MM'$
Example: +33°55.55' -22°44.44'
- Decimal degrees, with suffixed orientation (N/S, E/W)
 $D.DDN/S$ $D.DDW/E$
Example: 33.33N 22.22W
- Decimal degrees, with prefixed sign (+/-); the plus sign for (N/S E/W) is optional
 $+/-D.DD$ $+/-D.DD$
Example: 33.33 -22.22

Examples of format-combinations:

33.33N -22°44'55.25"
 33.33 22°44'55.25"W
 33.33 22.45

▣ Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute `Geolocation` from standard Exif metadata tags. `Geolocation` is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (see table below).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

▼ geolocations-bounding-rectangle [altova:]

`altova:geolocations-bounding-rectangle`(*Geolocations* as *xs:sequence*,
GeolocationOutputStringFormat as *xs:integer*) as *xs:string* **XP3.1 XQ3.1**

Takes a sequence of strings as its first argument; each string in the sequence is a geolocation. The function returns a sequence of two strings which are, respectively, the top-left and bottom-right geolocation coordinates of a bounding rectangle that is optimally sized to enclose all the geolocations submitted in the first argument. The formats in which a geolocation input string can be supplied are listed below (see '*Geolocation input string formats*'). Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

The function's second argument specifies the format of the two geolocation strings in the output sequence. The argument takes an integer value from 1 to 4, where each value identifies a different geolocation string format (see '*Geolocation output string formats*' below).

Note: The [image-exif-data](#)¹¹⁰² function and the Exif metadata's attributes can be used to supply the input strings.

Examples

- `altova:geolocations-bounding-rectangle` ("48.2143531 16.3707266", "51.50939 - 0.11832"), 1) returns the sequence ("51°30'33.804"N 0°7'5.952"W", "48°12'51.67116"N 16°22'14.61576"E")
- `altova:geolocations-bounding-rectangle` ("48.2143531 16.3707266", "51.50939 - 0.11832", "42.5584577 -70.8893334"), 4) returns the sequence ("51.50939 -70.8893334", "42.5584577 16.3707266")

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, E/W)
D°M'S.SS"N/S D°M'S.SS"W/E
Example: 33°55'11.11"N 22°44'55.25"W
- Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (N/E) is optional
+/-D°M'S.SS" +/-D°M'S.SS"
Example: 33°55'11.11" -22°44'55.25"
- Degrees, decimal minutes, with suffixed orientation (N/S, E/W)
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/E) is optional
 $\pm D^{\circ}M.MM'$ $\pm D^{\circ}M.MM'$
Example: $+33^{\circ}55.55'$ $-22^{\circ}44.44'$
- Decimal degrees, with suffixed orientation (N/S, E/W)
 $D.DDN/S$ $D.DDW/E$
Example: $33.33N$ $22.22W$
- Decimal degrees, with prefixed sign (+/-); the plus sign for (N/S E/W) is optional
 $\pm D.DD$ $\pm D.DD$
Example: 33.33 -22.22

Examples of format-combinations:

$33.33N -22^{\circ}44'55.25''$

$33.33 22^{\circ}44'55.25''W$

$33.33 22.45$

▣ Geolocation output string formats:

The supplied latitude and longitude is formatted in one of the output formats given below. The desired format is identified by its integer ID (1 to 4). Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

1
Degrees, minutes, decimal seconds, with suffixed orientation (N/S, E/W) $D^{\circ}M'S.SS''N/S$ $D^{\circ}M'S.SS''E/W$ <i>Example:</i> $33^{\circ}55'11.11''N$ $22^{\circ}44'66.66''W$
2
Decimal degrees, with suffixed orientation (N/S, E/W) $D.DDN/S$ $D.DDE/W$ <i>Example:</i> $33.33N$ $22.22W$
3
Degrees, minutes, decimal seconds, with prefixed sign (+/-); plus sign for (N/E) is optional $\pm D^{\circ}M'S.SS''$ $\pm D^{\circ}M'S.SS''$ <i>Example:</i> $33^{\circ}55'11.11''$ $-22^{\circ}44'66.66''$
4
Decimal degrees, with prefixed sign (+/-); plus sign for (N/E) is optional $\pm D.DD$ $\pm D.DD$ <i>Example:</i> 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: `GPSPLatitude`, `GPSPLatitudeRef`,

GPSLongitude, GPSLongitudeRef, with units added (see table below).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

▼ geolocation-within-polygon [altova:]

altova:geolocation-within-polygon(Geolocation as xs:string, ((PolygonPoint as xs:string)+)) as xs:boolean **XP3.1 XQ3.1**

Determines whether Geolocation (the first argument) is within the polygonal area described by the PolygonPoint arguments. If the PolygonPoint arguments do not form a closed figure (formed when the first point and the last point are the same), then the first point is implicitly added as the last point in order to close the figure. All the arguments (Geolocation and PolygonPoint+) are given by geolocation input strings (formats listed below). If the Geolocation argument is within the polygonal area, then the function returns true(); otherwise it returns false(). Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The [image-exif-data](#)¹¹⁰² function and the Exif metadata's [@Geolocation](#)¹¹⁰² attribute can be used to supply geolocation input strings.

☐ Examples

- **altova:geolocation-within-polygon**("33 -22", ("58 -32", "-78 -55", "48 24", "58 -32")) returns true()
- **altova:geolocation-within-polygon**("33 -22", ("58 -32", "-78 -55", "48 24")) returns true()
- **altova:geolocation-within-polygon**("33 -22", ("58 -32", "-78 -55", "48°51'29.6"N 24°17'40.2"W")) returns true()

☐ Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (") while unit indicators that are escaped are highlighted in blue (").

- Degrees, minutes, decimal seconds, with suffixed orientation (N/S, E/W)
D°M'S.SS"N/S D°M'S.SS"W/E

Example: 33°55'11.11"N 22°44'55.25"W

- Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (N/E) is optional
`+/-D°M'S.SS" +/-D°M'S.SS"`
Example: 33°55'11.11" -22°44'55.25"
- Degrees, decimal minutes, with suffixed orientation (N/S, E/W)
`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/E) is optional
`+/-D°M.MM' +/-D°M.MM'`
Example: +33°55.55' -22°44.44'
- Decimal degrees, with suffixed orientation (N/S, E/W)
`D.DDN/S D.DDW/E`
Example: 33.33N 22.22W
- Decimal degrees, with prefixed sign (+/-); the plus sign for (N/S E/W) is optional
`+/-D.DD +/-D.DD`
Example: 33.33 -22.22

Examples of format-combinations:

33.33N -22°44'55.25"
 33.33 22°44'55.25"W
 33.33 22.45

☐ Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **Geolocation** from standard Exif metadata tags. **Geolocation** is a concatenation of four Exif tags: **GPSLatitude**, **GPSLatitudeRef**, **GPSLongitude**, **GPSLongitudeRef**, with units added (see table below).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

▼ geolocation-within-rectangle [altova:]

altova:geolocation-within-rectangle(*Geolocation as xs:string, RectCorner-1 as xs:string, RectCorner-2 as xs:string*) as **xs:boolean** **XP3.1 XQ3.1**

Determines whether **Geolocation** (the first argument) is within the rectangle defined by the second and third arguments, **RectCorner-1** and **RectCorner-2**, which specify opposite corners of the rectangle. All the arguments (**Geolocation**, **RectCorner-1** and **RectCorner-2**) are given by geolocation input strings (*formats listed below*). If the **Geolocation** argument is within the rectangle, then the function returns `true()`; otherwise it returns `false()`. Latitude values range from +90 to -90 (N to S). Longitude values

range from +180 to -180 (E to W).

Note: The `image-exif-data`¹¹⁰² function and the Exif metadata's `@Geolocation`¹¹⁰² attribute can be used to supply geolocation input strings.

Examples

- `altova:geolocation-within-rectangle("33 -22", "58 -32", "-48 24")` returns `true()`
- `altova:geolocation-within-rectangle("33 -22", "58 -32", "48 24")` returns `false()`
- `altova:geolocation-within-rectangle("33 -22", "58 -32", "48°51'29.6"S 24°17'40.2"W")` returns `true()`

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (") while unit indicators that are escaped are highlighted in blue ("").

- Degrees, minutes, decimal seconds, with suffixed orientation (N/S, E/W)
`D°M'S.SS"N/S` `D°M'S.SS"W/E`
Example: `33°55'11.11"N 22°44'55.25"W`
- Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (N/E) is optional
`+/-D°M'S.SS"` `+/-D°M'S.SS"`
Example: `33°55'11.11" -22°44'55.25"`
- Degrees, decimal minutes, with suffixed orientation (N/S, E/W)
`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/E) is optional
`+/-D°M.MM'` `+/-D°M.MM'`
Example: `+33°55.55' -22°44.44'`
- Decimal degrees, with suffixed orientation (N/S, E/W)
`D.DDN/S` `D.DDW/E`
Example: `33.33N 22.22W`
- Decimal degrees, with prefixed sign (+/-); the plus sign for (N/S E/W) is optional
`+/-D.DD` `+/-D.DD`
Example: `33.33 -22.22`

Examples of format-combinations:

33.33N -22°44'55.25"

33.33 22°44'55.25"W

33.33 22.45

☐ Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute `Geolocation` from standard Exif metadata tags. `Geolocation` is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (see table below).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

[[Top](#)¹⁰⁹⁰]

18.2.1.4 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.

<i>XPath functions (used in XPath expressions in XSLT):</i>	<code>XP1</code> <code>XP2</code> <code>XP3.1</code>
<i>XSLT functions (used in XPath expressions in XSLT):</i>	<code>XSLT1</code> <code>XSLT2</code> <code>XSLT3</code>
<i>XQuery functions (used in XQuery expressions in XQuery):</i>	<code>XQ1</code> <code>XQ3.1</code>

▼ `suggested-image-file-extension` [`altova:`]

`altova:suggested-image-file-extension` (`Base64String` as `string`) as `string?` `XP3.1` `XQ3.1`

Takes the Base64 encoding of an image file as its argument and returns the file extension of the image as recorded in the Base64-encoding of the image. The returned value is a suggestion based on the image type information available in the encoding. If this information is not available, then an empty string is

returned. This function is useful if you wish to save a Base64 image as a file and wish to dynamically retrieve an appropriate file extension.

Examples

- `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? XP3.1 XQ3.1`

Takes a Base64-encoded JPEG 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 Altova Exif attributes are listed below.

Examples

- To access any one attribute, use the function like this:
`image-exif-data (/MyImages/Image20141130.01)/@GPSLatitude`
`image-exif-data (/MyImages/Image20141130.01)/@Geolocation`
- To access all the attributes, use the function like this:
`image-exif-data (/MyImages/Image20141130.01)/@*`
- To access the names of all the attributes, use the following expression:
`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

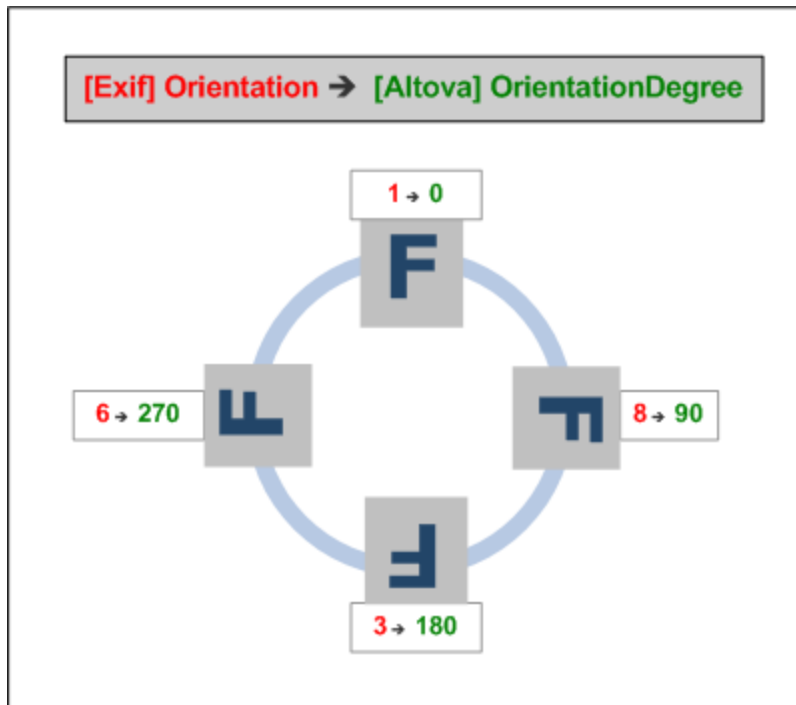
The Altova XPath/XQuery Engine generates the custom attribute `Geolocation` from standard Exif metadata tags. `Geolocation` is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (see table below).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

Altova Exif Attribute: OrientationDegree

The Altova XPath/XQuery Engine generates the custom attribute `OrientationDegree` from the Exif metadata tag `orientation`.

OrientationDegree translates the standard Exif tag `Orientation` from an integer value (1, 8, 3, or 6) to the respective degree values of each (0, 90, 180, 270), as shown in the figure below. Note that there are no translations of the `Orientation` values of 2, 4, 5, 7. (These orientations are obtained by flipping image 1 across its vertical center axis to get the image with a value of 2, and then rotating this image in 90-degree jumps clockwise to get the values of 7, 4, and 5, respectively).



Listing of standard Exif meta tags

- ImageWidth
- ImageLength
- BitsPerSample
- Compression
- PhotometricInterpretation
- Orientation
- SamplesPerPixel
- PlanarConfiguration
- YCbCrSubSampling
- YCbCrPositioning
- XResolution
- YResolution
- ResolutionUnit
- StripOffsets
- RowsPerStrip
- StripByteCounts
- JPEGInterchangeFormat
- JPEGInterchangeFormatLength
- TransferFunction

- WhitePoint
- PrimaryChromaticities
- YCbCrCoefficients
- ReferenceBlackWhite
- DateTime
- ImageDescription
- Make
- Model
- Software
- Artist
- Copyright

-
- ExifVersion
 - FlashpixVersion
 - ColorSpace
 - ComponentsConfiguration
 - CompressedBitsPerPixel
 - PixelXDimension
 - PixelYDimension
 - MakerNote
 - UserComment
 - RelatedSoundFile
 - DateTimeOriginal
 - DateTimeDigitized
 - SubSecTime
 - SubSecTimeOriginal
 - SubSecTimeDigitized
 - ExposureTime
 - FNumber
 - ExposureProgram
 - SpectralSensitivity
 - ISOSpeedRatings
 - OECF
 - ShutterSpeedValue
 - ApertureValue
 - BrightnessValue
 - ExposureBiasValue
 - MaxApertureValue
 - SubjectDistance
 - MeteringMode
 - LightSource
 - Flash
 - FocalLength
 - SubjectArea
 - FlashEnergy
 - SpatialFrequencyResponse
 - FocalPlaneXResolution
 - FocalPlaneYResolution
 - FocalPlaneResolutionUnit
 - SubjectLocation
 - ExposureIndex
 - SensingMethod
 - FileSource
 - SceneType

- CFAPattern
- CustomRendered
- ExposureMode
- WhiteBalance
- DigitalZoomRatio
- FocalLengthIn35mmFilm
- SceneCaptureType
- GainControl
- Contrast
- Saturation
- Sharpness
- DeviceSettingDescription
- SubjectDistanceRange
- ImageUniqueID

-
- GPSVersionID
 - GPSLatitudeRef
 - GPSLatitude
 - GPSLongitudeRef
 - GPSLongitude
 - GPSAltitudeRef
 - GPSAltitude
 - GPSTimeStamp
 - GPSSatellites
 - GPSStatus
 - GPSMeasureMode
 - GPSDOP
 - GPSSpeedRef
 - GPSSpeed
 - GPSTrackRef
 - GPSTrack
 - GPSImgDirectionRef
 - GPSImgDirection
 - GPSMapDatum
 - GPSDestLatitudeRef
 - GPSDestLatitude
 - GPSDestLongitudeRef
 - GPSDestLongitude
 - GPSDestBearingRef
 - GPSDestBearing
 - GPSDestDistanceRef
 - GPSDestDistance
 - GPSProcessingMethod
 - GPSAreaInformation
 - GPSDateStamp
 - GPSDifferential

[[Top](#)¹¹⁰²]

18.2.1.5 XPath/XQuery Functions: Numeric

Altova's numeric extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace**, <http://www.altova.com/xslt-extensions>, and are indicated in this section with the prefix **altova:**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

<i>XPath functions (used in XPath expressions in XSLT):</i>	XP1 XP2 XP3.1
<i>XSLT functions (used in XPath expressions in XSLT):</i>	XSLT1 XSLT2 XSLT3
<i>XQuery functions (used in XQuery expressions in XQuery):</i>	XQ1 XQ3.1

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 XP1 XP2 XQ1 XP3.1 XQ3.1**

Generates a number each time the function is called. The first number, which is generated the first time the function is called, is specified by the `StartsWith` argument. Each subsequent call to the function generates a new number, this number being incremented over the previously generated number by the value specified in the `Increment` argument. In effect, the `altova:generate-auto-number` function creates a counter having a name specified by the `ID` argument, with this counter being incremented each time the function is called. If the value of the `ResetOnChange` argument changes from that of the previous function call, then the value of the number to be generated is reset to the `StartsWith` value. Auto-numbering can also be reset by using the `altova:reset-auto-number` function.

☐ Examples

- **altova:generate-auto-number**("ChapterNumber", 1, 1, "SomeString") will return one number each time the function is called, starting with 1, and incrementing by 1 with each call to the function. As long as the fourth argument remains "SomeString" in each subsequent call, the incrementing will continue. When the value of the fourth argument changes, the counter (called `ChapterNumber`) will reset to 1. The value of `ChapterNumber` can also be reset by a call to the `altova:reset-auto-number` function, like this: `altova:reset-auto-number("ChapterNumber")`.

▼ reset-auto-number [altova:]

altova:reset-auto-number(ID as xs:string) **XP1 XP2 XQ1 XP3.1 XQ3.1**

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`.

[[Top](#)¹¹⁰⁷]

Numeric functions

hex-string-to-integer [altova:]

`altova:hex-string-to-integer(HexString as xs:string) as xs:integer` **XP3.1** **XQ3.1**

Takes a string argument that is the Base-16 equivalent of an integer in the decimal system (Base-10), and returns the decimal integer.

Examples

- `altova:hex-string-to-integer('1')` returns 1
- `altova:hex-string-to-integer('9')` returns 9
- `altova:hex-string-to-integer('A')` returns 10
- `altova:hex-string-to-integer('B')` returns 11
- `altova:hex-string-to-integer('F')` returns 15
- `altova:hex-string-to-integer('G')` returns an error
- `altova:hex-string-to-integer('10')` returns 16
- `altova:hex-string-to-integer('01')` returns 1
- `altova:hex-string-to-integer('20')` returns 32
- `altova:hex-string-to-integer('21')` returns 33
- `altova:hex-string-to-integer('5A')` returns 90
- `altova:hex-string-to-integer('USA')` returns an error

integer-to-hex-string [altova:]

`altova:integer-to-hex-string(Integer as xs:integer) as xs:string` **XP3.1** **XQ3.1**

Takes an integer argument and returns its Base-16 equivalent as a string.

Examples

- `altova:integer-to-hex-string(1)` returns '1'
- `altova:integer-to-hex-string(9)` returns '9'
- `altova:integer-to-hex-string(10)` returns 'A'
- `altova:integer-to-hex-string(11)` returns 'B'
- `altova:integer-to-hex-string(15)` returns 'F'
- `altova:integer-to-hex-string(16)` returns '10'
- `altova:integer-to-hex-string(32)` returns '20'
- `altova:integer-to-hex-string(33)` returns '21'
- `altova:integer-to-hex-string(90)` returns '5A'

[[Top](#)¹¹⁰⁷]

Number-formatting functions

[[Top](#) ¹¹⁰⁷]

18.2.1.6 XPath/XQuery Functions: Schema

The Altova extension functions listed below return schema information. Given below are descriptions of the functions, together with (i) examples and (ii) a listing of schema components and their respective properties. They can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines and are available in XPath/XQuery contexts.

Schema information from schema documents

The function `altova:schema` has two arguments: one with zero arguments and the other with two arguments. The zero-argument function returns the whole schema. You can then, from this starting point, navigate into the schema to locate the schema components you want. The two-argument function returns a specific component kind that is identified by its QName. In both cases, the return value is a function. To navigate into the returned component, you must select a property of that specific component. If the property is a non-atomic item (that is, if it is a component), then you can navigate further by selecting a property of this component. If the selected property is an atomic item, then the value of the item is returned and you cannot navigate any further.

Note: In XPath expressions, the schema must be imported into the processing environment (for example, into XSLT) with the `xslt:import-schema` instruction. In XQuery expressions, the schema must be explicitly imported using a [schema import](#).

Schema information from XML nodes

The function `altova:type` submits the node of an XML document and returns the node's type information from the PSVI.

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.

<i>XPath functions (used in XPath expressions in XSLT):</i>	<code>XP1</code> <code>XP2</code> <code>XP3.1</code>
<i>XSLT functions (used in XPath expressions in XSLT):</i>	<code>XSLT1</code> <code>XSLT2</code> <code>XSLT3</code>
<i>XQuery functions (used in XQuery expressions in XQuery):</i>	<code>XQ1</code> <code>XQ3.1</code>

▼ Schema (zero arguments)

```
altova:schema() as (function(xs:string) as item(*))? XP3.1 XQ3.1
```

Returns the `schema` component as a whole. You can navigate further into the `schema` component by selecting one of the `schema` component's properties.

- If this property is a component, you can navigate another step deeper by selecting one of this component's properties. This step can be repeated to navigate further into the schema.
- If the component is an atomic value, the atomic value is returned and you cannot navigate any deeper.

The properties of the `schema` component are:

```
"type definitions"
"attribute declarations"
"element declarations"
"attribute group definitions"
"model group definitions"
"notation declarations"
"identity-constraint definitions"
```

The properties of all other component kinds (besides `schema`) are listed below.

Note: In XQuery expressions, the schema must be explicitly imported. In XPath expressions, the schema must have been imported into the processing environment, for example, into XSLT with the `xslt:import` instruction.

Examples

- `import schema "" at "C:\Test\ExpReport.xsd"; for $typedef in altova:schema() ("type definitions") return $typedef ("name")` returns the names of all simple types or complex types in the schema
- `import schema "" at "C:\Test\ExpReport.xsd"; altova:schema() ("type definitions")[1] ("name")` returns the name of the first of all simple types or complex types in the schema

Components and their properties

Assertion

Property name	Property type	Property value
kind	string	"Assertion"
test	XPath Property Record	

Attribute Declaration

Property name	Property type	Property value
kind	string	"Attribute Declaration"
name	string	Local name of the attribute
target namespace	string	Namespace URI of the attribute

type definition	Simple Type or Complex Type	
scope	A function with properties ("class": "Scope", "variety": "global" or "local", "parent": the containing Complex Type or Attribute Group)	
value constraint	If present, a function with properties ("class": "Value Constraint", "variety": "fixed" or "default", "value": atomic value, "lexical form": string. Note that the "value" property is not available for namespace-sensitive types	
inheritable	boolean	

Attribute Group Declaration

Property name	Property type	Property value
kind	string	"Attribute Group Definition"
name	string	Local name of the attribute group
target namespace	string	Namespace URI of the attribute group
attribute uses	Sequence of (Attribute Use)	
attribute wildcard	Optional Attribute Wildcard	

Attribute Use

Property name	Property type	Property value
kind	string	"Attribute Use"
required	boolean	true if the attribute is required, false if optional
value constraint	See Attribute Declaration	
inheritable	boolean	

Attribute Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any" "enumeration" "not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	

process contents	string ("strict" "lax" "skip")	
------------------	--------------------------------	--

☐ Complex Type

Property name	Property type	Property value
kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
base type definition	Complex Type Definition	
final	Sequence of strings ("restriction" "extension")	
context	Empty sequence (not implemented)	
derivation method	string ("restriction" "extension")	
abstract	boolean	
attribute uses	Sequence of Attribute Use	
attribute wildcard	Optional Attribute Wildcard	
content type	function with properties: ("class": "Content Type", "variety": string ("element-only" "empty" "mixed" "simple"), particle: optional Particle, "open content": function with properties ("class": "Open Content", "mode": string ("interleave" "suffix"), "wildcard": Wildcard), "simple type definition": Simple Type)	
prohibited substitutions	Sequence of strings ("restriction" "extension")	
assertions	Sequence of Assertion	

☐ Element Declaration

Property name	Property type	Property value
kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
type definition	Simple Type or Complex Type	
type table	function with properties ("class": "Type	

	Table", "alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	function with properties ("class": "Scope", "variety": ("global" "local"), "parent": optional Complex Type)	
value constraint	see Attribute Declaration	
nillable	boolean	
identity-constraint definitions	Sequence of Identity Constraint	
substitution group affiliations	Sequence of Element Declaration	
substitution group exclusions	Sequence of strings ("restriction" "extension")	
disallowed substitutions	Sequence of strings ("restriction" "extension" "substitution")	
abstract	boolean	

Element Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any" "enumeration" "not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict" "lax" "skip")	

Facet

Property name	Property type	Property value
kind	string	The name of the facet, for example "minLength" or "enumeration"
value	depends on facet	The value of the facet
fixed	boolean	
typed-value	For the enumeration facet only, array(xs:anyAtomicType*)	An array containing the enumeration values, each of which may in general be a sequence of atomic values. (Note: for the enumeration facet, the "value"

		property is a sequence of strings, regardless of the actual type)
--	--	---

Identity Constraint

Property name	Property type	Property value
kind	string	"Identity-Constraint Definition"
name	string	Local name of the constraint
target namespace	string	Namespace URI of the constraint
identity-constraint category	string ("key" "unique" "keyRef")	
selector	XPath Property Record	
fields	Sequence of XPath Property Record	
referenced key	(For keyRef only): Identity Constraint	The corresponding key constraint

Model Group

Property name	Property type	Property value
kind	string	"Model Group"
compositor	string ("sequence" "choice" "all")	
particles	Sequence of Particle	

Model Group Definition

Property name	Property type	Property value
kind	string	"Model Group Definition"
name	string	Local name of the model group
target namespace	string	Namespace URI of the model group
model group	Model Group	

Notation

Property name	Property type	Property value
kind	string	"Notation Declaration"
name	string	Local name of the notation
target namespace	string	Namespace URI of the notation
system identifier	anyURI	
public identifier	string	

Particle

Property name	Property type	Property value
kind	string	"Particle"
min occurs	integer	
max occurs	integer, or string("unbounded")	
term	Element Declaration, Element Wildcard, or ModelGroup	

Simple Type

Property name	Property type	Property value
kind	string	"Simple Type Definition"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
final	Sequence of string("restriction" "extension" "list" "union")	
context	containing component	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	Empty sequence (not implemented)	
variety	string ("atomic" "list" "union")	
primitive type definition	Simple Type	
item type definition	(for list types only) Simple Type	
member type definitions	(for union types only) Sequence of Simple Type	

Type Alternative

Property name	Property type	Property value
kind	string	"Type Alternative"
test	XPath Property Record	
type definition	Simple Type or Complex Type	

XPath Property Record

Property name	Property type	Property value
namespace bindings	Sequence of functions with properties ("prefix": string, "namespace": anyURI)	

default namespace	anyURI	
base URI	anyURI	The static base URI of the XPath expression
expression	string	The XPath expression as a string

▼ Schema (two arguments)

altova:schema(ComponentKind as xs:string, Name as xs:QName) as (function(xs:string) as item(*)?) **XP3.1 XQ3.1**

Returns the component kind that is specified in the first argument which has a name that is the same as the name supplied in the second argument. You can navigate further by selecting one of the component's properties.

- If this property is a component, you can navigate another step deeper by selecting one of this component's properties. This step can be repeated to navigate further into the schema.
- If the component is an atomic value, the atomic value is returned and you cannot navigate any deeper.

Note: In XQuery expressions, the schema must be explicitly imported. In XPath expressions, the schema must have been imported into the processing environment, for example, into XSLT with the `xslt:import` instruction.

▣ Examples

- `import schema "" at "C:\Test\ExpReport.xsd";`
`altova:schema("element declaration", xs:QName("OrgChart"))("type definition")`
`("content type")("particles")[3]!.("term")("kind")`
returns the `kind` property of the `term` of the third `particles` component. This `particles` component is a descendant of the `element declaration` having a `QName` of `OrgChart`
- `import schema "" at "C:\Test\ExpReport.xsd";`
`let $typedef := altova:schema("type definition", xs:QName("emailType"))`
`for $facet in $typedef ("facets")`
`return [$facet ("kind"), $facet("value")]`
returns, for each `facet` of each `emailType` component, an array containing that facet's `kind` and `value`

Components and their properties

▣ Assertion

Property name	Property type	Property value
kind	string	"Assertion"
test	XPath Property Record	

▣ Attribute Declaration

Property name	Property type	Property value
kind	string	"Attribute Declaration"
name	string	Local name of the attribute
target namespace	string	Namespace URI of the attribute
type definition	Simple Type or Complex Type	
scope	A function with properties ("class": "Scope", "variety": "global" or "local", "parent": the containing Complex Type or Attribute Group)	
value constraint	If present, a function with properties ("class": "Value Constraint", "variety": "fixed" or "default", "value": atomic value, "lexical form": string. Note that the "value" property is not available for namespace-sensitive types	
inheritable	boolean	

Attribute Group Declaration

Property name	Property type	Property value
kind	string	"Attribute Group Definition"
name	string	Local name of the attribute group
target namespace	string	Namespace URI of the attribute group
attribute uses	Sequence of (Attribute Use)	
attribute wildcard	Optional Attribute Wildcard	

Attribute Use

Property name	Property type	Property value
kind	string	"Attribute Use"
required	boolean	true if the attribute is required, false if optional
value constraint	See Attribute Declaration	
inheritable	boolean	

Attribute Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety":	

	"any" "enumeration" "not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict" "lax" "skip")	

Complex Type

Property name	Property type	Property value
kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
base type definition	Complex Type Definition	
final	Sequence of strings ("restriction" "extension")	
context	Empty sequence (not implemented)	
derivation method	string ("restriction" "extension")	
abstract	boolean	
attribute uses	Sequence of Attribute Use	
attribute wildcard	Optional Attribute Wildcard	
content type	function with properties: ("class": "Content Type", "variety": string ("element-only" "empty" "mixed" "simple"), particle: optional Particle, "open content": function with properties ("class": "Open Content", "mode": string ("interleave" "suffix"), "wildcard": Wildcard), "simple type definition": Simple Type)	
prohibited substitutions	Sequence of strings ("restriction" "extension")	
assertions	Sequence of Assertion	

Element Declaration

Property name	Property type	Property value
kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)

target namespace	string	Namespace URI of the type (empty if anonymous)
type definition	Simple Type or Complex Type	
type table	function with properties ("class": "Type Table", "alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	function with properties ("class": "Scope", "variety": ("global" "local"), "parent": optional Complex Type)	
value constraint	see Attribute Declaration	
nillable	boolean	
identity-constraint definitions	Sequence of Identity Constraint	
substitution group affiliations	Sequence of Element Declaration	
substitution group exclusions	Sequence of strings ("restriction" "extension")	
disallowed substitutions	Sequence of strings ("restriction" "extension" "substitution")	
abstract	boolean	

Element Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any" "enumeration" "not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings")	
process contents	string ("strict" "lax" "skip")	

Facet

Property name	Property type	Property value
kind	string	The name of the facet, for example "minLength" or "enumeration"
value	depends on facet	The value of the facet
fixed	boolean	

typed-value	For the enumeration facet only, array(xs:anyAtomicType*)	An array containing the enumeration values, each of which may in general be a sequence of atomic values. (Note: for the enumeration facet, the "value" property is a sequence of strings, regardless of the actual type)
-------------	--	--

☐ Identity Constraint

Property name	Property type	Property value
kind	string	"Identity-Constraint Definition"
name	string	Local name of the constraint
target namespace	string	Namespace URI of the constraint
identity-constraint category	string ("key" "unique" "keyRef")	
selector	XPath Property Record	
fields	Sequence of XPath Property Record	
referenced key	(For keyRef only): Identity Constraint	The corresponding key constraint

☐ Model Group

Property name	Property type	Property value
kind	string	"Model Group"
compositor	string ("sequence" "choice" "all")	
particles	Sequence of Particle	

☐ Model Group Definition

Property name	Property type	Property value
kind	string	"Model Group Definition"
name	string	Local name of the model group
target namespace	string	Namespace URI of the model group
model group	Model Group	

☐ Notation

Property name	Property type	Property value
kind	string	"Notation Declaration"
name	string	Local name of the notation
target namespace	string	Namespace URI of the notation

system identifier	anyURI	
public identifier	string	

Particle

Property name	Property type	Property value
kind	string	"Particle"
min occurs	integer	
max occurs	integer, or string("unbounded")	
term	Element Declaration, Element Wildcard, or ModelGroup	

Simple Type

Property name	Property type	Property value
kind	string	"Simple Type Definition"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
final	Sequence of string("restriction" "extension" "list" "union")	
context	containing component	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	Empty sequence (not implemented)	
variety	string ("atomic" "list" "union")	
primitive type definition	Simple Type	
item type definition	(for list types only) Simple Type	
member type definitions	(for union types only) Sequence of Simple Type	

Type Alternative

Property name	Property type	Property value
kind	string	"Type Alternative"
test	XPath Property Record	
type definition	Simple Type or Complex Type	

▣ XPath Property Record

Property name	Property type	Property value
namespace bindings	Sequence of functions with properties ("prefix": string, "namespace": anyURI)	
default namespace	anyURI	
base URI	anyURI	The static base URI of the XPath expression
expression	string	The XPath expression as a string

▼ Type

`altova:type(Node as item?) as (function(xs:string) as item(*))?` **XP3.1 XQ3.1**

The function `altova:type` submits an element or attribute node of an XML document and returns the node's type information from the PSVI.

Note: The XML document must have a schema declaration so that the schema can be referenced.

▣ *Examples*

- `for $element in //Email`
`let $type := altova:type($element)`
`return $type`
returns a function that contains the `Email` node's type information
- `for $element in //Email`
`let $type := altova:type($element)`
`return $type ("kind")`
takes the `Email` node's type component (Simple Type or Complex Type) and returns the value of the component's `kind` property

The "`_props`" parameter returns the properties of the selected component. For example:

- `for $element in //Email`
`let $type := altova:type($element)`
`return ($type ("kind"), $type ("_props"))`
takes the `Email` node's type component (Simple Type or Complex Type) and returns (i) the value of the component's `kind` property, and then (ii) the properties of that component.

Components and their properties

▣ Assertion

Property name	Property type	Property value
kind	string	"Assertion"
test	XPath Property Record	

Attribute Declaration

Property name	Property type	Property value
kind	string	"Attribute Declaration"
name	string	Local name of the attribute
target namespace	string	Namespace URI of the attribute
type definition	Simple Type or Complex Type	
scope	A function with properties ("class": "Scope", "variety": "global" or "local", "parent": the containing Complex Type or Attribute Group)	
value constraint	If present, a function with properties ("class": "Value Constraint", "variety": "fixed" or "default", "value": atomic value, "lexical form": string. Note that the "value" property is not available for namespace-sensitive types	
inheritable	boolean	

Attribute Group Declaration

Property name	Property type	Property value
kind	string	"Attribute Group Definition"
name	string	Local name of the attribute group
target namespace	string	Namespace URI of the attribute group
attribute uses	Sequence of (Attribute Use)	
attribute wildcard	Optional Attribute Wildcard	

Attribute Use

Property name	Property type	Property value
kind	string	"Attribute Use"
required	boolean	true if the attribute is required, false if optional
value constraint	See Attribute Declaration	
inheritable	boolean	

Attribute Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class":	

	"Namespace Constraint", "variety": "any" "enumeration" "not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict" "lax" "skip")	

Complex Type

Property name	Property type	Property value
kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
base type definition	Complex Type Definition	
final	Sequence of strings ("restriction" "extension")	
context	Empty sequence (not implemented)	
derivation method	string ("restriction" "extension")	
abstract	boolean	
attribute uses	Sequence of Attribute Use	
attribute wildcard	Optional Attribute Wildcard	
content type	function with properties: ("class": "Content Type", "variety": string ("element-only" "empty" "mixed" "simple"), particle: optional Particle, "open content": function with properties ("class": "Open Content", "mode": string ("interleave" "suffix"), "wildcard": Wildcard), "simple type definition": Simple Type)	
prohibited substitutions	Sequence of strings ("restriction" "extension")	
assertions	Sequence of Assertion	

Element Declaration

Property name	Property type	Property value
kind	string	"Complex Type"
name	string	Local name of the type (empty if

		anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
type definition	Simple Type or Complex Type	
type table	function with properties ("class": "Type Table", "alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	function with properties ("class": "Scope", "variety": ("global" "local"), "parent": optional Complex Type)	
value constraint	see Attribute Declaration	
nillable	boolean	
identity-constraint definitions	Sequence of Identity Constraint	
substitution group affiliations	Sequence of Element Declaration	
substitution group exclusions	Sequence of strings ("restriction" "extension")	
disallowed substitutions	Sequence of strings ("restriction" "extension" "substitution")	
abstract	boolean	

Element Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any" "enumeration" "not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings")	
process contents	string ("strict" "lax" "skip")	

Facet

Property name	Property type	Property value
kind	string	The name of the facet, for example "minLength" or "enumeration"
value	depends on facet	The value of the facet

fixed	boolean	
typed-value	For the enumeration facet only, array(xs:anyAtomicType*)	An array containing the enumeration values, each of which may in general be a sequence of atomic values. (Note: for the enumeration facet, the "value" property is a sequence of strings, regardless of the actual type)

Identity Constraint

Property name	Property type	Property value
kind	string	"Identity-Constraint Definition"
name	string	Local name of the constraint
target namespace	string	Namespace URI of the constraint
identity-constraint category	string ("key" "unique" "keyRef")	
selector	XPath Property Record	
fields	Sequence of XPath Property Record	
referenced key	(For keyRef only): Identity Constraint	The corresponding key constraint

Model Group

Property name	Property type	Property value
kind	string	"Model Group"
compositor	string ("sequence" "choice" "all")	
particles	Sequence of Particle	

Model Group Definition

Property name	Property type	Property value
kind	string	"Model Group Definition"
name	string	Local name of the model group
target namespace	string	Namespace URI of the model group
model group	Model Group	

Notation

Property name	Property type	Property value
kind	string	"Notation Declaration"
name	string	Local name of the notation

target namespace	string	Namespace URI of the notation
system identifier	anyURI	
public identifier	string	

▣ Particle

Property name	Property type	Property value
kind	string	"Particle"
min occurs	integer	
max occurs	integer, or string("unbounded")	
term	Element Declaration, Element Wildcard, or ModelGroup	

▣ Simple Type

Property name	Property type	Property value
kind	string	"Simple Type Definition"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
final	Sequence of string("restriction" "extension" "list" "union")	
context	containing component	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	Empty sequence (not implemented)	
variety	string ("atomic" "list" "union")	
primitive type definition	Simple Type	
item type definition	(for list types only) Simple Type	
member type definitions	(for union types only) Sequence of Simple Type	

▣ Type Alternative

Property name	Property type	Property value
kind	string	"Type Alternative"
test	XPath Property Record	
type definition	Simple Type or Complex Type	

☐ XPath Property Record

Property name	Property type	Property value
namespace bindings	Sequence of functions with properties ("prefix": string, "namespace": anyURI)	
default namespace	anyURI	
base URI	anyURI	The static base URI of the XPath expression
expression	string	The XPath expression as a string

18.2.1.7 XPath/XQuery Functions: Sequence

Altova's sequence extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace**, <http://www.altova.com/xslt-extensions>, and are indicated in this section with the prefix **altova:**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

<i>XPath functions (used in XPath expressions in XSLT):</i>	XP1 XP2 XP3.1
<i>XSLT functions (used in XPath expressions in XSLT):</i>	XSLT1 XSLT2 XSLT3
<i>XQuery functions (used in XQuery expressions in XQuery):</i>	XQ1 XQ3.1

▼ attributes [altova:]

altova:attributes(AttributeName as xs:string) as attribute()* XP3.1 XQ3.1

Returns all attributes that have a local name which is the same as the name supplied in the input argument, *AttributeName*. The search is case-sensitive and conducted along the `attribute::` axis. This means that the context node must be the parent element node.

☐ Examples

- **altova:attributes("MyAttribute")** returns `MyAttribute()*`

altova:attributes(AttributeName as xs:string, SearchOptions as xs:string) as attribute()* XP3.1 XQ3.1

Returns all attributes that have a local name which is the same as the name supplied in the input argument, `AttributeName`. The search is case-sensitive and conducted along the `attribute::` axis. The context node must be the parent element node. The second argument is a string containing option flags. Available flags are:

r = switches to a regular-expression search; `AttributeName` must then be a regular-expression search string;

f = If this option is specified, then `AttributeName` provides a full match; otherwise `AttributeName` need only partially match an attribute name to return that attribute. For example: if **f** is not specified, then `MyAtt` will return `MyAttribute`;

i = switches to a case-insensitive search;

p = includes the namespace prefix in the search; `AttributeName` should then contain the namespace prefix, for example: `altova:MyAttribute`.

The flags can be written in any order. Invalid flags will generate errors. One or more flags can be omitted. The empty string is allowed, and will produce the same effect as the function having only one argument (*previous signature*). However, an empty sequence is not allowed as the second argument.

Examples

- `altova:attributes("MyAttribute", "rfip")` returns `MyAttribute()*`
- `altova:attributes("MyAttribute", "pri")` returns `MyAttribute()*`
- `altova:attributes("MyAtt", "rip")` returns `MyAttribute()*`
- `altova:attributes("MyAttributes", "rfip")` returns no match
- `altova:attributes("MyAttribute", "")` returns `MyAttribute()*`
- `altova:attributes("MyAttribute", "Rip")` returns an unrecognized-flag error.
- `altova:attributes("MyAttribute",)` returns a missing-second-argument error.

▼ elements [altova:]

`altova:elements(ElementName as xs:string) as element()*` **XP3.1 XQ3.1**

Returns all elements that have a local name which is the same as the name supplied in the input argument, `ElementName`. The search is case-sensitive and conducted along the `child::` axis. The context node must be the parent node of the element/s being searched for.

Examples

- `altova:elements("MyElement")` returns `MyElement()*`

`altova:elements(ElementName as xs:string, SearchOptions as xs:string) as element()*`
XP3.1 XQ3.1

Returns all elements that have a local name which is the same as the name supplied in the input argument, `ElementName`. The search is case-sensitive and conducted along the `child::` axis. The context node must be the parent node of the element/s being searched for. The second argument is a string containing option flags. Available flags are:

r = switches to a regular-expression search; `ElementName` must then be a regular-expression search string;

f = If this option is specified, then `ElementName` provides a full match; otherwise `ElementName` need only partially match an element name to return that element. For example: if **f** is not specified, then `MyElem` will return `MyElement`;

i = switches to a case-insensitive search;

p = includes the namespace prefix in the search; `ElementName` should then contain the namespace prefix, for example: `altova:MyElement`.

The flags can be written in any order. Invalid flags will generate errors. One or more flags can be omitted. The empty string is allowed, and will produce the same effect as the function having only one argument

(previous signature). However, an empty sequence is not allowed.

Examples

- `altova:elements("MyElement", "rip")` returns `MyElement()*`
- `altova:elements("MyElement", "pri")` returns `MyElement()*`
- `altova:elements("MyElement", "")` returns `MyElement()*`
- `altova:elements("MyElem", "rip")` returns `MyElement()*`
- `altova:elements("MyElements", "rfip")` returns no match
- `altova:elements("MyElement", "Rip")` returns an unrecognized-flag error.
- `altova:elements("MyElement",)` returns a missing-second-argument error.

find-first [altova:]

`altova:find-first((Sequence as item()*), (Condition(Sequence-Item as xs:boolean)) as item())?` [XP3.1](#) [XQ3.1](#)

This function takes two arguments. The first argument is a sequence of one or more items of any datatype. The second argument, `Condition`, is a reference to an XPath function that takes one argument (has an arity of 1) and returns a boolean. Each item of `sequence` is submitted, in turn, to the function referenced in `Condition`. (*Remember:* This function takes a single argument.) The first `sequence` item that causes the function in `Condition` to evaluate to `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:

- `altova:find-first(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1))` returns `xs:string http://www.altova.com/index.html`

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()* XP3.1 XQ3.1)
This function takes three arguments:
```

- The first two arguments, `seq-01` and `seq-02`, are sequences of one or more items of any datatype.
- The third argument, `Condition`, is a reference to an XPath function that takes two arguments (has an arity of 2) and returns a boolean.

The items of `Seq-01` and `Seq-02` are passed in ordered pairs (one item from each sequence making up a pair) as the arguments of the function in `Condition`. The pairs are ordered as follows.

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X1 Y2), (X1 Y3) ... (X1 Yn), (X2 Y1), (X2 Y2) ... (Xn Yn)
```

The first ordered pair that causes the `Condition` function to evaluate to `true()` is returned as the result of `altova:find-first-combination`. Note that: (i) If the `Condition` function iterates through the submitted argument pairs and does not once evaluate to `true()`, then `altova:find-first-combination` returns *No results*; (ii) The result of `altova:find-first-combination` will always be a pair of items (of any datatype) or no item at all.

▣ Examples

- `altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32})` returns the sequence of `xs:integers` (11, 21)
- `altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33})` returns the sequence of `xs:integers` (11, 22)
- `altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a+$b = 34})`

returns the sequence of `xs:integers` (11, 23)

▼ find-first-pair [altova:]

`altova:find-first-pair((Seq-01 as item()*), (Seq-02 as item()*), (Condition(Seq-01-Item, Seq-02-Item as xs:boolean))) as item()*` **XP3.1 XQ3.1**

This function takes three arguments:

- The first two arguments, `seq-01` and `seq-02`, are sequences of one or more items of any datatype.
- The third argument, `Condition`, is a reference to an XPath function that takes two arguments (has an arity of 2) and returns a boolean.

The items of `seq-01` and `seq-02` are passed in ordered pairs as the arguments of the function in `Condition`. The pairs are ordered as follows.

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X2 Y2), (X3 Y3) ... (Xn Yn)
```

The first ordered pair that causes the `Condition` function to evaluate to `true()` is returned as the result of `altova:find-first-pair`. Note that: (i) If the `Condition` function iterates through the submitted argument pairs and does not once evaluate to `true()`, then `altova:find-first-pair` returns *No results*; (ii) The result of `altova:find-first-pair` will always be a pair of items (of any datatype) or no item at all.

▣ Examples

- `altova:find-first-pair(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32})` returns the sequence of `xs:integers` (11, 21)
- `altova:find-first-pair(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33})` returns *No results*

Notice from the two examples above that the ordering of the pairs is: (11, 21) (12, 22) (13, 23) ... (20, 30). This is why the second example returns *No results* (because no ordered pair gives a sum of 33).

▼ find-first-pair-pos [altova:]

`altova:find-first-pair-pos((Seq-01 as item()*), (Seq-02 as item()*), (Condition(Seq-01-Item, Seq-02-Item as xs:boolean))) as xs:integer` **XP3.1 XQ3.1**

This function takes three arguments:

- The first two arguments, `seq-01` and `seq-02`, are sequences of one or more items of any datatype.
- The third argument, `Condition`, is a reference to an XPath function that takes two arguments (has an arity of 2) and returns a boolean.

The items of `seq-01` and `seq-02` are passed in ordered pairs as the arguments of the function in `Condition`. The pairs are ordered as follows.

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X2 Y2), (X3 Y3) ... (Xn Yn)
```

The index position of the first ordered pair that causes the `Condition` function to evaluate to `true()` is returned as the result of `altova:find-first-pair-pos`. Note that if the `Condition` function iterates through the submitted argument pairs and does not once evaluate to `true()`, then `altova:find-first-pair-pos` returns *No results*.

Examples

- `altova:find-first-pair-pos(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32})` returns `1`
- `altova:find-first-pair-pos(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33})` returns *No results*

Notice from the two examples above that the ordering of the pairs is: (11, 21) (12, 22) (13, 23) ... (20, 30). In the first example, the first pair causes the `Condition` function to evaluate to `true()`, and so its index position in the sequence, `1`, is returned. The second example returns *No results* because no pair gives a sum of 33.

find-first-pos [altova:]

```
altova:find-first-pos((Sequence as item()*), (Condition( Sequence-Item as xs:boolean)
as xs:integer XP3.1 XQ3.1
```

This function takes two arguments. The first argument is a sequence of one or more items of any datatype. The second argument, `Condition`, is a reference to an XPath function that takes one argument (has an arity of 1) and returns a boolean. Each item of `sequence` is submitted, in turn, to the function referenced in `Condition`. (*Remember:* This function takes a single argument.) The first `sequence` item that causes the function in `Condition` to evaluate to `true()` has its index position in `sequence` returned as the result of `altova:find-first-pos`, and the iteration stops.

Examples

- `altova:find-first-pos(5 to 10, function($a) {$a mod 2 = 0})` returns `xs:integer 2`
The `Condition` argument references the XPath 3.0 inline function, `function()`, which declares an inline function named `$a` and then defines it. Each item in the `sequence` argument of `altova:find-first-pos` is passed, in turn, to `$a` as its input value. The input value is tested on the condition in the function definition (`$a mod 2 = 0`). The index position in the sequence of the first input value to satisfy this condition is returned as the result of `altova:find-first-pos` (in this case 2, since 6, the first value (in the sequence) to satisfy the condition, is at index position 2 in the sequence).
- `altova:find-first-pos((2 to 10), (function($a) {$a+3=7}))` returns `xs:integer 3`

Further examples

If the file `C:\Temp\Customers.xml` exists:

- `altova:find-first-pos("C:\Temp\Customers.xml", "http://www.altova.com/index.html", (doc-available#1))` returns `1`

If the file `c:\Temp\Customers.xml` does not exist, and `http://www.altova.com/index.html` exists:

- `altova:find-first-pos` (`"C:\Temp\Customers.xml"`, `"http://www.altova.com/index.html"`), (`doc-available#1`) returns `2`

If the file `c:\Temp\Customers.xml` does not exist, and `http://www.altova.com/index.html` also does not exist:

- `altova:find-first-pos` (`"C:\Temp\Customers.xml"`, `"http://www.altova.com/index.html"`), (`doc-available#1`) returns no result

Notes about the examples given above

- The XPath 3.0 function, `doc-available`, takes a single string argument, which is used as a URI, and returns `true` if a document node is found at the submitted URI. (The document at the submitted URI must therefore be an XML document.)
- The `doc-available` function can be used for `condition`, the second argument of `altova:find-first-pos`, because it takes only one argument (arity=1), because it takes an `item()` as input (a string which is used as a URI), and returns a boolean value.
- Notice that the `doc-available` function is only referenced, not called. The `#1` suffix that is attached to it indicates a function with an arity of 1. In its entirety `doc-available#1` simply means: *Use the `doc-available()` function that has arity=1, passing to it as its single argument, in turn, each of the items in the first sequence.* As a result, each of the two strings will be passed to `doc-available()`, which uses the string as a URI and tests whether a document node exists at the URI. If one does, the `doc-available()` function evaluates to `true()` and the index position of that string in the sequence is returned as the result of the `altova:find-first-pos` function. *Note about the `doc-available()` function: Relative paths are resolved relative to the the current base URI, which is by default the URI of the XML document from which the function is loaded.*

▼ `for-each-attribute-pair` [`altova:`]

`altova:for-each-attribute-pair`(`Seq1 as element()?`, `Seq2 as element()?`, `Function as function()`) as `item()*` **XP3.1 XQ3.1**

The first two arguments identify two elements, the attributes of which are used to build attribute pairs, where one attribute of a pair is obtained from the first element and the other attribute is obtained from the second element. Attribute pairs are selected on the basis of having the same name, and the pairs are ordered alphabetically (on their names) into a set. If, for one attribute no corresponding attribute on the other element exists, then the pair is "disjoint", meaning that it consists of one member only. The function `item` (third argument `Function`) is applied separately to each pair in the sequence of pairs (joint and disjoint), resulting in an output that is a sequence of items.

☐ Examples

- `altova:for-each-attribute-pair`(`/Example/Test-A`, `/Example/Test-B`, `function($a, $b) {$a+$b}`) returns ...

```
(2, 4, 6) if
<Test-A att1="1" att2="2" att3="3" />
```

```
<Test-B att1="1" att2="2" att3="3" />
```

```
(2, 4, 6) if
```

```
<Test-A att2="2" att1="1" att3="3" />
```

```
<Test-B att3="3" att2="2" att1="1" />
```

```
(2, 6) if
```

```
<Test-A att4="4" att1="1" att3="3" />
```

```
<Test-B att3="3" att2="2" att1="1" />
```

Note: The result (2, 6) is obtained by way of the following action: (1+1, ()+2, 3+3, 4+()). If one of the operands is the empty sequence, as in the case of items 2 and 4, then the result of the addition is an empty sequence.

- **altova:for-each-attribute-pair**(/Example/Test-A, /Example/Test-B, concat#2) returns ...

```
(11, 22, 33) if
```

```
<Test-A att1="1" att2="2" att3="3" />
```

```
<Test-B att1="1" att2="2" att3="3" />
```

```
(11, 2, 33, 4) if
```

```
<Test-A att4="4" att1="1" att3="3" />
```

```
<Test-B att3="3" att2="2" att1="1" />
```

▼ for-each-combination [altova:]

```
altova:for-each-combination(FirstSequence as item()*, SecondSequence as item()*,  
Function($i,$j){$i || $j} ) as item()* XP3.1 XQ3.1
```

The items of the two sequences in the first two arguments are combined so that each item of the first sequence is combined, in order, once with each item of the second sequence. The function given as the third argument is applied to each combination in the resulting sequence, resulting in an output that is a sequence of items (see *example*).

▢ *Examples*

- **altova:for-each-combination**(('a', 'b', 'c'), ('1', '2', '3'), function(\$i, \$j) { \$i || \$j }) returns ('a1', 'a2', 'a3', 'b1', 'b2', 'b3', 'c1', 'c2', 'c3')

▼ for-each-matching-attribute-pair [altova:]

```
altova:for-each-matching-attribute-pair(Seq1 as element()?, Seq2 as element()?,  
Function as function()) as item()* XP3.1 XQ3.1
```

The first two arguments identify two elements, the attributes of which are used to build attribute pairs, where one attribute of a pair is obtained from the first element and the other attribute is obtained from the second element. Attribute pairs are selected on the basis of having the same name, and the pairs are ordered alphabetically (on their names) into a set. If, for one attribute no corresponding attribute on the other element exists, then no pair is built. The function item (third argument *Function*) is applied separately to each pair in the sequence of pairs, resulting in an output that is a sequence of items.

▢ *Examples*

- **altova:for-each-matching-attribute-pair**(/Example/Test-A, /Example/Test-B,

```
function($a, $b){$a+$b}) returns ...
```

```
(2, 4, 6) if
<Test-A att1="1" att2="2" att3="3" />
<Test-B att1="1" att2="2" att3="3" />
```

```
(2, 4, 6) if
<Test-A att2="2" att1="1" att3="3" />
<Test-B att3="3" att2="2" att1="1" />
```

```
(2, 6) if
<Test-A att4="4" att1="1" att3="3" />
<Test-B att3="3" att2="2" att3="1" />
```

- **altova:for-each-matching-attribute-pair**(/Example/Test-A, /Example/Test-B, concat#2) returns ...

```
(11, 22, 33) if
<Test-A att1="1" att2="2" att3="3" />
<Test-B att1="1" att2="2" att3="3" />
```

```
(11, 33) if
<Test-A att4="4" att1="1" att3="3" />
<Test-B att3="3" att2="2" att1="1" />
```

▼ substitute-empty [altova:]

```
altova:substitute-empty(FirstSequence as item()*, SecondSequence as item()) as item()*
```

XP3.1 XQ3.1

If *FirstSequence* is empty, returns *SecondSequence*. If *FirstSequence* is not empty, returns *FirstSequence*.

Examples

- **altova:substitute-empty**((1,2,3), (4,5,6)) returns (1,2,3)
- **altova:substitute-empty**((), (4,5,6)) returns (4,5,6)

18.2.1.8 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.

<i>XPath functions (used in XPath expressions in XSLT):</i>	XP1 XP2 XP3.1
<i>XSLT functions (used in XPath expressions in XSLT):</i>	XSLT1 XSLT2 XSLT3
<i>XQuery functions (used in XQuery expressions in XQuery):</i>	XQ1 XQ3.1

▼ camel-case [altova:]

altova:camel-case(*InputString* as *xs:string*) as *xs:string* **XP3.1** **XQ3.1**

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* **XP3.1** **XQ3.1**

Converts the input string *InputString* to camel case by using *SplitChars* to determine the character/s that trigger the next capitalization. *SplitChars* is used as a regular expression when *IsRegex* = `true()`, or as plain characters when *IsRegex* = `false()`. The first character in the output string is capitalized.

☐ Examples

- **altova:camel-case**("setname getname", "set|get", `true()`) returns setName getName
- **altova:camel-case**("altova\documents\testcases", "\", `false()`) returns Altova\Documents\Testcases

▼ char [altova:]

altova:char(*Position* as *xs:integer*) as *xs:string* **XP3.1** **XQ3.1**

Returns a string containing the character at the position specified by the *Position* argument, in the string obtained by converting the value of the context item to *xs:string*. The result string will be empty if no character exists at the index submitted by the *Position* argument.

☐ Examples

If the context item is 1234ABCD:

- **altova:char**(2) returns 2
- **altova:char**(5) returns A
- **altova:char**(9) returns the empty string.

- `altova:char` (-2) returns the empty string.

`altova:char`(`InputString` as `xs:string`, `Position` as `xs:integer`) as `xs:string` **XP3.1 XQ3.1**

Returns a string containing the character at the position specified by the `Position` argument, in the string submitted as the `InputString` argument. The result string will be empty if no character exists at the index submitted by the `Position` argument.

☐ Examples

- `altova:char`("2014-01-15", 5) returns -
- `altova:char`("USA", 1) returns U
- `altova:char`("USA", 10) returns the empty string.
- `altova:char`("USA", -2) returns the empty string.

▼ create-hash-from-string[altova:]

`altova:create-hash-from-string`(`InputString` as `xs:string`) as `xs:string` **XP2 XQ1 XP3.1 XQ3.1**

`altova:create-hash-from-string`(`InputString` as `xs:string`, `HashAlgo` as `xs:string`) as `xs:string` **XP2 XQ1 XP3.1 XQ3.1**

Generates a hash string from `InputString` by using the hashing algorithm specified by the `HashAlgo` argument. The following hashing algorithms may be specified (in upper or lower case): MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512. If the second argument is not specified (see the first signature above), then the SHA-256 hashing algorithm is used.

☐ Examples

- `altova:create-hash-from-string`('abc') returns a hash string generated by using the SHA-256 hashing algorithm.
- `altova:create-hash-from-string`('abc', 'md5') returns a hash string generated by using the MD5 hashing algorithm.
- `altova:create-hash-from-string`('abc', 'MD5') returns a hash string generated by using the MD5 hashing algorithm.

▼ first-chars [altova:]

`altova:first-chars`(`X-Number` as `xs:integer`) as `xs:string` **XP3.1 XQ3.1**

Returns a string containing the first `X-Number` of characters of the string obtained by converting the value of the context item to `xs:string`.

☐ Examples

If the context item is 1234ABCD:

- `altova:first-chars`(2) returns 12
- `altova:first-chars`(5) returns 1234A
- `altova:first-chars`(9) returns 1234ABCD

`altova:first-chars`(`InputString` as `xs:string`, `X-Number` as `xs:integer`) as `xs:string` **XP3.1 XQ3.1**

Returns a string containing the first `X-Number` of characters of the string submitted as the `InputString` argument.

☐ Examples

- `altova:first-chars("2014-01-15", 5)` returns `2014-`
- `altova:first-chars("USA", 1)` returns `U`

▼ `format-string` [altova:]

`altova:format-string`(*InputString* as *xs:string*, *FormatSequence* as *item()**) as *xs:string* **XP3.1 XQ3.1**

The input string (first argument) contains positional parameters (`%1`, `%2`, etc). Each parameter is replaced by the string item that is located at the corresponding position in the format sequence (submitted as the second argument). So the first item in the format sequence replaces the positional parameter `%1`, the second item replaces `%2`, and so on. The function returns this formatted string that contains the replacements. If no string exists for a positional parameter, then the positional parameter itself is returned. This happens when the index of a positional parameter is greater than the number of items in the format sequence.

▢ Examples

- `altova:format-string('Hello %1, %2, %3', ('Jane', 'John', 'Joe'))` returns `"Hello Jane, John, Joe"`
- `altova:format-string('Hello %1, %2, %3', ('Jane', 'John', 'Joe', 'Tom'))` returns `"Hello Jane, John, Joe"`
- `altova:format-string('Hello %1, %2, %4', ('Jane', 'John', 'Joe', 'Tom'))` returns `"Hello Jane, John, Tom"`
- `altova:format-string('Hello %1, %2, %4', ('Jane', 'John', 'Joe'))` returns `"Hello Jane, John, %4"`

▼ `last-chars` [altova:]

`altova:last-chars`(*X-Number* as *xs:integer*) as *xs:string* **XP3.1 XQ3.1**

Returns a string containing the last *X-Number* of characters of the string obtained by converting the value of the context item to *xs:string*.

▢ Examples

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* **XP3.1 XQ3.1**

Returns a string containing the last *X-Number* of characters of the string submitted as the *InputString* argument.

▢ Examples

- `altova:last-chars("2014-01-15", 5)` returns `01-15`
- `altova:last-chars("USA", 10)` returns `USA`

▼ `pad-string-left` [altova:]

`altova:pad-string-left`(*StringToPad* as *xs:string*, *StringLength* as *xs:integer*,

PadCharacter as *xs:string*) as *xs:string* **XP3.1 XQ3.1**

The `PadCharacter` argument is a single character. It is padded to the left of the string to increase the number of characters in `StringToPad` so that this number equals the integer value of the `StringLength` argument. The `StringLength` argument can have any integer value (positive or negative), but padding will occur only if the value of `StringLength` is greater than the number of characters in `StringToPad`. If `StringToPad` has more characters than the value of `StringLength`, then `StringToPad` is left unchanged.

▣ Examples

- `altova:pad-string-left('AP', 1, 'Z')` returns 'AP'
- `altova:pad-string-left('AP', 2, 'Z')` returns 'AP'
- `altova:pad-string-left('AP', 3, 'Z')` returns 'ZAP'
- `altova:pad-string-left('AP', 4, 'Z')` returns 'ZZAP'
- `altova:pad-string-left('AP', -3, 'Z')` returns 'AP'
- `altova:pad-string-left('AP', 3, 'YZ')` returns a pad-character-too-long error

▼ pad-string-right [altova:]

altova:pad-string-right(*StringToPad* as *xs:string*, *StringLength* as *xs:integer*, *PadCharacter* as *xs:string*) as *xs:string* **XP3.1 XQ3.1**

The `PadCharacter` argument is a single character. It is padded to the right of the string to increase the number of characters in `StringToPad` so that this number equals the integer value of the `StringLength` argument. The `StringLength` argument can have any integer value (positive or negative), but padding will occur only if the value of `StringLength` is greater than the number of characters in `StringToPad`. If `StringToPad` has more characters than the value of `StringLength`, then `StringToPad` is left unchanged.

▣ Examples

- `altova:pad-string-right('AP', 1, 'Z')` returns 'AP'
- `altova:pad-string-right('AP', 2, 'Z')` returns 'AP'
- `altova:pad-string-right('AP', 3, 'Z')` returns 'APZ'
- `altova:pad-string-right('AP', 4, 'Z')` returns 'APZZ'
- `altova:pad-string-right('AP', -3, 'Z')` returns 'AP'
- `altova:pad-string-right('AP', 3, 'YZ')` returns a pad-character-too-long error

▼ repeat-string [altova:]

altova:repeat-string(*InputString* as *xs:string*, *Repeats* as *xs:integer*) as *xs:string* **XP2 XQ1 XP3.1 XQ3.1**

Generates a string that is composed of the first `InputString` argument repeated `Repeats` number of times.

▣ Examples

- `altova:repeat-string("Altova #", 3)` returns "Altova #Altova #Altova #"

▼ substring-after-last [altova:]

altova:substring-after-last(*MainString* as *xs:string*, *CheckString* as *xs:string*) as *xs:string* **XP3.1 XQ3.1**

If `CheckString` is found in `MainString`, then the substring that occurs after `CheckString` in `MainString` is returned. If `CheckString` is not found in `MainString`, then the empty string is returned. If `CheckString` is an empty string, then `MainString` is returned in its entirety. If there is more than one occurrence of

CheckString in MainString, then the substring after the last occurrence of CheckString is returned.

▣ Examples

- `altova:substring-after-last('ABCDEFGH', 'B')` returns `'CDEFGH'`
- `altova:substring-after-last('ABCDEFGH', 'BC')` returns `'DEFGH'`
- `altova:substring-after-last('ABCDEFGH', 'BD')` returns `''`
- `altova:substring-after-last('ABCDEFGH', 'Z')` returns `''`
- `altova:substring-after-last('ABCDEFGH', '')` returns `'ABCDEFGH'`
- `altova:substring-after-last('ABCD-ABCD', 'B')` returns `'CD'`
- `altova:substring-after-last('ABCD-ABCD-ABCD', 'BCD')` returns `''`

▼ `substring-before-last` [altova:]

`altova:substring-before-last(MainString as xs:string, CheckString as xs:string) as xs:string` **XP3.1 XQ3.1**

If CheckString is found in MainString, then the substring that occurs before CheckString in MainString is returned. If CheckString is not found in MainString, or if CheckString is an empty string, then the empty string is returned. If there is more than one occurrence of CheckString in MainString, then the substring before the last occurrence of CheckString is returned.

▣ Examples

- `altova:substring-before-last('ABCDEFGH', 'B')` returns `'A'`
- `altova:substring-before-last('ABCDEFGH', 'BC')` returns `'A'`
- `altova:substring-before-last('ABCDEFGH', 'BD')` returns `''`
- `altova:substring-before-last('ABCDEFGH', 'Z')` returns `''`
- `altova:substring-before-last('ABCDEFGH', '')` returns `''`
- `altova:substring-before-last('ABCD-ABCD', 'B')` returns `'ABCD-A'`
- `altova:substring-before-last('ABCD-ABCD-ABCD', 'ABCD')` returns `'ABCD-ABCD-'`

▼ `substring-pos` [altova:]

`altova:substring-pos(StringToCheck as xs:string, StringToFind as xs:string) as xs:integer` **XP3.1 XQ3.1**

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` **XP3.1 XQ3.1**

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` **XP3.1** **XQ3.1**

This function takes an `xs:string` argument, removes any leading and trailing whitespace, and returns a "trimmed" `xs:string`.

▣ Examples

- `altova:trim-string(" Hello World ")` returns "Hello World"
- `altova:trim-string("Hello World ")` returns "Hello World"
- `altova:trim-string(" Hello World")` returns "Hello World"
- `altova:trim-string("Hello World")` returns "Hello World"
- `altova:trim-string("Hello World")` returns "Hello World"

▼ `trim-string-left` [`altova:`]

`altova:trim-string-left(InputString as xs:string) as xs:string` **XP3.1** **XQ3.1**

This function takes an `xs:string` argument, removes any leading whitespace, and returns a left-trimmed `xs:string`.

▣ Examples

- `altova:trim-string-left(" Hello World ")` returns "Hello World "
- `altova:trim-string-left("Hello World ")` returns "Hello World "
- `altova:trim-string-left(" Hello World")` returns "Hello World"
- `altova:trim-string-left("Hello World")` returns "Hello World"
- `altova:trim-string-left("Hello World")` returns "Hello World"

▼ `trim-string-right` [`altova:`]

`altova:trim-string-right(InputString as xs:string) as xs:string` **XP3.1** **XQ3.1**

This function takes an `xs:string` argument, removes any trailing whitespace, and returns a right-trimmed `xs:string`.

▣ Examples

- `altova:trim-string-right(" Hello World ")` returns " Hello World"
- `altova:trim-string-right("Hello World ")` returns "Hello World"
- `altova:trim-string-right(" Hello World")` returns " Hello World"
- `altova:trim-string-right("Hello World")` returns "Hello World"
- `altova:trim-string-right("Hello World")` returns "Hello World"

18.2.1.9 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.

<i>XPath functions (used in XPath expressions in XSLT):</i>	XP1 XP2 XP3.1
<i>XSLT functions (used in XPath expressions in XSLT):</i>	XSLT1 XSLT2 XSLT3
<i>XQuery functions (used in XQuery expressions in XQuery):</i>	XQ1 XQ3.1

▼ decode-string [altova:]

altova:decode-string(Input as *xs:base64Binary*) as *xs:string* **XP3.1 XQ3.1**

altova:decode-string(Input as *xs:base64Binary*, Encoding as *xs:string*) as *xs:string* **XP3.1 XQ3.1**

Decodes the submitted base64Binary input to a string using the specified encoding. If no encoding is specified, then the UTF-8 encoding is used. The following encodings are supported: US-ASCII, ISO-8859-1, UTF-16, UTF-16LE, UTF-16BE, ISO-10646-UCS2, UTF-32, UTF-32LE, UTF-32BE, ISO-10646-UCS4

☐ Examples

- **altova:decode-string**(\$XML1/MailData/Meta/b64B) returns the base64Binary input as a UTF-8 encoded string
- **altova:decode-string**(\$XML1/MailData/Meta/b64B, "UTF-8") returns the base64Binary input as a UTF-8-encoded string
- **altova:decode-string**(\$XML1/MailData/Meta/b64B, "ISO-8859-1") returns the base64Binary input as an ISO-8859-1-encoded string

▼ encode-string [altova:]

altova:encode-string(InputString as *xs:string*) as *xs:base64Binaryinteger* **XP3.1 XQ3.1**

altova:encode-string(InputString as *xs:string*, Encoding as *xs:string*) as *xs:base64Binaryinteger* **XP3.1 XQ3.1**

Encodes the submitted string using, if one is given, the specified encoding. If no encoding is given, then

the UTF-8 encoding is used. The encoded string is converted to base64Binary characters, and the converted base64Binary value is returned. Initially, UTF-8 encoding is supported, and support will be extended to the following encodings: US-ASCII, ISO-8859-1, UTF-16, UTF-16LE, UTF-16BE, ISO-10646-UCS2, UTF-32, UTF-32LE, UTF-32BE, ISO-10646-UCS4

▣ Examples

- `altova:encode-string("Altova")` returns the base64Binary equivalent of the UTF-8 encoded string "Altova"
- `altova:encode-string("Altova", "UTF-8")` returns the base64Binary equivalent of the UTF-8 encoded string "Altova"

▼ get-temp-folder [altova:]

`altova:get-temp-folder()` as `xs:string` XP2 XQ1 XP3.1 XQ3.1

This function takes no argument. It returns the path to the temporary folder of the current user.

▣ Examples

- `altova:get-temp-folder()` would return, on a Windows machine, something like `C:\Users\\AppData\Local\Temp` as an `xs:string`.

▼ generate-guid [altova:]

`altova:generate-guid()` as `xs:string` XP2 XQ1 XP3.1 XQ3.1

Generates a unique string GUID string.

▣ Examples

- `altova:generate-guid()` returns (for example) `85F971DA-17F3-4E4E-994E-99137873ACCD`

▼ high-res-timer [altova:]

`altova:high-res-timer()` as `xs:double` XP3.1 XQ3.1

Returns a system high-resolution timer value in seconds. A high-resolution timer, when present on a system, enables high precision time measurements when these are required (for example, in animations and for determining precise code-execution time). This function provides the resolution of the system's high-res timer.

▣ Examples

- `altova:high-res-timer()` returns something like `'1.16766146154566E6'`

▼ parse-html [altova:]

`altova:parse-html(HTMLText as xs:string) as node()` XP3.1 XQ3.1

The `HTMLText` argument is a string that contains the text of an HTML document. The function creates an HTML tree from the string. The submitted string may or may not contain the HTML element. In either case, the root element of the tree is an element named `HTML`. It is best to make sure that the HTML code in the submitted string is valid HTML.

▣ Examples

- `altova:parse-html("<html><head/><body><h1>Header</h1></body></html>")` creates an

HTML tree from the submitted string

▼ sleep[altova:]

`altova:sleep(Millisecs as xs:integer) as empty-sequence()` **XP2 XQ1 XP3.1 XQ3.1**

Suspends execution of the current operation for the number of milliseconds given by the `Millisecs` argument.

▣ Examples

- `altova:sleep(1000)` suspends execution of the current operation for 1000 milliseconds.

[[Top](#) ¹¹⁴³]

18.2.1.10 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\Common2024\jar`.

The Java libraries used are in sub-folders of the folder `<ProgramFilesFolder>\Altova\Common2024\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.

Java virtual machine

In order to be able to use the barcode functions, a Java virtual machine must be available on your machine and it must match the bit version of the Altova application (32-bit or 64-bit). The path to the machine is found as noted below.

- If you are using an Altova desktop product, the Altova application attempts to detect the path to the Java virtual machine automatically, by reading (in this order): (i) the Windows registry, and (ii) the `JAVA_HOME` environment variable. You can also add a custom path in the Options dialog of the application; this entry will take priority over any other Java VM path detected automatically.
- If you are running an Altova server product on a Windows machine, the path to the Java virtual machine will be read first from the Windows registry; if this is not successful the `JAVA_HOME` environment variable will be used.
- If you are running an Altova server product on a Linux or macOS machine, then make sure that the `JAVA_HOME` environment variable is properly set and that the Java Virtual Machines library (on Windows, the `jvm.dll` file) can be located in either the `\bin\server` or `\bin\client` directory.

XSLT example to generate barcode

Given below is an XSLT example showing how barcode functions are used in an XSLT stylesheet.

```

<?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"
  xmlns:altovaext-barcode-
property="java:com.altova.extensions.barcode.BarcodePropertyWrapper">
  <xsl:output method="html" encoding="UTF-8" indent="yes"/>
  <xsl:template match="/">
    <html>
      <head><title/></head>
      <body>
        
      </body>
    </html>
    <xsl:result-document
      href="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:template>
</xsl:stylesheet>

```

XQuery example to generate QR code

Given below is an XQuery example showing how barcode functions can be used to generate a QR code image.

```

declare variable $lines := unparsed-text-
lines('https://info.healthministry.gv.at/data/timeline-cases-provinces.csv', 'utf-8');
declare variable $main := map:merge(tokenize(head($lines), ';')!map{.:position()});
declare variable $data := map:merge(tail($lines)!array{tokenize(., ';')!map{?($main?Name):
[?($main?Date), xs:integer(?($main?ConfirmedCasesProvinces)) - xs:integer(?($main?
Recovered)]}], map{'duplicates':'combine'});
declare variable $chart_img := altovaext:chart(map{'width': 1900, 'height': 600}, map:for-
each($data, function($k, $v){[$k, $v?1!substring-before(., 'T'), $v?2][$k != 'Austria']}));

(:$main, $data,:)

```

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:

```

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()

```

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()

```

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.krysalis.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

```

18.2.2 Miscellaneous Extension Functions

There are several ready-made functions in programming languages such as Java and C# that are not available as XQuery/XPath functions or as XSLT functions. A good example would be the math functions available in Java, such as `sin()` and `cos()`. If these functions were available to the designers of XSLT stylesheets and

XQuery queries, it would increase the application area of stylesheets and queries and greatly simplify the tasks of stylesheet creators. The XSLT and XQuery engines used in a number of Altova products support the use of extension functions in [Java](#)¹¹⁴⁸ and [.NET](#)¹¹⁵⁷, as well as [MSXSL scripts for XSLT](#)¹¹⁶³. 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](#)¹¹⁴⁸
- [.NET Extension Functions](#)¹¹⁵⁷
- [MSXSL Scripts for XSLT](#)¹¹⁶³

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.

18.2.2.1 Java Extension Functions

A Java extension function can be used within an XPath or XQuery expression to invoke a Java constructor or call a Java method (static or instance).

A field in a Java class is considered to be a method without any argument. A field can be static or instance. How to access fields is described in the respective sub-sections, static and instance.

This section is organized into the following sub-sections:

- [Java: Constructors](#)¹¹⁵³
- [Java: Static Methods and Static Fields](#)¹¹⁵⁴
- [Java: Instance Methods and Instance Fields](#)¹¹⁵⁵
- [Datatypes: XPath/XQuery to Java](#)¹¹⁵⁵
- [Datatypes: Java to XPath/XQuery](#)¹¹⁵⁶

Note the following

- If you are using an Altova desktop product, the Altova application attempts to detect the path to the Java virtual machine automatically, by reading (in this order): (i) the Windows registry, and (ii) the `JAVA_HOME` environment variable. You can also add a custom path in the Options dialog of the application; this entry will take priority over any other Java VM path detected automatically.
- If you are running an Altova server product on a Windows machine, the path to the Java virtual machine will be read first from the Windows registry; if this is not successful the `JAVA_HOME` environment variable will be used.
- If you are running an Altova server product on a Linux or macOS machine, then make sure that the `JAVA_HOME` environment variable is properly set and that the Java Virtual Machines library (on Windows, the `jvm.dll` file) can be located in either the `\bin\server` or `\bin\client` directory.

Form of the extension function

The extension function in the XPath/XQuery expression must have the form `prefix:fname()`.

- The `prefix:` part identifies the extension function as a Java function. It does so by associating the extension function with an in-scope namespace declaration, the URI of which must begin with `java:` (see *below for examples*). The namespace declaration should identify a Java class, for example: `xmlns:myns="java:java.lang.Math"`. However, it could also simply be: `xmlns:myns="java"` (without a colon), with the identification of the Java class being left to the `fname()` part of the extension function.
- The `fname()` part identifies the Java method being called, and supplies the arguments for the method (see *below for examples*). However, if the namespace URI identified by the `prefix:` part does not identify a Java class (see *preceding point*), then the Java class should be identified in the `fname()` part, before the class and separated from the class by a period (see *the second XSLT example below*).

Note: The class being called must be on the classpath of the machine.

XSLT example

Here are two examples of how a static method can be called. In the first example, the class name (`java.lang.Math`) is included in the namespace URI and, therefore, must not be in the `fname()` part. In the second example, the `prefix:` part supplies the prefix `java:` while the `fname()` part identifies the class as well as the method.

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
  select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jmath="java"
  select="jmath:java.lang.Math.cos(3.14)" />
```

The method named in the extension function (`cos()` in the example above) must match the name of a public static method in the named Java class (`java.lang.Math` in the example above).

XQuery example

Here is an XQuery example similar to the XSLT example above:

```
<cosine xmlns:jMath="java:java.lang.Math">
  {jMath:cos(3.14)}
</cosine>
```

User-defined Java classes

If you have created your own Java classes, methods in these classes are called differently according to: (i) whether the classes are accessed via a JAR file or a class file, and (ii) whether these files (JAR or class) are located in the current directory (the same directory as the XSLT or XQuery document) or not. How to locate these files is described in the sections [User-Defined Class Files](#)¹¹⁵⁰ and [User-Defined Jar Files](#)¹¹⁵². 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.

18.2.2.1.1 User-Defined Class Files

If access is via a class file, then there are four 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](#)¹¹⁵⁰.)
- The class file is not packaged. The XSLT or XQuery file is in the same folder as the class file. ([See example below](#)¹¹⁵¹.)
- The class file is in a package. The XSLT or XQuery file is at some random location. ([See example below](#)¹¹⁵¹.)
- The class file is not packaged. The XSLT or XQuery file is at some random location. ([See example below](#)¹¹⁵².)

Consider the case where the class file is not packaged and is in the same folder as the XSLT or XQuery document. In this case, since all classes in the folder are found, the file location does not need to be specified. The syntax to identify a class is:

```
java:classname
```

where

`java:` indicates that a user-defined Java function is being called; (Java classes in the current directory will be loaded by default)

`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call.

Class file packaged, XSLT/XQuery file in same folder as Java package

The example below calls the `getVehicleType()` method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is also in the folder `JavaProject`.

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:car="java:com.altova.extfunc.Car" >
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
  <a>
    <xsl:value-of select="car:getVehicleType()" />
  </a>
</xsl:template>

</xsl:stylesheet>
```

Class file referenced, XSLT/XQuery file in same folder as class file

The example below calls the `getVehicleType()` method of the `Car` class. Let us say that: (i) the `Car` class file is in the following folder: `JavaProject/com/altova/extfunc`, and (ii) that this folder is the current folder in the example below. The XSLT file is also in the folder `JavaProject/com/altova/extfunc`.

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:car="java:Car" >
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
  <a>
    <xsl:value-of select="car:getVehicleType()" />
  </a>
</xsl:template>

</xsl:stylesheet>
```

Class file packaged, XSLT/XQuery file at any location

The example below calls the `getCarColor()` method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is at any location. In this case, the location of the package must be specified within the URI as a query string. The syntax is:

```
java:classname[?path=uri-of-package]
```

where

`java:` indicates that a user-defined Java function is being called
`uri-of-package` is the URI of the Java package
`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call. The example below shows how to access a class file that is located in another directory than the current directory.

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:car="java:com.altova.extfunc.Car?path=file:///C:/JavaProject/" >

<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
  <xsl:variable name="myCar" select="car:new('red') " />
  <a><xsl:value-of select="car:getCarColor($myCar)" /></a>
</xsl:template>
```

```
</xsl:stylesheet>
```

Class file referenced, XSLT/XQuery file at any location

The example below calls the `getCarColor()` method of the `Car` class. Let us say that the `Car` class file is in the folder `C:/JavaProject/com/altova/extfunc`, and the XSLT file is at any location. The location of the class file must then be specified within the namespace URI as a query string. The syntax is:

```
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`.

18.2.2.1.2 User-Defined Jar Files

If access is via a JAR file, the URI of the JAR file must be specified using the following syntax:

```
xmlns:classNS="java:classname?path=jar:uri-of-jarfile!/"
```

The method is then called by using the prefix of the namespace URI that identifies the class:
`classNS:method()`

In the above:

`java:` indicates that a Java function is being called
`classname` is the name of the user-defined class

? is the separator between the classname and the path
 path=jar: indicates that a path to a JAR file is being given
 uri-of-jarfile is the URI of the jar file
 !/ is the end delimiter of the path
 classNS:method() is the call to the method

Alternatively, the classname can be given with the method call. Here are two examples of the syntax:

```
xmlns:ns1="java:docx.layout.pages?path=jar:file:///c:/projects/docs/docx.jar!/"
ns1:main()

xmlns:ns2="java?path=jar:file:///c:/projects/docs/docx.jar!/"
ns2:docx.layout.pages.main()
```

Here is a complete XSLT example that uses a JAR file to call a Java extension function:

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:car="java?path=jar:file:///C:/test/Car1.jar!/" >
<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
  <xsl:variable name="myCar" select="car:Car1.new('red') " />
  <a><xsl:value-of select="car:Car1.getCarColor($myCar)"/></a>
</xsl:template>

<xsl:template match="car"/>

</xsl:stylesheet>
```

Note: When a path is supplied via the extension function, the path is added to the ClassLoader.

18.2.2.1.3 Java: Constructors

An extension function can be used to call a Java constructor. All constructors are called with the pseudo-function `new()`.

If the result of a Java constructor call can be [implicitly converted to XPath/XQuery datatypes](#)¹¹⁵⁶, then the Java extension function will return a sequence that is an XPath/XQuery datatype. If the result of a Java constructor call cannot be converted to a suitable XPath/XQuery datatype, then the constructor creates a wrapped Java object with a type that is the name of the class returning that Java object. For example, if a constructor for the class `java.util.Date` is called (`java.util.Date.new()`), then an object having a type `java.util.Date` is returned. The lexical format of the returned object may not match the lexical format of an XPath datatype and the value would therefore need to be converted to the lexical format of the required XPath datatype and then to the required XPath datatype.

There are two things that can be done with a Java object created by a constructor:

- It can be assigned to a variable:

```
<xsl:variable name="currentdate" select="date:new()"
xmlns:date="java:java.util.Date" />
```
- It can be passed to an extension function (see [Instance Method and Instance Fields](#)¹¹⁵⁵):

```
<xsl:value-of select="date:toString(date:new())" xmlns:date="java:java.util.Date" />
```

18.2.2.1.4 Java: Static Methods and Static Fields

A static method is called directly by its Java name and by supplying the arguments for the method. Static fields (methods that take no arguments), such as the constant-value fields `E` and `PI`, are accessed without specifying any argument.

XSLT examples

Here are some examples of how static methods and fields can be called:

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
select="jMath:cos(jMath:PI())" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
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:

```
<xsl:value-of xmlns:java="java:"
select="java:java.lang.Math.cos(3.14)" />
```

XQuery example

A similar example in XQuery would be:

```
<cosine xmlns:jMath="java:java.lang.Math">
  {jMath:cos(3.14)}
</cosine>
```

18.2.2.1.5 Java: Instance Methods and Instance Fields

An instance method has a Java object passed to it as the first argument of the method call. Such a Java object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:

```
<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`.

18.2.2.1.6 Datatypes: XPath/XQuery to Java

When a Java function is called from within an XPath/XQuery expression, the datatype of the function's arguments is important in determining which of multiple Java classes having the same name is called.

In Java, the following rules are followed:

- If there is more than one Java method with the same name, but each has a different number of arguments than the other/s, then the Java method that best matches the number of arguments in the function call is selected.
- The XPath/XQuery string, number, and boolean datatypes (see *list below*) are implicitly converted to a corresponding Java datatype. If the supplied XPath/XQuery type can be converted to more than one Java type (for example, `xs:integer`), then that Java type is selected which is declared for the selected

method. For example, if the Java method being called is `fx(decimal)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to Java's `decimal` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to Java datatypes.

<code>xs:string</code>	<code>java.lang.String</code>
<code>xs:boolean</code>	<code>boolean (primitive)</code> , <code>java.lang.Boolean</code>
<code>xs:integer</code>	<code>int</code> , <code>long</code> , <code>short</code> , <code>byte</code> , <code>float</code> , <code>double</code> , and the wrapper classes of these, such as <code>java.lang.Integer</code>
<code>xs:float</code>	<code>float (primitive)</code> , <code>java.lang.Float</code> , <code>double (primitive)</code>
<code>xs:double</code>	<code>double (primitive)</code> , <code>java.lang.Double</code>
<code>xs:decimal</code>	<code>float (primitive)</code> , <code>java.lang.Float</code> , <code>double(primitive)</code> , <code>java.lang.Double</code>

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.

18.2.2.1.7 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).

18.2.2.2 .NET Extension Functions

If you are working on the .NET platform on a Windows machine, you can use extension functions written in any of the .NET languages (for example, C#). A .NET extension function can be used within an XPath or XQuery expression to invoke a constructor, property, or method (static or instance) within a .NET class.

A property of a .NET class is called using the syntax `get_PropertyName()`.

This section is organized into the following sub-sections:

- [.NET: Constructors](#) ¹¹⁵⁹
- [.NET: Static Methods and Static Fields](#) ¹¹⁶⁰
- [.NET: Instance Methods and Instance Fields](#) ¹¹⁶⁰
- [Datatypes: XPath/XQuery to .NET](#) ¹¹⁶¹
- [Datatypes: .NET to XPath/XQuery](#) ¹¹⁶²

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:

<code>asm</code>	The name of the assembly to be loaded.
<code>ver</code>	The version number (maximum of four integers separated by periods).
<code>sn</code>	The key token of the assembly's strong name (16 hex digits).
<code>from</code>	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.
<code>partialname</code>	The partial name of the assembly. It is supplied to <code>Assembly.LoadWith.PartialName()</code> , which will attempt to load the assembly. If <code>partialname</code> 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`:

```
xmlns:myns="clitype:System.Environment"
```

An example of a namespace declaration in XSLT that identifies the class to be loaded as `Trade.Forward.Scrip`:

```
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:

```
declare namespace cs="clitype:MyManagedDLL.testClass?asm=MyManagedDLL;
ver=1.2.3.4;loc=neutral;sn=b9f091b72dccfba8";
```

2. When the assembly is loaded from the DLL (complete and partial references below):

```
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`:

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions">
  <xsl:output method="xml" omit-xml-declaration="yes" />
  <xsl:template match="/">
    <math xmlns:math="clitype:System.Math">
      <sqrt><xsl:value-of select="math:Sqrt(9)"/></sqrt>
      <pi><xsl:value-of select="math:PI()"/></pi>
      <e><xsl:value-of select="math:E()"/></e>
      <pow><xsl:value-of select="math:Pow(math:PI(), math:E())"/></pow>
    </math>
  </xsl:template>
</xsl:stylesheet>
```

The namespace declaration on the element `math` associates the prefix `math:` with the URI `clitype:System.Math`. The `clitype:` beginning of the URI indicates that what follows identifies either a system class or a loaded class. The `math:` prefix in the XPath expressions associates the extension functions with the URI (and, by extension, the class) `System.Math`. The extension functions identify methods in the class `System.Math` and supply arguments where required.

XQuery example

Here is an XQuery example fragment similar to the XSLT example above:

```
<math xmlns:math="clitype:System.Math">
  {math:Sqrt(9)}
</math>
```

As with the XSLT example above, the namespace declaration identifies the .NET class, in this case a system class. The XQuery expression identifies the method to be called and supplies the argument.

18.2.2.2.1 .NET: Constructors

An extension function can be used to call a .NET constructor. All constructors are called with the pseudo-function `new()`. If there is more than one constructor for a class, then the constructor that most closely matches the number of arguments supplied is selected. If no constructor is deemed to match the supplied argument/s, then a 'No constructor found' error is returned.

Constructors that return XPath/XQuery datatypes

If the result of a .NET constructor call can be [implicitly converted to XPath/XQuery datatypes](#)¹¹⁵⁶, then the .NET extension function will return a sequence that is an XPath/XQuery datatype.

Constructors that return .NET objects

If the result of a .NET constructor call cannot be converted to a suitable XPath/XQuery datatype, then the constructor creates a wrapped .NET object with a type that is the name of the class returning that object. For example, if a constructor for the class `System.DateTime` is called (with `System.DateTime.new()`), then an object having a type `System.DateTime` is returned.

The lexical format of the returned object may not match the lexical format of a required XPath datatype. In such cases, the returned value would need to be: (i) converted to the lexical format of the required XPath datatype; and (ii) cast to the required XPath datatype.

There are three things that can be done with a .NET object created by a constructor:

- It can be used within a variable:


```
<xsl:variable name="currentdate" select="date:new(2008, 4, 29)"
  xmlns:date="clitype:System.DateTime" />
```
- It can be passed to an extension function (see [Instance Method and Instance Fields](#)¹¹⁵⁶):


```
<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(date:get_Month(date:new(2008, 4, 29)))" xmlns:date="clitype:System.DateTime" />`

18.2.2.2 .NET: Static Methods and Static Fields

A static method is called directly by its name and by supplying the arguments for the method. The name used in the call must exactly match a public static method in the class specified. If the method name and the number of arguments that were given in the function call matches more than one method in a class, then the types of the supplied arguments are evaluated for the best match. If a match cannot be found unambiguously, an error is reported.

Note: A field in a .NET class is considered to be a method without any argument. A property is called using the syntax `get_PropertyName()`.

Examples

An XSLT example showing a call to a method with one argument (`System.Math.Sin(arg)`):

```
<xsl:value-of select="math:Sin(30)" xmlns:math="clitype:System.Math"/>
```

An XSLT example showing a call to a field (considered a method with no argument)

```
(System.Double.MaxValue());
```

```
<xsl:value-of select="double:MaxValue()" xmlns:double="clitype:System.Double"/>
```

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>
```

18.2.2.3 .NET: Instance Methods and Instance Fields

An instance method has a .NET object passed to it as the first argument of the method call. This .NET object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions">
  <xsl:output method="xml" omit-xml-declaration="yes"/>
  <xsl:template match="/">
    <xsl:variable name="releasedate"
      select="date:new(2008, 4, 29)"
```



```

        xmlns:date="clitype:System.DateTime"/>
<doc>
  <date>
    <xsl:value-of select="date:ToString(date:new(2008, 4, 29))"
      xmlns:date="clitype:System.DateTime"/>
  </date>
  <date>
    <xsl:value-of select="date:ToString($releasedate)"
      xmlns:date="clitype:System.DateTime"/>
  </date>
</doc>
</xsl:template>
</xsl:stylesheet>

```

In the example above, a `System.DateTime` constructor (`new(2008, 4, 29)`) is used to create a .NET object of type `System.DateTime`. This object is created twice, once as the value of the variable `releasedate`, a second time as the first and only argument of the `System.DateTime.ToString()` method. The instance method `System.DateTime.ToString()` is called twice, both times with the `System.DateTime` constructor (`new(2008, 4, 29)`) as its first and only argument. In one of these instances, the variable `releasedate` is used to get the .NET object.

Instance methods and instance fields

The difference between an instance method and an instance field is theoretical. In an instance method, a .NET object is directly passed as an argument; in an instance field, a parameter or variable is passed instead—though the parameter or variable may itself contain a .NET object. For example, in the example above, the variable `releasedate` contains a .NET object, and it is this variable that is passed as the argument of `ToString()` in the second `date` element constructor. Therefore, the `ToString()` instance in the first `date` element is an instance method while the second is considered to be an instance field. The result produced in both instances, however, is the same.

18.2.2.2.4 Datatypes: XPath/XQuery to .NET

When a .NET extension function is used within an XPath/XQuery expression, the datatypes of the function's arguments are important for determining which one of multiple .NET methods having the same name is called.

In .NET, the following rules are followed:

- If there is more than one method with the same name in a class, then the methods available for selection are reduced to those that have the same number of arguments as the function call.
- The XPath/XQuery string, number, and boolean datatypes (see *list below*) are implicitly converted to a corresponding .NET datatype. If the supplied XPath/XQuery type can be converted to more than one .NET type (for example, `xs:integer`), then that .NET type is selected which is declared for the selected method. For example, if the .NET method being called is `fx(double)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to .NET's `double` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to .NET datatypes.

<code>xs:string</code>	<code>StringValue, string</code>
<code>xs:boolean</code>	<code>BooleanValue, bool</code>
<code>xs:integer</code>	<code>IntegerValue, decimal, long, integer, short, byte, double, float</code>
<code>xs:float</code>	<code>FloatValue, float, double</code>
<code>xs:double</code>	<code>DoubleValue, double</code>
<code>xs:decimal</code>	<code>DecimalValue, decimal, double, float</code>

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.

18.2.2.2.5 Datatypes: .NET to XPath/XQuery

When a .NET method returns a value and the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, .NET's `decimal` datatype is converted to `xsd:decimal`.

When a .NET object or a 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).

18.2.2.3 MSXSL Scripts for XSLT

The `<msxsl:script>` element contains user-defined functions and variables that can be called from within XPath expressions in the XSLT stylesheet. The `<msxsl:script>` is a top-level element, that is, it must be a child element of `<xsl:stylesheet>` or `<xsl:transform>`.

The `<msxsl:script>` element must be in the namespace `urn:schemas-microsoft-com:xslt` (see example below).

Scripting language and namespace

The scripting language used within the block is specified in the `<msxsl:script>` element's `language` attribute and the namespace to be used for function calls from XPath expressions is identified with the `implements-prefix` attribute (see below).

```
<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 version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:msxsl="urn:schemas-microsoft-com:xslt"
  xmlns:user="http://mycompany.com/mynamespace">

  <msxsl:script language="VBScript" implements-prefix="user">
    <![CDATA[
      ' Input: A currency value: the wholesale price
```

```

' Returns: The retail price: the input value plus 20% margin,
' rounded to the nearest cent
dim a as integer = 13
Function AddMargin(WholesalePrice) as integer
    AddMargin = WholesalePrice * 1.2 + a
End Function
]]>
</msxsl:script>

<xsl:template match="/">
    <html>
        <body>
            <p>
                <b>Total Retail Price =
                $<xsl:value-of select="user:AddMargin(50)"/>
            </b>
            <br/>
            <b>Total Wholesale Price =
                $<xsl:value-of select="50"/>
            </b>
            </p>
        </body>
    </html>
</xsl:template>
</xsl:stylesheet>

```

Datatypes

The values of parameters passed into and out of the script block are limited to XPath datatypes. This restriction does not apply to data passed among functions and variables within the script block.

Assemblies

An assembly can be imported into the script by using the `msxsl:assembly` element. The assembly is identified via a name or a URI. The assembly is imported when the stylesheet is compiled. Here is a simple representation of how the `msxsl:assembly` element is to be used.

```

<msxsl:script>
    <msxsl:assembly name="myAssembly.assemblyName" />
    <msxsl:assembly href="pathToAssembly" />
    ...
</msxsl:script>

```

The assembly name can be a full name, such as:

```
"system.Math, Version=3.1.4500.1 Culture=neutral PublicKeyToken=a46b3f648229c514"
```

or a short name, such as "myAssembly.Draw".

Namespaces

Namespaces can be declared with the `msxsl:using` element. This enables assembly classes to be written in the script without their namespaces, thus saving you some tedious typing. Here is how the `msxsl:using` element is used so as to declare namespaces.

```
<msxsl:script>
  <msxsl:using namespace="myAssemblyNS.NamespaceName" />

  ...

</msxsl:script>
```

The value of the `namespace` attribute is the name of the namespace.

18.3 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](#)¹¹⁶⁶
- [MS Access](#)¹¹⁶⁷
- [MS SQL Server](#)¹¹⁶⁸
- [MySQL](#)¹¹⁶⁸
- [ODBC](#)¹¹⁶⁹
- [Oracle](#)¹¹⁷⁰
- [Sybase](#)¹¹⁷¹

18.3.1 ADO

When an XML Schema is generated from an ADO database (DB), the ADO DB datatypes are converted to XML Schema datatypes as listed in the table below.

ADO Datatype	XML Schema Datatype
adGUID	xs:ID
adChar	xs:string
adWChar	xs:string
adVarChar	xs:string
adWVarChar	xs:string
adLongVarChar	xs:string
adWLongVarChar	xs:string
adVarWChar	xs:string
adBoolean	xs:boolean
adSingle	xs:float
adDouble	xs:double
adNumeric	xs:decimal
adCurrency	xs:decimal
adDBTimeStamp	xs:dateTime
adDate	xs:date
adBinary	xs:base64Binary
adVarBinary	xs:base64Binary

adLongVarBinary	xs:base64Binary
adInteger	xs:Integer
adUnsignedInt	xs:unsignedInt
adSmallInt	xs:short
adUnsignedSmallInt	xs:unsignedShort
adBigInt	xs:long
adUnsignedBigInt	xs:unsignedLong
adTinyInt	xs:byte
adUnsignedTinyInt	xs:unsignedByte

18.3.2 MS Access

When an XML Schema is generated from an MS Access database (DB), the MS Access DB datatypes are converted to XML Schema datatypes as listed in the table below.

MS Access Datatype	XML Schema Datatype
GUID	xs:ID
char	xs:string
varchar	xs:string
memo	xs:string
bit	xs:boolean
Number (single)	xs:float
Number (double)	xs:double
Decimal	xs:decimal
Currency	xs:decimal
Date/Time	xs:dateTime
Number (Long Integer)	xs:integer
Number (Integer)	xs:short
Number (Byte)	xs:byte
OLE Object	xs:base64Binary

18.3.3 MS SQL Server

When an XML Schema is generated from an MS SQL Server database (DB), the MS SQL Server DB datatypes are converted to XML Schema datatypes as listed in the table below.

MS SQL Server Datatype	XML Schema Datatype
uniqueidentifier	xs:ID
char	xs:string
nchar	xs:string
varchar	xs:string
nvarchar	xs:string
text	xs:string
ntext	xs:string
sysname	xs:string
bit	xs:boolean
real	xs:float
float	xs:double
decimal	xs:decimal
money	xs:decimal
smallmoney	xs:decimal
datetime	xs:dateTime
smalldatetime	xs:dateTime
binary	xs:base64Binary
varbinary	xs:base64Binary
image	xs:base64Binary
integer	xs:integer
smallint	xs:short
bigint	xs:long
tinyint	xs:byte

18.3.4 MySQL

When an XML Schema is generated from a MySQL database (DB), the MySQL DB datatypes are converted to XML Schema datatypes as listed in the table below.

MySQL Datatype	XML Schema Datatype
char	xs:string
varchar	xs:string
text	xs:string
tinytext	xs:string
mediumtext	xs:string
longtext	xs:string
tinyint(1)	xs:boolean
float	xs:float
double	xs:double
decimal	xs:decimal
datetime	xs:dateTime
blob	xs:base64Binary
tinyblob	xs:base64Binary
mediumblob	xs:base64Binary
longblob	xs:base64Binary
smallint	xs:short
bigint	xs:long
tinyint	xs:byte

18.3.5 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.

ODBC Datatype	XML Schema Datatype
SQL_GUID	xs:ID
SQL_CHAR	xs:string
SQL_VARCHAR	xs:string
SQL_LONGVARCHAR	xs:string
SQL_BIT	xs:boolean
SQL_REAL	xs:float

SQL_DOUBLE	xs:double
SQL_DECIMAL	xs:decimal
SQL_TIMESTAMP	xs:dateTime
SQL_DATE	xs:date
SQL_BINARY	xs:base64Binary
SQL_VARBINARY	xs:base64Binary
SQL_LONGVARBINARY	xs:base64Binary
SQL_INTEGER	xs:integer
SQL_SMALLINT	xs:short
SQL_BIGINT	xs:long
SQL_TINYINT	xs:byte

18.3.6 Oracle

When an XML Schema is generated from an Oracle database (DB), the Oracle DB datatypes are converted to XML Schema datatypes as listed in the table below.

Oracle Datatype	XML Schema Datatype
ROWID	xs:ID
CHAR	xs:string
NCHAR	xs:string
VARCHAR2	xs:string
NVARCHAR2	xs:string
CLOB	xs:string
NCLOB	xs:string
NUMBER (with check constraint applied)*	xs:boolean
NUMBER	xs:decimal
FLOAT	xs:double
DATE	xs:dateTime
INTERVAL YEAR TO MONTH	xs:gYearMonth
BLOB	xs:base64Binary

* 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.

18.3.7 Sybase

When an XML Schema is generated from a Sybase database (DB), the Sybase DB datatypes are converted to XML Schema datatypes as listed in the table below.

Sybase Datatype	XML Schema Datatype
char	xs:string
nchar	xs:string
varchar	xs:string
nvarchar	xs:string
text	xs:string
sysname-varchar(30)	xs:string
bit	xs:boolean
real	xs:float
float	xs:float
double	xs:double
decimal	xs:decimal
money	xs:decimal
smallmoney	xs:decimal
datetime	xs:dateTime
smalldatetime	xs:dateTime
timestamp	xs:dateTime
binary<=255	xs:base64Binary
varbinary<=255	xs:base64Binary
image	xs:base64Binary
integer	xs:integer
smallint	xs:short
tinyint	xs:byte

18.4 Technical Data

This section contains information on some technical aspects of your software. This information is organized into the following sections:

- [OS and Memory Requirements](#)¹¹⁷²
- [Altova Engines](#)¹¹⁷²
- [Unicode Support](#)¹¹⁷³
- [Internet Usage](#)¹¹⁷³

18.4.1 OS and Memory Requirements

Operating System

Altova software applications are available for the following platforms:

- Windows 10, Windows 11
- Windows Server 2016 or newer

Memory

Since the software is written in C++ it does not require the overhead of a Java Runtime Environment and typically requires less memory than comparable Java-based applications. However, each document is loaded fully into memory so as to parse it completely and to improve viewing and editing speed. As a result, the memory requirement increases with the size of the document.

Memory requirements are also influenced by the unlimited Undo history. When repeatedly cutting and pasting large selections in large documents, available memory can rapidly be depleted.

18.4.2 Altova Engines

XML Validator

When opening an XML document, the application uses its built-in XML validator to check for well-formedness, to validate the document against a schema (if specified), and to build trees and infosets. The XML validator is also used to provide intelligent editing help while you edit documents and to dynamically display any validation error that may occur.

The built-in XML validator implements the Final Recommendation of the W3C's XML Schema 1.0 and 1.1 specifications. New developments recommended by the W3C's XML Schema Working Group are continuously being incorporated in the XML validator, so that Altova products give you a state-of-the-art development environment.

XSLT and XQuery Engines

Altova products use the Altova XSLT 1.0, 2.0, and 3.0 Engines and the Altova XQuery 1.0 and 3.1 Engines. If one of these engines is included in the product, then documentation about implementation-specific behavior for each engine is given in the appendices of the documentation.

Note: Altova MapForce generates code using the XSLT 1.0, 2.0 and XQuery 1.0 engines.

18.4.3 Unicode Support

Altova's XML products provide full Unicode support. To edit an XML document, you will also need a font that supports the Unicode characters being used by that document.

Please note that most fonts only contain a very specific subset of the entire Unicode range and are therefore typically targeted at the corresponding writing system. If some text appears garbled, the reason could be that the font you have selected does not contain the required glyphs. So it is useful to have a font that covers the entire Unicode range, especially when editing XML documents in different languages or writing systems. A typical Unicode font found on Windows PCs is Arial Unicode MS.

In the `/Examples` folder of your application folder you will find an XHTML file called `UnicodeUTF-8.html` that contains the following sentence in a number of different languages and writing systems:

- *When the world wants to talk, it speaks Unicode*
- *Wenn die Welt miteinander spricht, spricht sie Unicode*
- 世界的に話すなら、Unicode です。

Opening this XHTML file will give you a quick impression of Unicode's possibilities and also indicate what writing systems are supported by the fonts available on your PC.

18.4.4 Internet Usage

Altova applications will initiate Internet connections on your behalf in the following situations:

- If you click the "Request evaluation key-code" in the Registration dialog (**Help | Software Activation**), the three fields in the registration dialog box are transferred to our web server by means of a regular http (port 80) connection and the free evaluation key-code is sent back to the customer via regular SMTP e-mail.
- In some Altova products, you can open a file over the Internet (**File | Open | Switch to URL**). In this case, the document is retrieved using one of the following protocol methods and connections: HTTP (normally port 80), FTP (normally port 20/21), HTTPS (normally port 443). You could also run an HTTP server on port 8080. (In the URL dialog, specify the port after the server name and a colon.)
- If you open an XML document that refers to an XML Schema or DTD and the document is specified through a URL, the referenced schema document is also retrieved through a HTTP connection (port 80) or another protocol specified in the URL (see Point 2 above). A schema document will also be retrieved when an XML file is validated. Note that validation might happen automatically upon opening a document if you have instructed the application to do this (in the File tab of the Options dialog (**Tools | Options**)).
- In Altova applications using WSDL and SOAP, web service connections are defined by the WSDL documents.
- If you are using the **Send by Mail** command (**File | Send by Mail**) in XMLSpy, the current selection or file is sent by means of any MAPI-compliant mail program installed on the user's PC.
- As part of Software Activation and LiveUpdate as further described in the Altova Software License Agreement.

18.5 License Information

This section contains information about:

- the distribution of this software product
- software activation and license metering
- the license agreement governing the use of this product

Please read this information carefully. It is binding upon you since you agreed to these terms when you installed this software product.

To view the terms of any Altova license, go to the [Altova Legal Information page](#) at the [Altova website](#).

18.5.1 Electronic Software Distribution

This product is available through electronic software distribution, a distribution method that provides the following unique benefits:

- You can evaluate the software free-of-charge for 30 days before making a purchasing decision. (*Note: Altova MobileTogether Designer is licensed free of charge.*)
- Once you decide to buy the software, you can place your order online at the [Altova website](#) and get a fully licensed product within minutes.
- When you place an online order, you always get the latest version of our software.
- The product package includes an onscreen help system that can be accessed from within the application interface. The latest version of the user manual is available at www.altova.com in (i) HTML format for online browsing, and (ii) PDF format for download (and to print if you prefer to have the documentation on paper).

30-day evaluation period

After downloading this product, you can evaluate it for a period of up to 30 days free of charge. About 20 days into the evaluation period, the software will start to remind you that it has not yet been licensed. The reminder message will be displayed once each time you start the application. If you would like to continue using the program after the 30-day evaluation period, you must purchase a product license, which is delivered in the form of a license file containing a key code. Unlock the product by uploading the license file in the Software Activation dialog of your product.

You can purchase product licenses at <https://shop.altova.com/>.

Helping Others within Your Organization to Evaluate the Software

If you wish to distribute the evaluation version within your company network, or if you plan to use it on a PC that is not connected to the Internet, you may distribute only the installer file, provided that this file is not modified in any way. Any person who accesses the software installer that you have provided must request their own 30-day evaluation license key code and after expiration of their evaluation period, must also purchase a license in order to be able to continue using the product.

18.5.2 Software Activation and License Metering

As part of Altova's Software Activation, the software may use your internal network and Internet connection for the purpose of transmitting license-related data at the time of installation, registration, use, or update to an Altova-operated license server and validating the authenticity of the license-related data in order to protect Altova against unlicensed or illegal use of the software and to improve customer service. Activation is based on the exchange of license related data such as operating system, IP address, date/time, software version, and computer name, along with other information between your computer and an Altova license server.

Your Altova product has a built-in license metering module that further helps you avoid any unintentional violation of the End User License Agreement. Your product is licensed either as a single-user or multi-user installation, and the license-metering module makes sure that no more than the licensed number of users use the application concurrently.

This license-metering technology uses your local area network (LAN) to communicate between instances of the application running on different computers.

Single license

When the application starts up, as part of the license metering process, the software sends a short broadcast datagram to find any other instance of the product running on another computer in the same network segment. If it doesn't get any response, it will open a port for listening to other instances of the application.

Multi-user license

If more than one instance of the application is used within the same LAN, these instances will briefly communicate with each other on startup. These instances exchange key-codes in order to help you to better determine that the number of concurrent licenses purchased is not accidentally violated. This is the same kind of license metering technology that is common in the Unix world and with a number of database development tools. It allows Altova customers to purchase reasonably-priced concurrent-use multi-user licenses.

We have also designed the applications so that they send few and small network packets so as to not put a burden on your network. The TCP/IP ports (2799) used by your Altova product are officially registered with the IANA (see the [IANA Service Name Registry](#) for details) and our license-metering module is tested and proven technology.

If you are using a firewall, you may notice communications on port 2799 between the computers that are running Altova products. You are, of course, free to block such traffic between different groups in your organization, as long as you can ensure by other means, that your license agreement is not violated.

Note about certificates

Your Altova application contacts the Altova licensing server (link.altova.com) via HTTPS. For this communication, Altova uses a registered SSL certificate. If this certificate is replaced (for example, by your IT department or an external agency), then your Altova application will warn you about the connection being insecure. You could use the replacement certificate to start your Altova application, but you would be doing this at your own risk. If you see a *Non-secure connection* warning message, check the origin of the certificate and consult your IT team (who would be able to decide whether the interception and replacement of the Altova certificate should continue or not).

If your organization needs to use its own certificate (for example, to monitor communication to and from client machines), then we recommend that you install Altova's free license management software, [Altova LicenseServer](#), on your network. Under this setup, client machines can continue to use your organization's certificates, while Altova LicenseServer can be allowed to use the Altova certificate for communication with Altova.

18.5.3 Altova End-User License Agreement

- The Altova End-User License Agreement is available here: <https://www.altova.com/legal/eula>
- Altova's Privacy Policy is available here: <https://www.altova.com/privacy>

Index

■

.docx (Enterprise Edition only), 18, 36

.NET,

differences to StyleVision standalone, 672

integration of StyleVision with, 670

.NET extension functions,

constructors, 1159

datatype conversions (.NET to XPath/XQuery), 1162

datatype conversions (XPath/XQuery to .NET), 1161

for XSLT and XQuery, 1157

in XPath expressions, 685, 697

instance methods, instance fields, 1160

overview, 1157

static methods, static fields, 1160

support for, in Authentic View, 685, 697

A

Abbreviations,

used in user manual, 29

About StyleVision, 851

Activating the software, 848

Active configuration, 819

ActiveX,

integration at application level, 1005

integration at document level, 1007

integration prerequisites, 1002

ActiveX controls,

adding to the Visual Studio Toolbox, 1003

Add Active and Related Files to Project, 756

Add Active File to Project, 756

Add Altova Resource to Project, 755

Add External Folder / Web Folder to Project, 756

Add Files to Project, 755

Add name, 433

Add Project Folder to Project, 756

Add URL to Project, 755

Adding schema, 714

Additional editing procedures, 415

Additional Validation, 438

ADO,

as data connection interface, 492

setting up a connection, 498

ADO.NET,

setting up a connection, 503

Alias,

see Global Resources, 416

Aligning table cell content,

in SPSs, 798

Alternative processing, 435

Altova extensions,

chart functions (see chart functions), 1069

Altova Global Resources,

see under Global Resources, 416

Altova website, 851

Altova XML Parser,

about, 1172

AltovaRowStatus,

in DB-based SPS, 594

AltovaXML,

and FOP, 665

Ambiguity,

of content model, 128

API,

documentation, 877

overview, 878

Append,

column to table in SPS, 795

row to table in SPS, 795

Appendices, 1059

Application,

ActiveDocument, 899

Application, 899

Documents, 899

Parent, 901

Quit, 902

ASP.NET application, 479

ASPX web application, 479

Assign predefined formats,

in Quick Start tutorial, 80

Attributes entry helper,

in Authentic View, 619

Authentic Browser, 22

Authentic Desktop, 22

Authentic menu, 799

dynamic table editing, 615

markup display, 615

Authentic node properties, 433**Authentic Scripting, 650, 652**

macros in, 653

Authentic toolbar, 800**Authentic View, 632**

and SPS, 108

and standard industry schemas, 108

and Working XML File, 35

context menus, 623

description of, 35

document creation process, 108

document display, 617

editing data in an XML DB, 805

entry helpers in, 619

formatting text in, 615

in Altova products, 22

main window in, 617

markup display in, 615, 617

overview of GUI, 614

paste as XML/Text, 623

save edits, 733

SPS Tables, 631

synchronizing with new version of StyleVision, 110

tables (SPS and XML), 631

toolbar buttons for, 706

toolbar icons, 615

usage of important features, 626

usage of XML tables, 632

XML table icons, 636

XML tables, 632

Authentic XML, 613**AuthenticRange,**

AppendRow, 918

Application, 919

CanPerformAction, 919

CanPerformActionWith, 920

Close, 920

CollapsToBegin, 920

CollapsToEnd, 921

Copy, 921

Cut, 921

Delete, 922

DeleteRow, 922

DuplicateRow, 922

ExpandTo, 924

FirstTextPosition, 924

FirstXMLData, 925

FirstXMLDataOffset, 926

GetElementAttributeNames, 927

GetElementAttributeValue, 927

GetElementHierarchy, 928

GetEntityNames, 928

Goto, 929

GotoNext, 929

GotoNextCursorPosition, 930

GotoPrevious, 930

GotoPreviousCursorPosition, 931

HasElementAttribute, 931

InsertEntity, 932

InsertRow, 932

IsCopyEnabled, 933

IsCutEnabled, 933

IsDeleteEnabled, 933

IsEmpty, 934

IsEqual, 934

IsFirstRow, 934

IsInDynamicTable, 935

IsLastRow, 935

IsPasteEnabled, 935

IsTextStateApplied, 936

LastTextPosition, 936

LastXMLData, 937

LastXMLDataOffset, 937

MoveBegin, 939

MoveEnd, 939

MoveRowDown, 940

MoveRowUp, 939

Parent, 940

Paste, 940

PerformAction, 941

Select, 941

SelectNext, 942

SelectPrevious, 943

SetElementAttributeValue, 944

SetFromRange, 945

Text, 945

AuthenticView, 960

Application, 952

AsXMLString, 952

DocumentBegin, 954

DocumentEnd, 954

Event, 955

Goto, 956

IsRedoEnabled, 957

IsUndoEnabled, 957

Markup Visibility, 958

AuthenticView, 960

OnBeforeCopy, 947
OnBeforeCut, 947
OnBeforeDelete, 947
OnBeforeDrop, 948
OnBeforePaste, 948
OnMouseEvent, 949
OnSelectionChanged, 950
Parent, 958
Print, 958
Redo, 959
Selection, 959
Undo, 960
WholeDocument, 961
XMLDataRoot, 961

Auto Hide,

feature of Design Entry Helpers, 39

Auto-add Date Picker, 804**Auto-Calculations, 270**

and conditions, 285
and output escaping, 352
based on result of other Auto-Calculations, 275
command for inserting in design, 767
creating, editing, formatting, 270
example files, 276
examples, 292
formatting of date results, 463
hiding, 272
how to use, 270
in Quick Start tutorial, 86
Java and :NET functions in (Enterprise edition only), 270
moving, 270
symbol in Design View, 681
updating node with value of, 767
updating nodes in XML document with value of, 272, 275

Auto-completion in DB Queries, 607**Automated processing, 661****Auto-numbering, 334****Azure SQL, 542****B****Background Information, 1172****Barcode, 769****Barcodes, 183****Base year,**

in input formatting, 354

Batch files,

and scheduled tasks, 669

Blueprints for layout, 187**Bookmarks, 182, 339**

command for inserting in design, 775
creating and editing, 339
deleting, 339
enclosing with, 790

Bookmarks (anchors),

symbol in Design View, 681

Bookmarks in DB Queries, 607**Borders,**

of SPS tables, 797

Breaks, 407**Browser pane,**

in Database Query window, 603

Bullets and Numbering, 163, 165, 773, 811

enclosing with, 789

Buttons, 180**C****C#,**

integration of StyleVision, 1010

CALS/HTML tables, 158, 798**Carriage return key,**

see Enter key, 649

Catalog customization, 120**Catalog files, 118****Catalog mechanism overview, 118****Catalogs and environment variables, 122****Catalogs in RaptorXML, 119****CDATA sections, 125**

inserting in Authentic View, 626

Cell (of table),

split horizontally, 796
split vertically, 796

Cells,

joining in SPS tables, 796

Change To command, 197**Changing view,**

to Authentic View, 615

Character references,

and output escaping, 352

Check boxes, 175

- Class attributes,**
 - in Quick Start tutorial, 80
- Close (SPS) command, 720**
- Close Project, 754**
- Column,**
 - append to SPS table, 795
 - delete from table in SPS, 796
 - insert in SPS table, 795
- Columns,**
 - forcing breaks, 785
 - inserting, 785
- Columns (of tables),**
 - hiding in HTML output, 157
- Columns for print output, 398**
- COM-API,**
 - documentation, 877
- Combo box,**
 - in Quick Start tutorial, 90
- Combo boxes, 177**
- Command line, 661**
 - and parameters, 302
 - and scheduled tasks, 669
- Command line utility, 25**
- Command reference, 1024**
- Commands,**
 - customizing, 835
- Commenting out content, 781, 792**
- Comments in DB Queries, 607**
- Companion software,**
 - for download, 851
- Complex global template, 244**
- Component download center,**
 - at Altova web site, 851
- Composite styles, 378**
 - for Authentic View, 381
- Condition,**
 - command for inserting in design, 779
- Conditional Presence, 286**
- Conditional templates, 779**
 - see under: Conditions, 280
 - symbol in Design View, 681
- Conditions,**
 - and Auto-Calculations, 285
 - editing, 283
 - enclosing with, 790
 - for different outputs, 284
 - in Quick Start tutorial, 90
 - output-based, 284
 - setting up, 280
- Configurations,**
 - of a global resource, 416, 819
- Configurations in global resources, 431**
- Consecutive markup, 33**
- Content editing procedures, 124**
- Content model,**
 - effect of ambiguity on design, 128
- Contents,**
 - command for inserting in design, 763
- Contents placeholder,**
 - in Quick Start tutorial, 68
 - inserting node as contents, 125
- Context menus,**
 - in Authentic View, 623
- Context node,**
 - in XPath dialog, 685, 697
- Copy command, 745**
- Copyright information, 1174**
- CoreCatalog.xml, 119**
- Cover pages, 395**
- Creating new SPS document,**
 - in Quick Start tutorial, 64
- Criteria,**
 - in DB Filters, 589
- Cross references, 338**
- CSS files,**
 - managing in Design Overview sidebar, 42
- CSS styles,**
 - in Modular SPSs, 234
 - in Quick Start tutorial, 80
 - see also Styles, 54
- CSS stylesheets,**
 - also see Styles, 365
 - external stylesheets, 365
 - import precedence of external, 365
 - media applied to, 365
- Custom buttons for Authentic toolbar, 800**
- Custom dictionaries,**
 - for SPS spell-checks, 815
- CustomCatalog.xml, 119**
- Customize dialog,**
 - for customizing StyleVision, 760
- Customizing StyleVision, 835**
- Cut command, 745**

D

Database,

toolbar buttons for editing, 712

Database (Enterprise and Professional editions),

see under DB, 17

Database connection,

reusing from Global Resources, 520

setting up, 492

setup examples, 521

starting the wizard, 493

Database drivers,

overview, 495

Database Query,

Browser pane in DB Query window, 603

Connecting to DB for query, 600

creating the query, 609

Messages pane, 610

Results of, 610

Database Query window, 599

toggling view on and off, 809

Databases,

and global resources, 430

see also DB, 638

see under DB, 488

Data-entry devices, 174

menu commands for inserting, 766

symbol in Design View, 681

Datatypes,

in DB Filters, 589

Date,

formatting of, 354

Date Picker,

adding by default to date nodes, 804

and XSD datatypes, 462

command for inserting in design, 768

description of use, 462

inserting in SPS, 462

lexical format of date entries, 462

using in Authentic View, 462, 645

Dates,

and Date Picker, 768

and the Date Picker, 462, 804

changing manually, 646

examples of data manipulation with XPath 2.0, 461

formatting of, 463

how to use in SPS, 461

DB, 586, 638, 639

creating queries, 639

editing in Authentic View, 638, 643

filtering display in Authentic View, 639

generated XML data files, 490

generated XML Schema file, 490

generating XML files and HTML/PDF output, 598

navigating tables in Authentic View, 639

parameters in DB queries, 639

queries in Authentic View, 638

records to display in Authentic View, 433

schema file for, 586

selecting schema for SPS, 575

selecting schema for SPS (non-XML DBs), 575

selecting schema for SPS (XML DBs), 582

selecting working XML data for SPS, 575

selecting working XML data for SPS (non-XML DBs), 575

selecting working XML data for SPS (XML DBs), 582

work mechanism in StyleVision, 490

working with in StyleVision, 488

XML data file, 586

DB controls, 804

auto-insertion, 594

command for inserting, 766

DB Filters,

clearing, 810

creating and modifying, 589

datatypes in, 589

editing, 810

filtering data for XML file, 589

DB Parameter Defaults, 589

DB Parameters,

creating and editing, 750

usage, 589

DB Query button,

inserting in SPS, 594

DB schemas (Enterprise and Professional editions), 208

DB table navigation controls, 804

Decimals,

formatting of, 354

Default user dictionary,

for SPS spell-checks, 815

Delete,

column from table in SPS, 796

row from table in SPS, 796

table in SPS, 794

Delete command, 745**Deleting,**

a DB Filter, 589

Design elements, 708**Design Entry Helper windows,**

docking, 39

floating, 39

Design Entry Helpers,

Auto Hide, 39

description of, 39

Hide, 39

switching display on and off, 761

Design Filters,

switching on and off, 762

Design Fragment,

insert, 784

Design Fragments, 255**Design Overview,**

sidebar window, 42

Design structure, 200**Design Tree,**

and Modular SPSs, 234

see also Design Entry Helpers, 39

sidebar window, 48

Design View,

and JavaScript Editor, 33

description of, 33

display of markup, 33

symbols in SPS design, 681

Dictionaries,

for SPS spell-checks, 815

Disabled command, 781, 792**disable-output-escaping, 352****Distribution,**

of Altova's software products, 1174

Docking,

Design Entry Helper windows, 39

Document,

Application, 963

Close, 964

FullName, 964

GetPathName, 964

Name, 965

Path, 966

Save, 966

SaveAs, 966

Saved, 967

SetPathName, 973

Document element,

definition of, 26

Document elements (see Root elements), 202**Document node,**

definition of, 26

Document properties, 268**Document styles, 268****Document views,**

in GUI, 32

Documentation,

overview of, 29

Document-level,

examples of integration of XMLSpy, 1010

Documents,

Count, 975

Item, 975

opening and closing, 32

DPI, 738

and pixel-defined lengths, 409

and print output, 409

DTD,

declaring unparsed entities, 440

DTDs,

as SPS source, 203

DTDs and catalogs, 118**Dynamic (SPS) tables in Authentic View,**

usage of, 631

Dynamic content,

in Quick Start tutorial, 68

Dynamic lists, 163, 165, 773**Dynamic table,**

toolbar buttons for editing, 705

Dynamic tables, 143

and global templates, 146

difference from appended/inserted rows, 146

editing, 615

editing in Authentic View, 807

headers and footers in, 146

nested dynamic tables, 146

see also SPS tables, 146

see also Tables, 153

E

Eclipse platform,

and StyleVision, 673

Eclipse platform,

- and StyleVision Integration Package, 674
- StyleVision entry points in, 679
- StyleVision Perspective in, 676

Edit menu, 745**Edit Parameters dialog, 750****Edit Template Match command, 137****Edit XPath Expression dialog,**

- see XPath dialog, 685

Editable variables, 309**Element templates,**

- user-defined, 140

Elements,

- adding in Authentic View and SPS, 128
- user-defined, 140

Elements entry helper,

- in Authentic View, 619

Embedded images, 738**Enclose With menu, 787****Encoding,**

- for output files, 839

Encoding command, 738**Encoding of output documents, 738****End User License Agreement, 1174, 1176****Enter key,**

- effects of using, 649

Entities,

- defining in Authentic View, 626, 646, 706, 806
- inserting in Authentic View, 626
- unparsed, 440
- using as URI holders, 440

Entities entry helper,

- in Authentic View, 619

Entity references,

- and output escaping, 352

Entry helpers in Design View,

- switching display on and off, 761

Enumerations,

- in StyleVisionControl, 1057
- SPYAuthenticActions, 998
- SPYAuthenticDocumentPosition, 998
- SPYAuthenticElementKind, 998
- SPYAuthenticMarkupVisibility, 999
- SPYMouseEvent, 999
- SPYXMLDataKind, 1001

Environment variables used in catalogs, 119**Environment variables, 122****Evaluation key,**

- for your Altova software, 848

Evaluation period,

- of Altova's software products, 1174

Event, 947, 948, 949, 950**Event handlers,**

- assigning functions to, 469

Excel table content,

- copy-pasting into design, 132

Exit command, 744**Expressions,**

- in DB Filter criteria, 589

Extension functions for XSLT and XQuery, 1147**Extension Functions in .NET for XSLT and XQuery,**

- see under .NET extension functions, 1157

Extension Functions in Java for XSLT and XQuery,

- see under Java extension functions, 1148

Extension Functions in MSXSL scripts, 1163

F

FAQs on StyleVision, 851**Features,**

- of StyleVision, 18

File DSN,

- setting up, 510

File menu, 714

- command Exit, 744
- File | Close, 720
- File | Encoding, 738
- File | New, 714
- File | Open, 720
- File | Print, 743
- File | Print Preview, 743
- File | Save As, 730
- File | Save Authentic XML Data, 733
- File | Save Design, 725
- File | Save Generated Files, 734

File modification alerts,

- in Modular SPSs, 234

Files,

- open recently used, 744

Filters,

- for viewing templates selectively, 711

Filters (for DB),

- clearing, 810
- editing, 810

Filters for design templates,
switching on and off, 762

Filters on node-templates, 252

Find,
using regular expressions, 745

Find & Replace sidebar, 61

Find command, 745

Find in Design View, 61

Find Next command, 745

Firebird,
Connecting through JDBC, 521
Connecting through ODBC, 523

Floating,
Design Entry Helper Windows, 39

FO processor (Enterprise edition),
setting up, 25

FO transformations, 665

Footers,
adding in table, 795
in tables, 153

Footers and headers,
containing subtotals, 405
in paged media output, 403, 405

Foreign keys,
disable in SQLite, 518

Form controls,
menu commands for inserting, 766

Format strings,
defining for Input Formatting, 811

Formatting,
also see Presentation, 349
for tables, 153
lists, 704
nodes on insertion, 127
of numeric fields, 354
overview of procedures, 349
predefined HTML formats, 704
text alignment, 704
text properties, 704
toolbar buttons for, 704

Formatting numbers,
in Auto-Numbering, 334

Form-based designs, 187, 714

Functions,
in XPath, defined by user, 446

G

General usage procedure, 102

Generated files, 111

Global Resources, 416
changing configurations, 431
defining, 416
defining database-type, 425
defining file-type, 419
defining folder-type, 424
dialog, 818
selecting configuration via toolbar, 712
toolbar, 712
using, 427, 430, 431

Global Resources XML File, 416

Global styles,
see under Styles, 369

Global templates, 244
effect on rest-of-contents, 130
in Quick Start tutorial, 97

Global types,
in templates, 244

Graphics,
overview of use in SPS, 168
see also under Images, 168

Graphics formats,
in Authentic View, 648

Grouping, 288
group-by example (Persons.sps), 290
group-by example (Scores.sps), 292

GUI,
description of, 31
document views in, 32
Main Window of, 32
multiple documents in, 32

H

Headers,
adding in table, 795
in tables, 153

Headers and footers,
containing subtotals, 405

Headers and footers,

in paged media output, 403, 405

Help menu, 848**Hide,**

feature of Design Entry Helpers, 39

Hide markup, 33, 615, 617**Horizontal line,**

command for inserting in design, 772

in Quick Start tutorial, 75

HTML document properties, 390**HTML import, 472**

creating a new SPS, 472

generating files from SPS, 478

of HTML lists, 476

of HTML tables, 476

schema structure, 474

SPS design, 474

HTML output, 111

and image support, 170

HTML page content,

copy-pasting into design, 132

HTML tables, 158, 798**HTML to XML conversion, 472****Hyperlink,**

command for inserting in design, 776

Hyperlinks, 182, 339

and unparsed entities, 341

creating and editing, 341

enclosing with, 790

linking to bookmarks, 341

linking to external resources, 341

locating via hyperlinks, 440

removing and deleting, 341

symbol in Design View, 681

IBM DB2,

connecting through JDBC, 525

connecting through ODBC, 527

IBM DB2 for i,

connecting through JDBC, 533

connecting through ODBC, 534

IBM Informix,

connecting through JDBC, 536

IE 9,

see under Internet Explorer compatibility, 106

Image,

command for inserting in design, 770

Image embedding, 738**Image formats,**

in Authentic View, 648

Images,

accessing for output rendering, 168

and unparsed entity URIs, 168

example files, 173

in Quick Start tutorial, 75

locating via unparsed entities, 440

specifying URIs for, 168

supported types, 170

symbol in Design View, 681

Import of XSLT templates,

into SPS, 259

Initial Document Section, 395**Input fields, 175****Input formatting,**

defining format strings for, 811

of dates, 463

Insert,

column in SPS table, 795

row in SPS table, 795

Insert menu, 763

Bullets and Numbering, 773

Insert | Auto-Calculation, 767

Insert | Bookmarks, 775

Insert | Condition, 779

Insert | Contents, 763

Insert | Date Picker, 768

Insert | Design Fragment, 784

Insert | Disabled, 781

Insert | Horizontal Line, 772

Insert | Hyperlink, 776

Insert | Image, 770

Insert | Page, 785

Insert | Paragraph, 769

Insert | Rest of contents, 764

Insert | Special Paragraph, 769

Inserting design elements via the toolbar, 708**Integer,**

formatting of, 354

Integrating,

StyleVision in applications, 1002

Integration Package for Eclipse,

installing, 674

Interface,

see GUI, 31

Internet Explorer compatibility, 106**Internet usage,**

in Altova products, 1173

J**Java, 1014****Java and .NET functions (Enterprise edition only),**

in Auto-Calculations, 270

Java extension functions,

constructors, 1153

datatype conversions, Java to XPath/XQuery, 1156

datatype conversions, XPath/XQuery to Java, 1155

for XSLT and XQuery, 1148

in XPath expressions, 685, 697

instance methods, instance fields, 1155

overview, 1148

static methods, static fields, 1154

support for, in Authentic View, 685, 697

user-defined class files, 1150

user-defined JAR files, 1152

JavaScript,

see under Scripts, 467

JavaScript Editor, 467, 468

in Design View, 33

JDBC,

as data connection interface, 492

connect to Teradata, 568

setting up a connection (Windows), 513

Joining cells,

in SPS tables, 796

JRE,

for StyleVision Integration Package for Eclipse, 674

JScript,

scripting with StyleVision, 854

K**Keeps, 407****Keyboard shortcuts,**

customizing for commands, 835

Key-codes,

for your Altova software, 848

L**Layout,**

of views in the GUI, 39

Layout Box, 784**Layout Boxes, 190****Layout Container, 784****Layout Containers, 187****Layout containers and elements, 708****Layout Modules,**

steps for creating, 187

Legal information, 1174**License, 1176**

information about, 1174

License metering,

in Altova products, 1175

Licenses,

for your Altova software, 848

Line,

in Layout Containers, 784

Links,

following in Authentic View, 626

see under Hyperlinks, 182, 339

List properties, 811**Lists, 163**

enclosing with, 789

imported from HTML document, 476

in Quick Start tutorial, 90

Lists (static and dynamic), 773**Local styles,**

see under Styles, 371

Local template, 244**M****Macros,**

developing, 854, 859

Main schema, 244**Main schema (Enterprise Edition only), 44****Main template, 244**

definition of, 26

MariaDB,

MariaDB,

- connect through ODBC, 538
- connecting natively, 519

Markup,

- in Authentic View, 615, 617, 806

Markup tags in Design View, 33**Memory requirements, 1172****Menu,**

- customizing, 835

Menu bar,

- moving, 31

Messages sidebar, 60**Metadata of output HTML document, 390****Microsoft Access,**

- connecting through ADO, 498, 540

Microsoft Azure SQL, 542**Microsoft Office 2007 (Enterprise Edition only), 18, 36****Microsoft SQL Server,**

- connecting through ADO, 543
- connecting through ODBC, 545

Mixed markup, 433**MobileTogether design,**

- export to, 733

Modular SPS,

- activating and de-activating, 234
- adding the SPS module, 234
- and CSS styles, 231, 234
- and file modification alerts, 234
- and module objects, 231
- and namespace declarations, 231
- and schema sources, 231, 234
- and Scripts, 231
- and Template XML Files, 231
- and Working XML Files, 231
- creating, 234
- effect of order on precedence, 234
- example project, 238
- overview, 230
- the SPS module to add, 234
- working with, 234

Modules,

- managing in Design Overview sidebar, 42

MS Word document content,

- copy-pasting into design, 132

msxsl:script, 1163**Multiline input fields, 175****Multiple document-outputs, 785****Multiple languages examples, 346****Multiple output-documents, 261**

- and output previews, 265
- linking between, 263
- location of files, 265

MySQL,

- connecting natively, 519
- connecting through ODBC, 551

N

Named templates, 244**Namespaces,**

- adding to the SPS, 44, 103, 111, 202
- in the SPS, 44
- overview of, 48

Native connections, 519**Network Proxy, 839****New command, 714****New document templates, 261**

- and design structure, 263
- inserting, 262
- URLs of, 263

New from XSLT, 442**New Project, 754****New releases of StyleVision,**

- synchronizing with StyleVision, 110

Node,

- changing what it is created as, 197

Node-templates,

- and chaining to child templates, 252
- and global templates, 252
- and XPath filters, 252
- operations on, 252
- User-Defined, 137

Numbering nodes automatically, 334**Numbers,**

- formatting of, 354

Numeric fields,

- formatting of, 354

O

Object Locator,

- in Database Query window, 603

ODBC,

- as data connection interface, 492
- connect to MariaDB, 538
- connect to Teradata, 570
- setting up a connection, 510

ODBC Drivers,

- checking availability of, 510

Office Open XML (Enterprise Edition only), 18, 36**OLE DB,**

- as data connection interface, 492

Online Help, 839, 848**OOXML (Enterprise Edition only), 18, 36****Open,**

- recently used files, 744

Open (SPS) command, 720**Open Project, 754****OpenJDK,**

- as Java Virtual Machine, 513

Oracle database,

- connecting through JDBC, 553
- connecting through ODBC, 555

Ordering Altova software, 848**OS,**

- for Altova products, 1172

Otherwise condition branch, 280**Output encoding, 738****Output escaping, 352****Output files,**

- from DB-based <%SV-PS%>, 598
- generating, 111

Output previews, 762**Output Views,**

- description of, 36

Output-based conditions, 284**Overview,**

- of XMLSpy API, 878

P

Page, 398

- commands for design, 785
- numbering in PDF output, 398
- setting margins of for PDF output, 398
- setting size of for PDF output, 398

Page breaks, 407, 785**Page numbers (Enterprise Edition), 785****Page properties in PDF, 398****Page total (Enterprise Edition), 785****Paged media,**

- and pixel-defined lengths, 409
- designing for, 392
- headers and footers, 403
- margins, 398
- page definitions, 398
- page size, 398
- pagination, 398
- properties, 392

Paragraph,

- command for inserting in design, 769
- enclosing with, 788

Parameter defaults,

- in DB Filters, 589

Parameters, 302

- and Authentic View, 302
- and command line, 302
- creating and editing, 750
- for design fragments, 304
- for schema sources, 306
- general description, 302
- in DB Filters, 589
- in DB queries, 639
- in SPS, 302
- locating nodes in in multiple documents with, 306
- managing in Design Overview sidebar, 42
- overview of user-defined parameters, 48

Parent, 965**Parser,**

- built into Altova products, 1172

Paste,

- as Text, 626
- as XML, 626

Paste As,

- Text, 623
- XML, 623

Paste command, 745**PDF,**

- defining page properties, 398

PDF Help, 839, 848**PDF output,**

- see Paged Media, 403

PDF output (Enterprise edition), 111

- and image support, 170

Pixel-defined lengths,

- and paged media, 409

Pixels,

- and print media lengths, 738
- and screen resolution, 738

Platforms,

- for Altova products, 1172

PostgreSQL,

- connecting natively, 519
- connecting through ODBC, 559

Precedence,

- of styles, 51

Predefined format strings,

- for input formatting, 811

Predefined formats,

- command for inserting in design, 769
- on inserting a node, 127
- symbol in Design View, 681

Presentation,

- also see Formats, Formatting, 349
- overview of procedures, 349

Print command, 743**Print output,**

- see Paged media, 392

Print Preview command, 743**Problems with preview, 25****Processors,**

- for download, 851

Product features,

- listing of, 18

Progress OpenEdge database,

- connecting through JDBC, 562
- connecting through ODBC, 563

Project menu, 753

- Add Active and Related Files to Project command, 756
- Add Active File to Project command, 756
- Add Altova Resource to Project command, 755
- Add External Folder / Web Folder to Project command, 756
- Add Files to Project command, 755
- Add Project Folder to Project command, 756
- Add URL to Project command, 755
- Close command, 754
- New command, 754
- Open command, 754
- Reload command, 754
- Save command, 754

Project options, 839**Project sidebar, 59****Projects,**

- and drag-and-drop, 753

- detailed description of, 113

- using, 113

Properties,

- and property groups, 55
- defining, 55
- for nodes in Authentic View, 433
- of SPS tables, 705, 797
- see also Design Entry Helpers, 39
- sidebar window, 55

Properties Entry Helper,

- Event group, 469

Properties menu, 811

- Bullets and Numbering, 811

Properties of output documents, 268**Proxy settings, 839****PXF files, 482**

- creating, 482
- deploying, 486
- editing, 485
- saving as, 730

Q

Queries,

- for DB display in Authentic View, 639

Query,

- see under Database Query, 599

Query button,

- inserting in SPS, 594

Query Database,

- see under Database Query, 599

Query Database command, 599**Query pane,**

- in Database Query window, 607

Quick Start tutorial,

- Auto-Calculations, 86
- class attributes, 80
- combo boxes, 90
- conditions, 90
- contents placeholder, 68
- creating new SPS document, 64
- CSS styles, 80
- dynamic content, 68
- generating XSLT stylesheets, 101
- global templates, 97
- horizontal lines, 75

Quick Start tutorial,

- images, 75
- introduction, 63
- lists, 90
- predefined formats, 80
- required files, 63
- rest-of-contents, 97
- setting up new SPS document, 64
- static content, 75
- static text, 75
- testing Authentic View (Enterprise and Professional editions), 101

R**Radio buttons, 180****RaptorXML, 661**

- and FOP, 665

Recently used files, 744**Records displayed in Authentic View,**

- setting, 594

Redo command, 745**Regions in DB Queries, 607****Registering your Altova software, 848****Regular expressions,**

- find and replace using, 745

Reload Project, 754**Replace,**

- using regular expressions, 745

Replace command (Enterprise and Professional editions), 745**Replace in Design View, 61****Rest-of-contents, 130**

- and global templates, 244
- command for inserting in design, 764
- in Quick Start tutorial, 97

Restore toolbars and windows, 839**Return key,**

- see Enter key, 649

Rich Edit, 383**RichEdit,**

- commands in Authentic View, 807
- insert, 764

RichEdit toolbar, 708**Root elements, 44****Root elements (aka document elements),**

- and schema sources, 202

- selecting for schema, 202

RootCatalog.xml, 119**Row,**

- append to SPS table, 795
- delete from table in SPS, 796
- insert in SPS table, 795

Rows (of tables),

- expanding/collapsing in HTML output, 157

RTF output,

- see Paged Media, 403

RTF output (Enterprise and Professional editions), 111**RTF output (Enterprise edition),**

- and image support, 170

Running totals,

- in headers and footers, 405

S**Save,**

- Working XML File, 733

Save Authentic XML Data command, 733**Save Design command, 725****Save Generated Files command, 734****Save Project, 754****Scheduled task,**

- creating a StyleVisionBatch command as, 669
- StyleVisionBatch batch files in, 669

Schema for DB-based SPSs, 575, 582**Schema Manager,**

- CLI Help command, 220, 829
- CLI Info command, 221, 830
- CLI Initialize command, 221, 830
- CLI Install command, 222, 831
- CLI List command, 222, 831
- CLI overview, 220, 829
- CLI Reset command, 223, 832
- CLI Uninstall command, 224, 833
- CLI Update command, 225, 834
- CLI Upgrade command, 225, 834
- how to run, 214, 823
- installing a schema, 218, 827
- listing schemas by status in, 216, 825
- overview of, 211, 820
- patching a schema, 218, 827
- resetting, 219, 828

Schema Manager,

- status of schemas in, 216, 825
- uninstalling a schema, 219, 828
- upgrading a schema, 218, 827

Schema sources, 103, 714

- and root elements (document elements), 202
- changing sources, 306
- managing in Design Overview sidebar, 42
- multiple in SPS (Enterprise edition), 202
- multiple sources and locating nodes, 306
- multiple sources and XPath, 306
- overview of, 48
- selecting for SPS, 202
- sidebar window, 44

Schema Sources window,

- see also Design Entry Helpers, 39

Schema structure,

- and SPS design, 128

Schema tree options, 839**Schemas,**

- as SPS source, 203
- from DB for SPS, 208
- looking up via catalogs, 120
- user-defined, 209

Schemas and catalogs, 118**Screen resolution,**

- and pixel-defined lengths, 409

Scripting,

- in Authentic View, 650, 652, 653

Scripting Editor, 652

- overview, 854, 856

Scripts,

- and JavaScript functions, 467
- defining JavaScript functions, 468
- in the Design Tree, 467
- JavaScript functions as event handlers, 469
- overview of, 48
- using in an SPS, 467

Scripts in XSLT/XQuery,

- see under Extension functions, 1147

Scroll buttons,

- in Main Window, 32

Sections,

- and page layout, 393
- default properties for new sections, 395
- deleting, 393
- in the SPS design, 393
- Initial Document Section, 395

- inserting, 785

Select All command, 745**Select Tables dialog,**

- for DB-based SPSs, 575, 582

Setting up new SPS document,

- in Quick Start tutorial, 64

Setting up StyleVision, 25**Shortcuts,**

- customizing for keyboard, 835

Show large markup, 615, 617**Show markup, 33****Show mixed markup, 615, 617****Show small markup, 617****Show small markup, 615****Simple global template, 244****Software product license, 1176****Sorting, 296**

- example files, 298
- of groups and within groups, 288, 290, 292
- Sorting mechanism, 296
- Sort-keys, 296

Sort-keys, 296**Source files for SPS, 103****Special paragraph,**

- command for inserting in design, 769
- enclosing with, 788

Spell-checker,

- in StyleVision, 814

Spell-checker options,

- for SPSs, 815

Split Preview, 36**Split table cell,**

- horizontally, 796
- vertically, 796

SPS,

- and Authentic View (Enterprise and Professional editions), 23
- and DBs, 488
- and StyleVision, 23
- and XSLT stylesheets, 23
- closing, 720
- general description of, 23
- opening, 720
- reloading, 720

SPS and Authentic View, 108**SPS design overview, 104****SPS file structure, 200****SPS tables,**

SPS tables,

- editing dynamic tables, 615
- see also Dynamic tables, 143
- see also Static tables, 143

SPS tables in Authentic View,

- usage of, 631

SQL Azure, 542**SQL Editor,**

- creating query in, 609
- description of, 607
- in Database Query window, 607

SQL Server,

- connecting through ADO, 498
- connecting through ADO.NET, 503
- connecting via JDBC, 513

SQLite,

- connecting natively, 519
- disable foreign keys, 518
- setting up a connection (Windows), 518

Static (SPS) tables in Authentic View,

- usage of, 631

Static content,

- in Quick Start tutorial, 75

Static lists, 163, 773, 789**Static table,**

- inserting, 794
- inserting in SPS, 705
- toolbar buttons for editing, 705

Static tables, 143

- see also SPS tables, 145
- see also Tables, 153

Static text,

- and output escaping, 352
- in Quick Start tutorial, 75

Status bar, 760**Structure of SPS design, 200****Style Repository,**

- and external CSS stylesheets, 365
- and global styles, 369
- see also Design Entry Helpers, 39
- sidebar window, 51

Styles,

- and property groups, 54
- assigning CSS stylesheets to SPS, 365
- cascading order, 364
- combining several, 378
- CSS rules combined, 378
- defining, 54

- defining global styles in SPS, 369
- defining local styles, 371
- from XML data, 375
- media for assigned external stylesheets, 365
- precedence of, 51
- precedence of styles, 369
- see also Design Entry Helpers, 39
- sidebar window, 54
- terminology of, 364
- via XPath expressions, 375
- working with in StyleVision, 364

Styles of output documents, 268**Stylesheets,**

- also see under CSS stylesheets, 365
- also see under XSLT stylesheets, 365

StyleVision,

- integration, 1002
- product features, 18
- synchronizing with Authentic, 110
- user manual, 17

StyleVision in Eclipse, 673**StyleVision Integration Package, 671****StyleVision perspective in Eclipse, 676****StyleVision Plugin for Eclipse,**

- see Integration Package for Eclipse, 674

StyleVision Plugin for VS .NET,

- installing, 671

StyleVision Power Stylesheet,

- see under SPS, 17

StyleVisionBatch, 25, 661**StyleVisionCommand,**

- in StyleVisionControl, 1038

StyleVisionCommands,

- in StyleVisionControl, 1040

StyleVisionControl, 1041

- documentation of, 1002
- examples of integration at document level, 1010
- integration using C#, 1010
- object reference, 1037

StyleVisionControlDocument, 1048**StyleVisionControlPlaceholder, 1055****Subtotals,**

- in headers and footers, 405

Support for StyleVision, 851**Support options, 29****Sybase,**

- connecting through JDBC, 566

Symbols in Design View,

Symbols in Design View,

- of Auto-Calculations, 681
- of bookmarks (anchors), 681
- of conditional templates, 681
- of data-entry devices, 681
- of hyperlinks, 681
- of images, 681
- of predefined formats, 681
- of XML document content, 681
- of XML document nodes, 681

System DSN,

- setting up, 510

T**Table,**

- adding headers and footers, 795
- append column to, 795
- append row to, 795
- cell content, 794
- delete column from, 796
- delete row from, 796
- deleting in SPS, 794
- editing in Authentic View, 807
- editing properties of, 797
- headers and footers, 794
- insert column in, 795
- insert row in, 795
- inserting a static table, 794
- navigating, 794
- show/hide borders in StyleVision, 797
- vertical alignment of cell content, 798

Table menu, 794**Table of contents,**

- see under TOC, 312

Tables,

- Close button to hide columns, 157
- conditional processing in, 150
- creating, 773
- creating dynamic tables, 146
- creating static tables, 145
- editing dynamic (SPS) tables, 615
- expanding/collapsing rows, 157
- formatting, 153
- headers and footers in PDF, 153
- hiding empty columns, 157

- imported from HTML document, 476
- joining cells in, 796
- overview, 143
- styles for alternate rows, 375

Tables (SPS),

- editing of properties, 705
- toolbar buttons for editing, 705

Tables in Authentic View,

- icons for editing XML tables, 636
- usage of, 631
- using SPS (static and dynamic) tables, 631
- using XML tables, 632

Tables in Design View,

- enclosing with and removing templates, 151
- representation of, 151

Tags,

- expanding and collapsing, 751

Technical Information, 1172**Technical support for StyleVision, 851****Template,**

- changing the node match for, 197
- enclosing with, 787
- inserting, 781

Template filters, 711**Template XML File (Enterprise and Professional editions), 103**

- definition of, 26

Templates,

- enclosing table rows and columns with, 151
- removing from around table rows and columns, 151
- switching view on and off, 762
- tree of, 48

Templates for nodes,

- see Node-templates, 252

Temporary output document, 25**Teradata,**

- connect through JDBC, 568
- connect through ODBC, 570

terminate, 902**Terminology,**

- used in StyleVision, 26

Text,

- editing in Authentic View, 626
- formatting in Authentic View, 626

Text output (Enterprise and Professional editions), 111**Text references, 338****Text state icon, 800****Text State Icons, 387**

TOC,

- example, hierarchical and sequential, 331
- example, simple, 327
- marking items for inclusion, 315
- menu commands, 784
- overview of usage, 312

TOC Bookmarks, 315

- and levels, 319
- creating, 319
- enclosing with, 792
- wizard for, 319

TOC items,

- constructing, 325
- formatting, 325

TOC Levels, 315, 316

- enclosing with, 792

TOC references, 325**TOC template,**

- creating and editing, 322
- formatting, 325
- level references in, 324
- reflevels in, 324
- structuring, 324

TOCrefs,

- see under TOC references, 325

Toolbar buttons,

- adding and removing, 703

Toolbars, 702

- adding/removing icons in, 702
- Authentic toolbar, 706
- customizing, 760
- Formatting toolbar, 704
- Insert Design Elements toolbar, 708
- moving, 31
- positioning in GUI, 702
- resetting, 702
- RichEdit, 708
- Standard toolbar, 712
- switching display on and off, 760
- switching display on/off, 702
- Table toolbar, 705

Tools menu, 814**Type-based templates, 244****Types as processing units,**

- in global templates, 244

U**User-Defined Elements, 140****User-Defined XML Text Blocks, 141****Undo command, 745****Unicode support,**

- in Altova products, 1173

unparsed-entity-uri function of XSLT, 440**Updating nodes,**

- with values of Auto-Calculations, 275

Updating nodes (Enterprise and Professional editions),

- with an Auto-Calculation result, 270
- with values of Auto-Calculations, 272

URIs,

- holding in unparsed entities, 440

Usage, 102**User DSN,**

- setting up, 510

User info, 433**User Interface,**

- see GUI, 31

User manual, 839, 848**User reference, 680****User-Defined Elements, 140, 786, 793****User-defined schemas, 209****User-defined template,**

- enclosing with, 788
- inserting, 782

User-Defined Templates, 137**User-Defined Text Blocks, 140, 786****User-defined XPath functions, 446****V****Validate XML,**

- in Authentic View, 804

Validator,

- in Altova products, 1172

Value formatting, 354**Variable template, 251**

- enclosing with, 788
- inserting, 783

Variables, 302, 307

Variables, 302, 307

- assigning values via Authentic, 435
- editing in Authentic View, 309

VBScript,

- scripting with StyleVision, 854

Vertical alignment of table cell content,

- in SPSs, 798

Vertical text,

- in layout boxes, 190
- in table cells, 153

View menu, 760**Views,**

- layout of in GUI, 39

Visual Studio,

- adding the StyleVision ActiveX Controls to the toolbox, 1003

Visual Studio .Net,

- and StyleVision, 670
- and StyleVision differences, 672

VS .NET,

- and StyleVision Integration Package, 671

W

Watermarks, 411**Window menu, 847****Windows,**

- support for Altova products, 1172

Word 2007 (Enterprise Edition only), 18, 36**Word 2007+ output (Enterprise edition), 111****Word document content,**

- copy-pasting into design, 132

WordML (Enterprise Edition only), 18, 36**Working XML File, 44, 103**

- and Authentic View, 35
- and Output Views, 36
- definition of, 26
- print preview, 743
- printing, 743
- validating in Authentic View, 804

X

XML,

- inserting in design, 141

XML data,

- inserting in SPS design, 125
- merging from multiple sources, 227

XML data for DB-based SPSs, 575**XML DB,**

- loading new data row into Authentic View, 805
- loading new XML data row, 639

XML document content,

- symbol in Design View, 681

XML document nodes,

- symbol in Design View, 681

XML file,

- with data from DB, 598

XML file for DB-based SPSs, 582**XML Parser,**

- about, 1172

XML Schemas and DTDs,

- as SPS source, 203

XML tables (Enterprise and Professional editions), 143**XML tables in Authentic View,**

- icons for editing, 636
- usage of, 632

XMLData,

- AppendChild, 985
- EraseAllChildren, 986
- EraseCurrentChild, 987
- GetChild, 988
- GetChildKind, 989
- GetCurrentChild, 989
- GetFirstChild, 990
- GetNextChild, 990
- HasChildren, 992
- HasChildrenKind, 992
- InsertChild, 993
- IsSameNode, 994
- Kind, 994
- MayHaveChildren, 994
- Name, 995
- Parent, 995
- TextValue, 996

XMLSpy, 22**XMLSpy API,**

- documentation, 877
- overview, 878

XMLSpyLib, 877

- Application, 898
- AuthenticRange, 916
- AuthenticView, 946

XMLSpyLib, 877

- Document, 961
- Documents, 974
- XMLData, 984

XPath,

- locating nodes in multiple documents, 306

XPath 1.0,

- and dates, 461

XPath 2.0,

- and dates, 461

XPath dialog,

- debugging expressions in, 689
- description of, 685, 697
- testing expressions in, 686

XPath expressions,

- and styles, 375
- building in Edit XPath Expression dialog, 685, 697

XPath filter,

- on global templates, 244

XPath filters on node-templates, 252**XPath functions,**

- in XPath dialog, 685, 697
- user-defined, 446

XPath operators,

- in XPath dialog, 685, 697

XPath to selected node, 614**XPath version in SPS, 105****XQuery,**

- Extension functions, 1147

xs:date,

- and the Date Picker, 462

xs:dateTime,

- and the Date Picker, 462

XSLT,

- Extension functions, 1147
- inserting code fragment in design, 141

XSLT import, 442**XSLT stylesheet preview,**

- in Output Views, 36

XSLT Templates, 48

- importing into SPS, 259
- managing in Design Overview sidebar, 42

XSLT to SPS, 442**XSLT transformations, 665****XSLT version,**

- setting for SPS, 712

XSLT version in SPS, 105**XSLTelements,**

inserting as code in design, 140