

Altova StyleVision 2023 Basic Edition



User & Reference Manual

Altova StyleVision 2023 Basic 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: 2022

© 2016-2022 Altova GmbH

Table of Contents

1	Introduction	13
1.1	Product Features.....	14
1.2	Authentic View in Altova Products.....	17
1.3	What Is an SPS?.....	18
1.4	Setting up StyleVision.....	19
1.5	Terminology.....	20
1.6	About This Documentation.....	23
2	User Interface	25
2.1	Main Window.....	26
2.1.1	Design View.....	27
2.1.2	Output Views.....	28
2.2	Sidebars.....	30
2.2.1	Design Overview.....	32
2.2.2	Schema Tree.....	35
2.2.3	Design Tree.....	38
2.2.4	Style Repository.....	41
2.2.5	Styles	43
2.2.6	Properties.....	44
2.2.7	Messages.....	48
2.2.8	Find and Replace.....	48
3	Quick Start Tutorial	50
3.1	Creating and Setting Up a New SPS.....	51
3.2	Inserting Dynamic Content (from XML Source).....	55
3.3	Inserting Static Content.....	62
3.4	Formatting the Content.....	67
3.5	Using Auto-Calculations.....	73

3.6	Using Conditions.....	77
3.7	Using Global Templates and Rest-of-Contents.....	84
3.8	That's It!.....	88
4	Usage Overview	89
4.1	SPS and Sources.....	90
4.2	Creating the Design.....	91
4.3	XSLT and XPath Versions.....	92
4.4	Internet Explorer Compatibility.....	93
4.5	Generated Files.....	95
4.6	Catalogs in StyleVision.....	96
4.6.1	How Catalogs Work.....	96
4.6.2	Catalog Structure in StyleVision.....	97
4.6.3	Customizing Your Catalogs.....	98
4.6.4	Variables for Windows System Locations.....	100
5	SPS Content	102
5.1	Inserting XML Content as Text.....	103
5.1.1	Inserting Content with a Predefined Format.....	105
5.1.2	Rest-of-Contents.....	106
5.2	Inserting MS Word Content.....	107
5.3	Inserting MS Excel Content.....	110
5.4	User-Defined Templates.....	112
5.5	User-Defined Elements, XML Text Blocks.....	115
5.5.1	User-Defined Elements.....	115
5.5.2	User-Defined XML Text Blocks.....	116
5.6	Tables.....	118
5.6.1	Static Tables.....	120
5.6.2	Dynamic Tables.....	121
5.6.3	Conditional Processing in Tables.....	125
5.6.4	Tables in Design View.....	126
5.6.5	Table Formatting.....	128
5.6.6	Row and Column Display.....	132

5.6.7	CALS/HTML Tables.....	133
5.7	Lists.....	138
5.7.1	Static Lists.....	138
5.7.2	Dynamic Lists.....	140
5.8	Graphics.....	143
5.8.1	Images: URIs and Inline Data.....	143
5.8.2	Image Types and Output.....	145
5.8.3	Example: A Template for Images.....	147
5.9	Form Controls.....	148
5.9.1	Input Fields, Multiline Input Fields.....	149
5.9.2	Check Boxes.....	149
5.9.3	Combo Boxes.....	151
5.9.4	Radio Buttons, Buttons.....	153
5.10	Links.....	154
5.11	Barcodes.....	155
5.12	Layout Modules.....	159
5.12.1	Layout Containers.....	159
5.12.2	Layout Boxes.....	162
5.12.3	Lines	166
5.13	The Change-To Feature.....	169
6	SPS Structure	172
6.1	Schema Sources.....	174
6.1.1	DTDs and XML Schemas.....	175
6.1.2	User-Defined Schemas.....	180
6.1.3	Schema Manager.....	182
6.2	Merging XML Data from Multiple Sources.....	198
6.3	Modular SPSs.....	201
6.3.1	Available Module Objects.....	202
6.3.2	Creating a Modular SPS.....	205
6.3.3	Example: An Address Book.....	209
6.4	Templates and Design Fragments.....	215
6.4.1	Main Template.....	215
6.4.2	Global Templates.....	215

6.4.3	User-Defined Templates.....	219
6.4.4	Variable Templates.....	222
6.4.5	Node-Template Operations.....	223
6.4.6	Design Fragments.....	225
6.5	XSLT Templates.....	229
6.6	Multiple Document Output.....	231
6.6.1	Inserting a New Document Template.....	232
6.6.2	New Document Templates and Design Structure.....	233
6.6.3	URLs of New Document Templates.....	233
6.6.4	Preview Files and Output Document Files.....	235
6.6.5	Document Properties and Styles.....	238

7 Advanced Features 239

7.1	Auto-Calculations.....	240
7.1.1	Editing and Moving Auto-Calculations.....	240
7.1.2	Example: An Invoice.....	242
7.2	Conditions.....	245
7.2.1	Setting Up the Conditions.....	245
7.2.2	Editing Conditions.....	248
7.2.3	Conditions and Auto-Calculations.....	249
7.3	Grouping.....	250
7.3.1	Example: Group-By (Persons.sps).....	252
7.3.2	Example: Group-By (Scores.sps).....	254
7.4	Sorting.....	258
7.4.1	The Sorting Mechanism.....	258
7.4.2	Example: Sorting on Multiple Sort-Keys.....	260
7.5	Parameters and Variables.....	263
7.5.1	User-Declared Parameters.....	263
7.5.2	Parameters for Design Fragments.....	264
7.5.3	SPS Parameters for Sources.....	267
7.5.4	Variables.....	268
7.6	Table of Contents, Referencing, Bookmarks.....	271
7.6.1	Bookmarking Items for TOC Inclusion.....	274
7.6.2	Creating the TOC Template.....	281

7.6.3	Example: Simple TOC.....	286
7.6.4	Example: Hierarchical and Sequential TOCs.....	290
7.6.5	Auto-Numbering in the Document Body.....	293
7.6.6	Cross-referencing.....	297
7.6.7	Bookmarks and Hyperlinks.....	298
8	Presentation and Output	305
8.1	Predefined Formats.....	306
8.2	Output Escaping.....	308
8.3	Value Formatting (Formatting Numeric Datatypes).....	310
8.3.1	The Value Formatting Mechanism.....	310
8.3.2	Value Formatting Syntax.....	313
8.4	Working with CSS Styles.....	319
8.4.1	External Stylesheets.....	320
8.4.2	Global Styles.....	323
8.4.3	Local Styles.....	325
8.4.4	Setting Style Values.....	327
8.4.5	Style Properties Via XPath.....	329
8.4.6	Composite Styles.....	332
8.5	HTML Document Properties.....	335
9	Additional Functionality	337
9.1	Unparsed Entity URIs.....	338
9.2	New from XSLT, XSL-FO or FO File.....	340
9.3	User-Defined XPath Functions.....	344
9.3.1	Defining an XPath Function.....	346
9.3.2	Reusing Functions to Locate Nodes.....	349
9.3.3	Parameters in XPath Functions.....	350
9.4	Working with Dates.....	359
9.4.1	Formatting Dates.....	359
9.5	Using Scripts.....	362
9.5.1	Defining JavaScript Functions.....	363
9.5.2	Assigning Functions as Event Handlers.....	364

9.5.3	External JavaScript Files.....	365
9.6	HTML Import.....	367
9.6.1	Creating New SPS via HTML Import.....	367
9.6.2	Creating the Schema and SPS Design.....	369
9.6.3	Creating Tables and Lists as Elements/Attributes.....	371
9.6.4	Generating Output.....	373
9.7	ASPX Interface for Web Applications.....	374
9.7.1	Example: Localhost on Windows 7.....	375
9.8	PXF File: Container for SPS and Related Files.....	377
9.8.1	Creating a PXF File.....	377
9.8.2	Editing a PXF File.....	380
9.8.3	Deploying a PXF File.....	381
10	Automated Processing	383
10.1	Command Line Interface.....	384
10.1.1	StyleVision.....	384
10.1.2	StyleVision Server.....	385
10.2	Using RaptorXML.....	387
10.2.1	PDF Output.....	387
10.3	Automation with FlowForce Server.....	389
10.4	How to Automate Processing.....	391
11	Menu Commands and Reference	392
11.1	Design View Symbols.....	393
11.2	Edit XPath Expression Dialog.....	397
11.2.1	Evaluator.....	398
11.2.2	Debugger.....	401
11.2.3	Expression Builder.....	409
11.3	Toolbars.....	414
11.3.1	Format.....	416
11.3.2	Table	417
11.3.3	Insert Design Elements.....	418
11.3.4	Design Filter.....	420

11.3.5	Standard.....	421
11.4	File Menu.....	423
11.4.1	New	423
11.4.2	Open, Reload, Close, Close All.....	429
11.4.3	Save Design, Save All.....	434
11.4.4	Save As.....	439
11.4.5	Export as MobileTogether Design File.....	440
11.4.6	Save Generated Files.....	440
11.4.7	Deploy to FlowForce.....	441
11.4.8	Web Design.....	443
11.4.9	Properties.....	443
11.4.10	Print Preview, Print.....	444
11.4.11	Most Recently Used Files, Exit.....	445
11.5	Edit Menu.....	447
11.5.1	Undo, Redo, Select All.....	447
11.5.2	Find, Find Next, Replace.....	447
11.5.3	Stylesheet Parameters.....	452
11.5.4	Collapse/Expand Markup.....	453
11.6	View Menu.....	454
11.6.1	Toolbars and Status Bar.....	454
11.6.2	Design Sidebars.....	455
11.6.3	Design Filter, Zoom.....	455
11.7	Insert Menu.....	457
11.7.1	Contents.....	457
11.7.2	Rest of Contents.....	458
11.7.3	Form Controls.....	458
11.7.4	Auto-Calculation.....	459
11.7.5	Paragraph, Special Paragraph.....	460
11.7.6	Image	461
11.7.7	Horizontal Line.....	463
11.7.8	Table	463
11.7.9	Bullets and Numbering.....	464
11.7.10	Bookmark.....	466
11.7.11	Hyperlink.....	467
11.7.12	Condition, Output-Based Condition.....	468

11.7.13	Disabled.....	470
11.7.14	Template.....	470
11.7.15	User-Defined Template.....	471
11.7.16	Variable Template.....	472
11.7.17	Design Fragment.....	473
11.7.18	Layout Container, Layout Box, Line.....	473
11.7.19	Table of Contents.....	473
11.7.20	New Document.....	474
11.7.21	User-Defined Item.....	474
11.8	Enclose With Menu.....	475
11.8.1	Template.....	475
11.8.2	User-Defined Template.....	476
11.8.3	Variable Template.....	476
11.8.4	Paragraph, Special Paragraph.....	476
11.8.5	Bullets and Numbering.....	477
11.8.6	Bookmarks and Hyperlinks.....	478
11.8.7	Condition, Output-Based Condition.....	478
11.8.8	Disabled.....	480
11.8.9	TOC Bookmarks and TOC Levels.....	480
11.8.10	New Document.....	480
11.8.11	User-Defined Element.....	481
11.9	Table Menu.....	482
11.9.1	Insert Table, Delete Table.....	482
11.9.2	Add Table Headers, Footers.....	483
11.9.3	Append/Insert Row/Column.....	483
11.9.4	Delete Row, Column.....	484
11.9.5	Join Cell Left, Right, Below, Above.....	484
11.9.6	Split Cell Horizontally, Vertically.....	484
11.9.7	View Cell Bounds, Table Markup.....	485
11.9.8	Table Properties.....	485
11.9.9	Edit CALS/HTML Tables.....	486
11.9.10	Vertical Alignment of Cell Content.....	486
11.10	Properties Menu.....	487
11.10.1	Edit Bullets and Numbering.....	487
11.10.2	Predefined Value Formatting Strings.....	487

11.11	Tools Menu.....	490
11.11.1	Spelling.....	490
11.11.2	Spelling Options.....	491
11.11.3	Schema Manager.....	494
11.11.4	Customize.....	509
11.11.5	Restore Toolbars and Windows.....	513
11.11.6	Options.....	513
11.12	Window Menu.....	519
11.13	Help Menu.....	520
11.13.1	Table of Contents, Index, Search.....	520
11.13.2	Activation, Order Form, Registration, Updates.....	520
11.13.3	Other Commands.....	524

12 Appendices 525

12.1	XSLT and XQuery Engine Information.....	526
12.1.1	XSLT 1.0.....	526
12.1.2	XSLT 2.0.....	526
12.1.3	XSLT 3.0.....	528
12.1.4	XQuery 1.0.....	528
12.1.5	XQuery 3.1.....	531
12.2	XSLT and XPath/XQuery Functions.....	532
12.2.1	Altova Extension Functions.....	533
12.2.2	Miscellaneous Extension Functions.....	608
12.3	Datatypes in DB-Generated XML Schemas.....	626
12.3.1	ADO	626
12.3.2	MS Access.....	627
12.3.3	MS SQL Server.....	628
12.3.4	MySQL.....	628
12.3.5	ODBC.....	629
12.3.6	Oracle.....	630
12.3.7	Sybase.....	631
12.4	Technical Data.....	632
12.4.1	OS and Memory Requirements.....	632
12.4.2	Altova Engines.....	632

12.4.3	Unicode Support.....	633
12.4.4	Internet Usage.....	633
12.5	License Information.....	634
12.5.1	Electronic Software Distribution.....	634
12.5.2	Software Activation and License Metering.....	635
12.5.3	Altova End-User License Agreement.....	636

Index

637

1 Introduction

[Altova StyleVision 2023 Basic Edition](#) is an application for graphically designing and editing StyleVision Power Stylesheets. StyleVision® runs on Windows 7 SP1 with Platform Update, Windows 8, Windows 10, Windows 11, and Windows Server 2008 R2 SP1 with Platform Update 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 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.
- To generate, directly from within StyleVision, **HTML output** from an XML document.

Last updated: 7 October 2022

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.

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:

-
- HTML documents can be [converted to XML](#) ³⁶⁷.

Interface

Given below are some general GUI features:

- [Multiple SPS designs](#) ²⁵ can be open simultaneously, with one being active at any given time. Each SPS design is shown in a separate tab.
- [Template filters](#) ⁴²⁰ 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](#) ²⁷: Markup tags in Design View can be hidden and collapsed, thus freeing up space in Design View.
- While designing the SPS, [output views](#) ²⁸ and stylesheets can be displayed by clicking the respective tabs. This enables you to quickly preview the output and the XSLT code.

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](#) ⁹² are supported.
- In the Enterprise and Professional Editions, [multiple output formats](#) ⁹⁵ (HTML) are generated from a single SPS design.
- Both [XSLT files and output files](#) ⁹⁵ can be [generated and saved](#) ⁹⁵, either directly from within the GUI or via [StyleVision Server](#).
- Altova has developed a special [PXF File format](#) ³⁷⁷ 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](#) ³⁷⁴: 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](#)¹⁰³, which you enter in the SPS, and [dynamic text](#)¹⁰³, which is selected from the [source document](#)¹⁷⁴.
- [Dynamic content](#)¹⁰³ is inserted in the design by dragging-and-dropping nodes from the [schema source](#)¹⁷⁴. 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](#)¹⁰³ can be inserted as text, or in the form of a [data-entry device](#)¹⁴⁸ (such as an [input field](#)¹⁴⁹ or [combo box](#)¹⁵¹).
- The [structure of the design](#)¹⁷² is specified and controlled in a single [main template](#)²¹. This structure can be modified by optional templates for individual elements—known as [global templates](#)²¹⁵ because they can be applied globally for that element.
- [Global templates](#)²¹⁵ can also be created for individual datatypes, thus enabling processing to be handled also on the basis of types.
- [Multiple Document Output](#)²³¹: 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](#)¹¹²: 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](#)¹¹⁵: 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](#)¹¹⁵: 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](#)²²⁵ enable the modularization and re-use of templates within an SPS, and also across multiple SPSs (see [modular SPSs](#)²⁰¹), in a manner similar to the way functions are used.
- [SPS modules](#)²⁰¹ 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](#)²²⁹: 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](#)³⁴⁰: 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](#)³⁴⁴: The user can define XPath functions which can be used anywhere in the document where XPath functions may be used.
- [Layout Containers](#)¹⁵⁹: A Layout Container is a block in which Design Elements can be laid out and absolutely positioned within the block.
- [Blueprints](#)¹⁶²: 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](#)¹⁰³ is defined once and applied to each relevant node in turn (the employee node in our example).
- Multiple [tables of contents](#)²⁷¹ can be inserted in XSLT 2.0 and 3.0 SPSs.

- Repeating data structures can also be inserted as [dynamic tables](#)¹¹⁸. 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](#)²⁵⁸ you select, and the sorted element set is sent to the output (HTML).
- [Variables](#)²⁶⁸: 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](#)²⁵⁰ on the basis of common data content (for example, the common value of an attribute value) and their positions.
- The [conditional templates](#)²⁴⁵ 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](#)²⁴⁰ 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.
- [Images](#)¹⁴³ 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](#)¹¹⁸: 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](#)¹³⁸ can be created: static and dynamic. In a [static list](#)¹³⁸, each list item is defined in the SPS. In a [dynamic list](#)¹⁴⁰, 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](#)²⁹⁸ 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](#)²⁹⁸ can be inserted. These serve as anchors that can be linked to with a hyperlink.
- [Parameters](#)²⁶³ 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](#)³¹⁰ feature, the contents of numeric XML Schema datatype nodes can be formatted as required for output display. Input Formatting can also be used to format the result of an [Auto-Calculation](#)²⁴⁰.
- [JavaScript functions](#)³⁶² can be used in the SPS to provide user-defined functionality for Authentic View and HTML output.
- A number of [predefined HTML formats](#)³⁰⁶ 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](#)³²⁵.
- Additionally, CSS styles can be defined for HTML selectors at the [global level](#)³²³ of an SPS and in external CSS stylesheets. These style rules will be applied to HTML output, thus providing considerable formatting and layout flexibility.
- [Styles can also be assigned using XPath expressions](#)³²⁹. This enables style property values to be selected from XML documents and to set property values conditionally.

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

1.3 What Is an SPS?

A StyleVision Power Stylesheet (or SPS) is an extended XSLT stylesheet which is used to graphically create a design for an HTML output document

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). The design of the SPS is flexible. It can contain dynamic and static content. The [dynamic content](#)²¹ is the data in one XML document. The [static content](#)²² 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 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](#)⁹⁵.

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](#)²⁸ 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](#)⁴⁴⁰ 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 7 SP1 with Platform Update, Windows 8, 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 HTML Preview and Design View. Internet Explorer 6.0 and later has better XML support and is recommended. For [copy-paste from Word documents](#)¹⁰⁷ (and of content that can be pasted into Word documents, such as Excel tables and HTML page content) Word 2007+ is required

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?](#)¹⁸.
- 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](#)²⁰ (*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](#)²⁰ (*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](#)²¹ 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](#)²⁰. 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](#)²⁰.
- 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](#)²⁰ in expressions that do not have sub-expressions. But where there are sub-expressions, the [context node](#)²⁰ 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](#)²¹, 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](#)²⁰, 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](#)²¹ can be defined only for global elements.

global template

A global template may be defined for a [global element](#)²¹. 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](#)²¹.

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](#)²¹. The local template applies to that particular occurrence of the element in the [main template](#)²¹. Instead of the local template, a [global template](#)²¹ can be applied to a given occurrence of an element in the [main template](#)²¹.

main schema

One of the assigned schema sources is designated the main schema; the document node of the [Working XML File](#)²² associated with the main schema is used as the starting point for the [main template](#)²¹.

main template

The main entry-point template. In StyleVision, this template matches the [document element](#)²¹ 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](#)²¹. The [main template](#)²¹ defines the basic output document structure and defines how the input document/s are to be processed. It can contain [local templates](#)²¹ and can reference [global templates](#)²¹.

- output** The output produced by processing an XML document with an XSLT stylesheet. Output files that can be generated by StyleVision would be HTML format. 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](#)²⁴⁰ or a text string; or (iii) a [predefined format](#)³⁰⁶ (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 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](#)²¹). 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. 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](#)²⁵, which provides an overview of the StyleVision GUI.
- A [tutorial](#)⁵⁰ section, which is a hands-on exercise to familiarize you with StyleVision features.
- [Usage Overview](#)⁸⁹, 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](#)¹⁰², which explains how static (stylesheet-originated) and dynamic (XML document-originated) components are created and edited in the SPS.
- [SPS File Structure](#)¹⁷², which shows how an SPS file can be structured and modularized, and describes the handling of StyleVision's templates.
- [SPS File Advanced Features](#)²³⁹, 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](#)³⁰⁵, which explains how SPS components are formatted and laid out.
- [SPS File Additional Editing Functionality](#)³³⁷, 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.
- A [reference](#)³⁹² section containing descriptions of all symbols and commands used in StyleVision.
- [Appendices](#)⁵²⁵ containing information about the Altova XSLT Engine information; technical data about StyleVision; and license information.

How to use

We suggest you read the Introduction, [User Interface](#)²⁵ and [Usage Overview](#)⁸⁹ sections first in order to get an overview of StyleVision features and general usage. Doing the [tutorial](#)⁵⁰ next would provide hands-on experience of creating an SPS. The SPS File sections ([SPS File Content](#)¹⁰², [SPS File Structure](#)¹⁷², [SPS File Advanced Features](#)²³⁹, [SPS File Presentation](#)³⁰⁵, [SPS File Additional Functionality](#)³³⁷) provide detailed descriptions of how to use various StyleVision features. For subsequent reference, the [Reference](#)³⁹² section provides a concise description of all toolbar icon, design symbols, and menu commands, organized according to toolbar and menu.

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 2008 R2 SP1 with Platform Update or newer.

Support options

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

1. Check the [Help](#) ¹³ 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

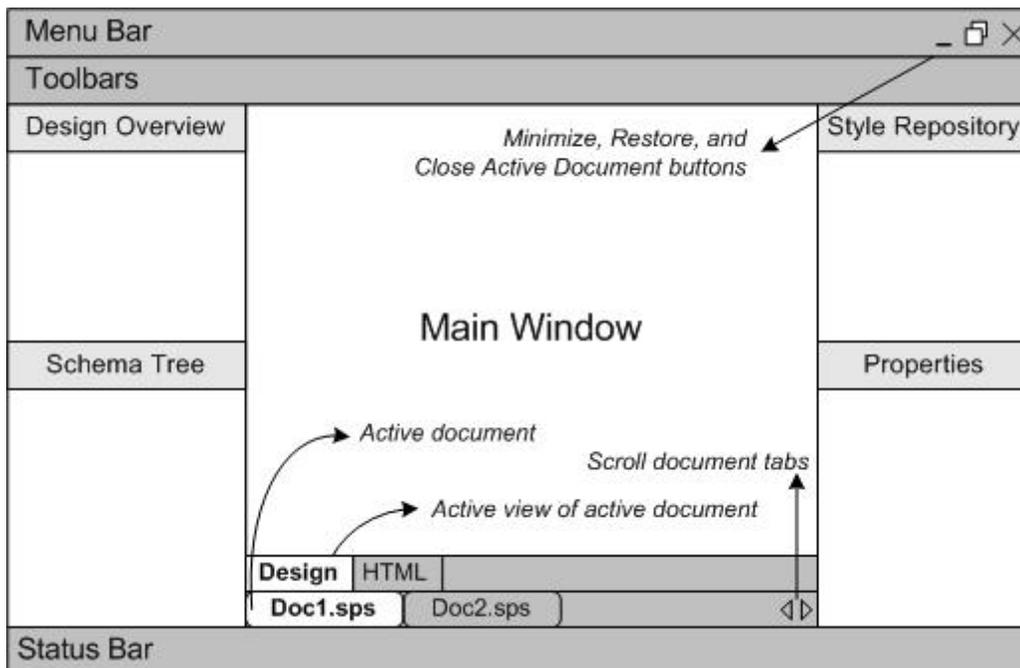
The following abbreviations are used frequently in this documentation:

- **SPS:** StyleVision Power Stylesheet
- **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](#)³⁹² section. The menu bar also contains the Minimize, Restore, and Close Active Document buttons.
- A **toolbar area**. The various [toolbars](#)⁴¹⁴ and the command shortcuts in each toolbar are described in the [User Reference](#)³⁹² section.
- A tabbed [Main Window](#)²⁶, which displays one or more open SPS documents at a time. In this window, you can [edit the design of the SPS](#)²⁷ and [preview the XSLT stylesheets and output](#)²⁸.
- The [Design Sidebars](#)³⁰—the [Design Overview](#)³², [Schema Tree](#)³⁵, [Design Tree](#)³⁸, [Style Repository](#)⁴¹, [Styles](#)⁴³, [Properties](#)⁴⁴ windows—which can be docked within the application GUI or made to float on the screen.
- A **status bar**, which displays application status information.

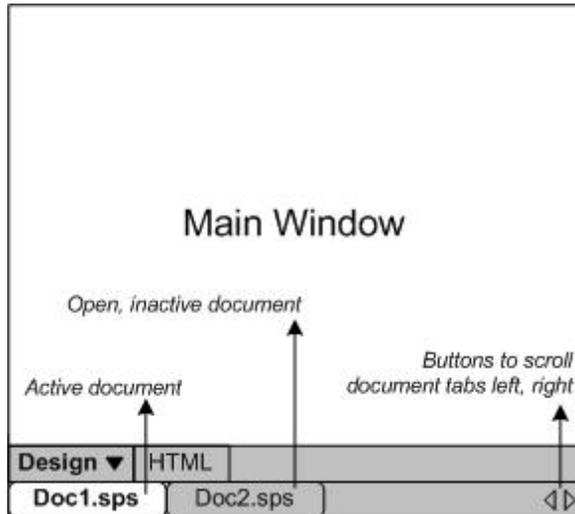


The [Main Window](#)²⁶ and [Design sidebars](#)³⁰ are described in more detail in the sub-sections of this section.

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, 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](#) ⁴²⁹).

Document views

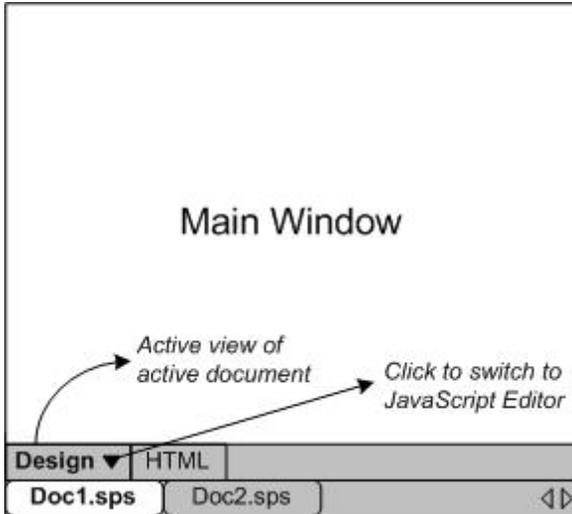
A document is displayed in the following views, one of which can be active at a time:

- [Design View](#) ²⁷, 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.
- [Output Views](#) ²⁸ (HTML, Text, 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.

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.

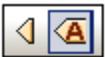


Design View can also be switched to a [JavaScript Editor](#)³⁶³. In the JavaScript Editor you can create and edit [JavaScript functions](#)³⁶² which then become available in the GUI for use in the SPS. To switch to the [JavaScript Editor](#)³⁶³, click the dropdown button in the Design tab (*see illustration*) and select JavaScript from the dropdown menu. To switch back to Design View, click the dropdown button in the JavaScript tab and select Design from the dropdown menu.

In Design View, the SPS can have several templates: the main template, global templates, and Design Fragments. You can control which of these template types is displayed in Design View by using [Template Display Filters](#)⁴²⁰, which are available as [toolbar icons](#)⁴²⁰. 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*).

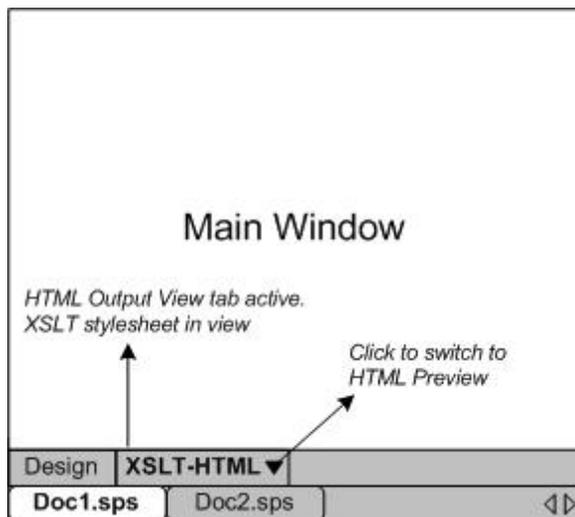


The icons shown above are toggles. They are, from left: (i) Show small design markups (tags without names); and (ii) Show large design markups (tags with names). When small markup is switched on, the path to a node is displayed when you mouseover that node.

2.1.2 Output Views

The **Output View** tab (*illustration below*) displays: (i) the XSLT-for-HTML stylesheet generated from the SPS design; and (ii) a preview of the HTML output, produced by transforming the [Working XML File](#)²² with the generated XSLT stylesheet.

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



XSLT view

The XSLT view displays the XSLT-for-HTML generated 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](#)⁴⁴⁷) and text from it can be copied to the clipboard (with [Edit | Copy](#)⁴⁴⁷).

Note: The XSLT stylesheets generated from the SPS can be separately generated and saved using the [File | Save Generated Files](#)⁴⁴⁰ command.

HTML preview

HTML preview displays the output produced by transforming the [Working XML File](#)²² with the XSLT-for-HTML. The output is generated afresh each time the HTML 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.

If no [Working XML File](#)²² is assigned when HTML preview is selected in the HTML 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](#)²² 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](#)²².

Note: The output files generated from the SPS can be separately generated and saved using the [File | Save Generated Files](#)⁴⁴⁰ command.

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 subsection of this section.

- [Design Overview](#) ³²
- [Schema Tree](#) ³⁵
- [Design Tree](#) ³⁸
- [Style Repository](#) ⁴¹
- [Styles](#) ⁴³
- [Properties](#) ⁴⁴
- [Messages](#) ⁴⁸
- [Find and Replace](#) ⁴⁸

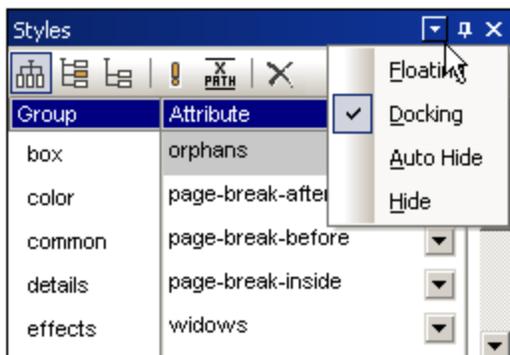
Layout of the views

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. The view categories are: (i) no document open; (ii) Design View; (iii) Output 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.

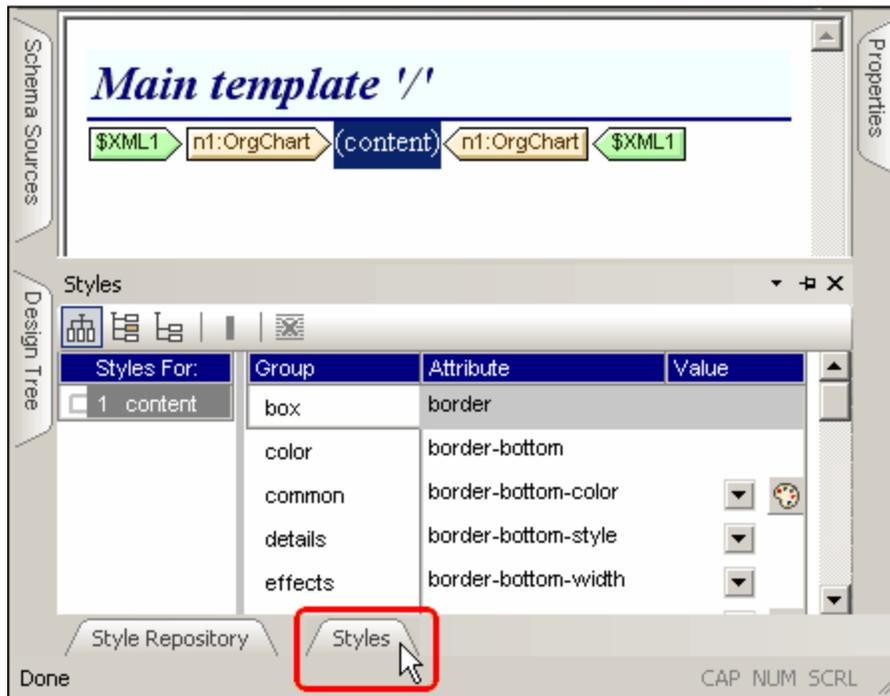
Alternatively, you can also use the following mechanisms. To float a docked window, click the **Menu** button at the top-right of a docked window (*see screenshot below*) and select Floating. This menu can also be accessed by right-clicking the title bar of the docked window.



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. 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.

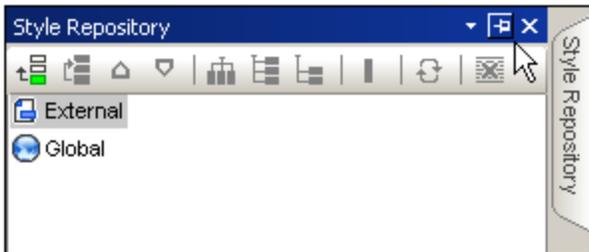
To auto-hide a window, in a docked window, click the Auto Hide button (the drawing pin icon) at the top right of the window (*screenshot below*). Alternatively, in the [Menu](#)³¹, select Auto Hide; (to display the [Menu](#)³¹, right-click the title bar of the window or click the [Menu button](#)³¹ in 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](#)³¹, deselect Auto Hide;

(to display the [Menu](#)³¹, right-click the title bar of the window or click the [Menu button](#)³¹ 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

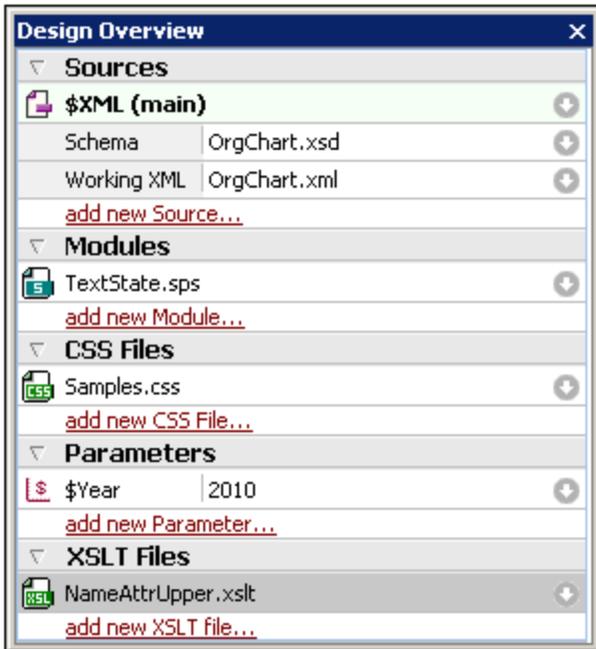
When a sidebar window is hidden it is no longer visible in the GUI, in either its maximized form (docked or floating) or in its minimized form (as a tab at an edge of the GUI, which is done using the [Auto-Hide feature](#)³¹).

To hide a window, click the **Close** button at the top right of a docked or floating window. Alternatively, in the [Menu](#)³¹, select **Hide**; (to display the [Menu](#)³¹, right-click the title bar of the window or click the [Menu button](#)³¹ 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](#)⁴⁵⁵ menu. The Design sidebar window is made visible in the position at which it was (docked or floating) when it was hidden.

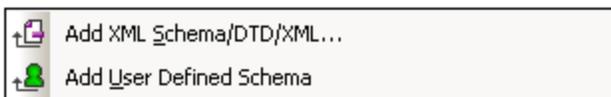
2.2.1 Design Overview

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 an XML Schema, DTD, schema generated from an XML file, or a user-defined schema.



The Working XML File

When a schema is added, it is listed under the Sources item. Each schema has an entry for the [Working XML File](#) ²² within the XML item.

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](#) ²⁰⁵ 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 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](#) ²⁰¹ 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](#) ²⁰⁵ 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](#) ²⁰² (objects inside the modules), however, are listed in the [Design Tree sidebar](#) ³⁸.

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](#) ³²⁰ 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](#) ³²⁰ inside the CSS files, however, are listed in the [Style Repository sidebar](#) ⁴¹.

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.

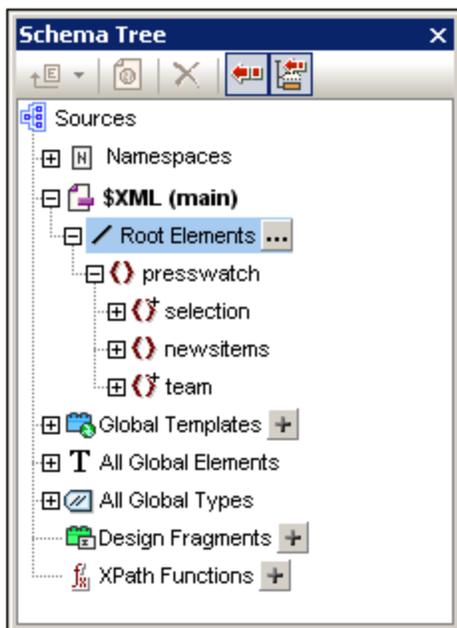
XSLT Files

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](#)²²⁹.

2.2.2 Schema Tree

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

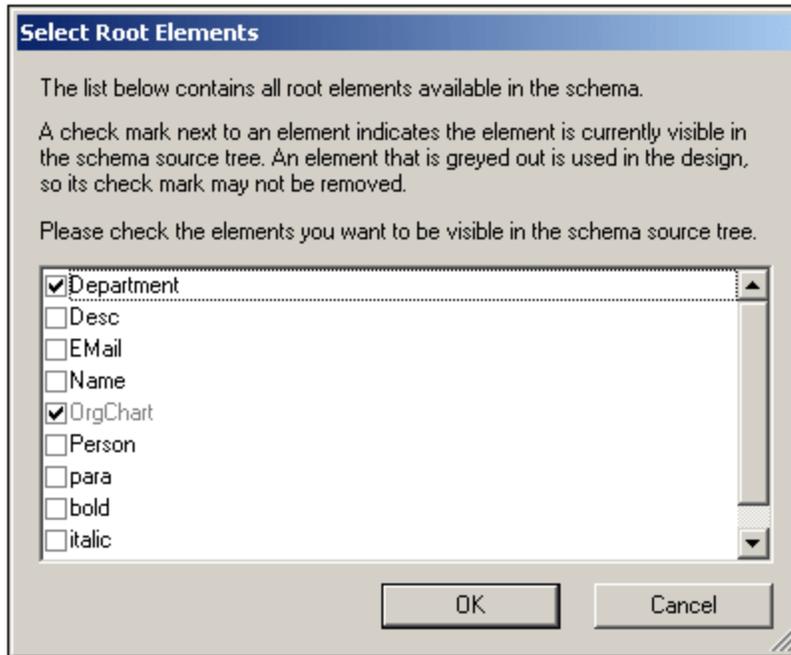
- Select multiple root elements (document elements) for a schema.
- Drag nodes (elements, attributes, global types) from a schema tree and drop them into the design. These nodes represent the XML content that is to be included in the output.
- View listings of all global elements and types in the schema 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](#)²²⁵.
- Insert and and edit user-defined [XPath functions](#)³⁴⁴ for the SPS.



Root elements

For each schema, under the `$XML` heading, the selected [Root elements](#)²¹ (or [document elements](#)²¹) 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](#) ¹⁷⁴ 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](#) ²¹⁵ can be created.

Global elements and global types

Global elements and global types can be used to create [global templates](#) ²¹⁵ which can be re-used in other templates. Additionally, global types can also be used directly in templates.

Design Fragments

All the [Design Fragments](#) ²²⁵ 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](#) ²²⁵ 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](#) ³⁴⁴, 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.

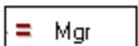
Symbols used in schema trees

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



Name

Element.



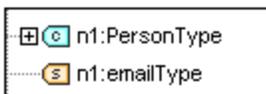
Mgr

Attribute.



Person

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



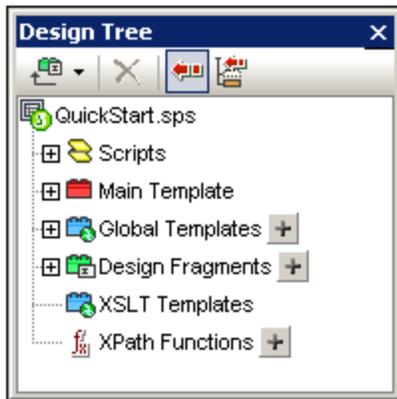
n1:PersonType

n1:emailType

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

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](#)³⁶², which shows all the JavaScript functions that have been defined for the SPS using the JavaScript Editor of StyleVision.
- [Main Template](#)²¹⁵, which displays a detailed structure of the main template.
- [Global Templates](#)²¹⁵, which lists the global templates in the current SPS, as well as the global templates in all included SPS modules.
- [Design Fragments](#)³⁹, which shows all the Design Fragments in the design, and enables you to create, edit, rename, and delete them.
- [XSLT Templates](#)⁴⁰, which provides the capability to view XSLT templates in imported XSLT files.
- [User-Defined XPath Functions](#)³⁴⁴, 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.



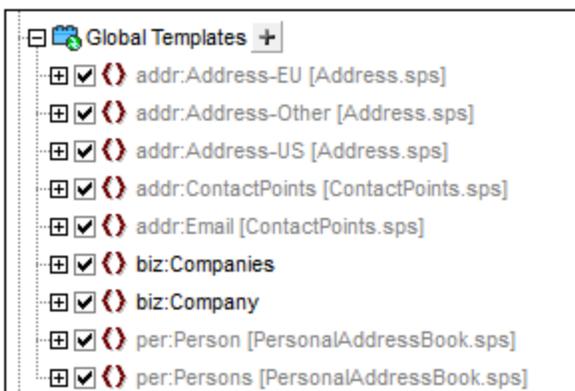
The **Remove** command removes the selected Design Tree item from the Design Tree. **Make Design Fragment** creates a [Design Fragment](#)³⁹ in the design and adds an item for it in the Design Tree. **Expand All** expands all the items of the Design Tree.

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

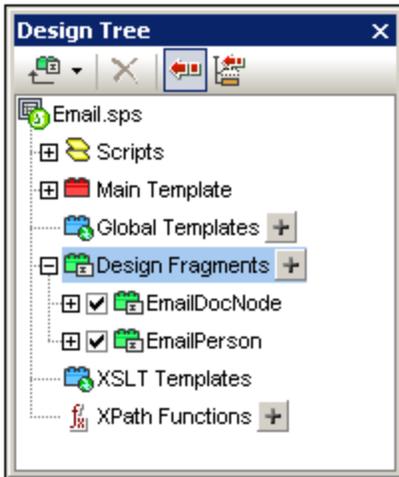
The [Global Templates](#)²¹⁵ item lists all global templates in the current SPS and in all added SPS modules. Global templates defined in the current SPS are displayed in black, while global templates that have been defined in added modules are displayed in gray (*see screenshot below*). Each global 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

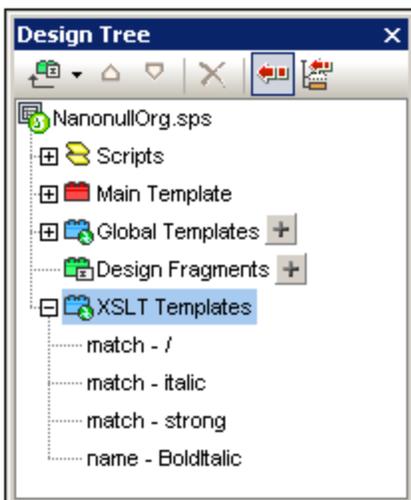
The [Design Fragments](#) ²²⁵ item lists all the Design Fragments in the current SPS and in all added SPS modules. Design Fragments defined in the current SPS are displayed in black, while Design Fragments that have been defined in added modules are displayed in gray (see *screenshot below*). Each Design 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

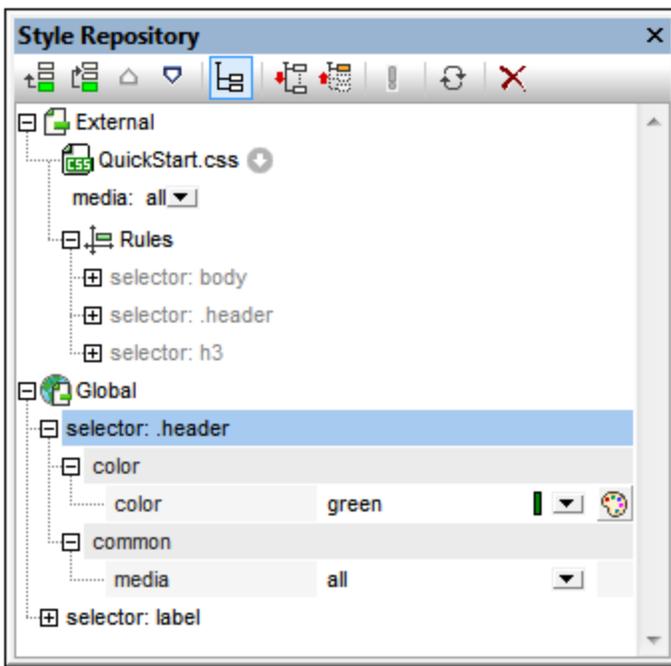
In the Design Tree sidebar (*screenshot below*), the XSLT Templates contained in the imported XSLT file are displayed under the XSLT Templates heading.



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](#)²²⁹.

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 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](#) ³¹⁹. Note that an external CSS stylesheet can also be added or a stylesheet removed via the [Design Overview sidebar](#) ³².

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](#) ³¹⁹.

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](#) ³¹⁹.

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.

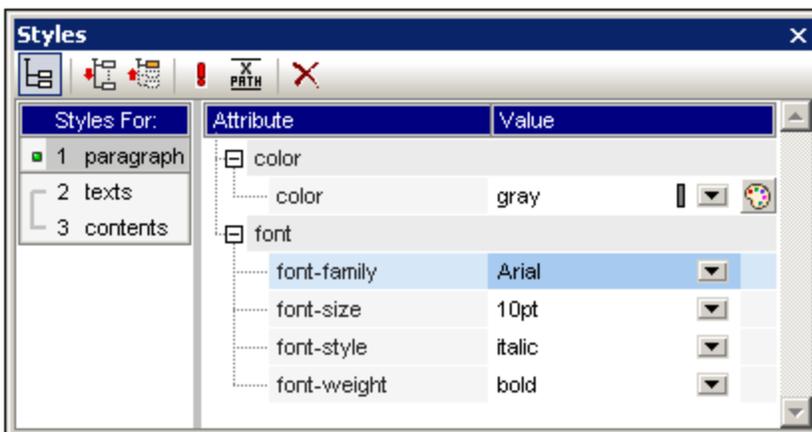
Editing CSS styles in the Style Repository

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](#) ³²⁰.
- 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](#) ³²³.

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](#) ⁴¹ 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](#) ³²⁵.
- 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](#) ³²⁷. 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.
-

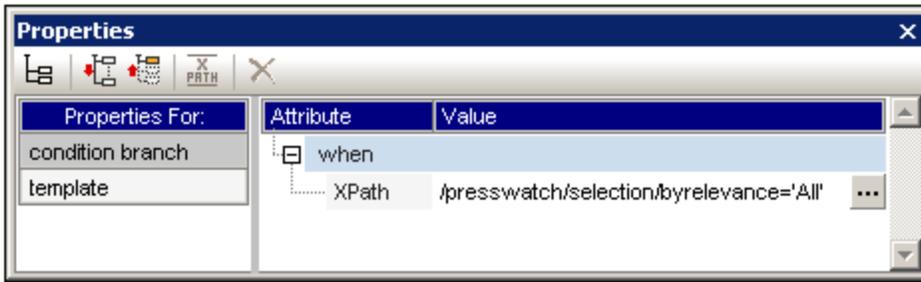
Toggle Important and Reset toolbar icons

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](#)⁴⁵ 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](#)⁴⁷ 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
Content	Content; Common; Event
Text	Text; Common; Event
Auto-Calculation	AutoCalc; Common; Event
Condition Branch	When
Data-Entry Device	Common; [Data-Entry Device]; Event; HTML
Image	Image; Common; Event; HTML
Link	Link; Common; Event; HTML
Table	Table; Common; Event; HTML; Interactive
Paragraph	Paragraph; Common; Event; HTML

The following points about component types should be noted:

- 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 ³¹⁰ 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 ²⁴⁰ .
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 ³⁶² 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 ¹⁴⁸ ; image ¹⁴³ ; link ²⁹⁸ ; table ¹¹⁸ ; paragraphs ³⁰⁶ . Note that there are different types of data-entry devices ¹⁴⁸ and paragraphs ³⁰⁶ , and that tables ¹¹⁸ 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](#)¹⁴³, [links](#)³⁰⁰, [paragraphs and other predefined formats](#)¹⁰⁵, and [condition branches](#)²⁴⁸. 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

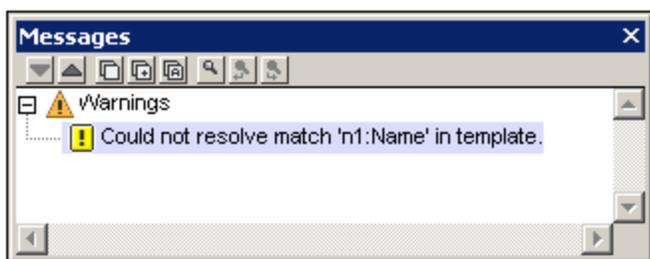
a property. When the XPath icon is toggled off, a static value can be entered as the value of the property. Also see [Style Properties Via XPath](#)³²⁹.

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](#)⁴⁷. To delete a property value, select the property and click the Reset icon  in the toolbar of the Properties sidebar.

2.2.7 Messages

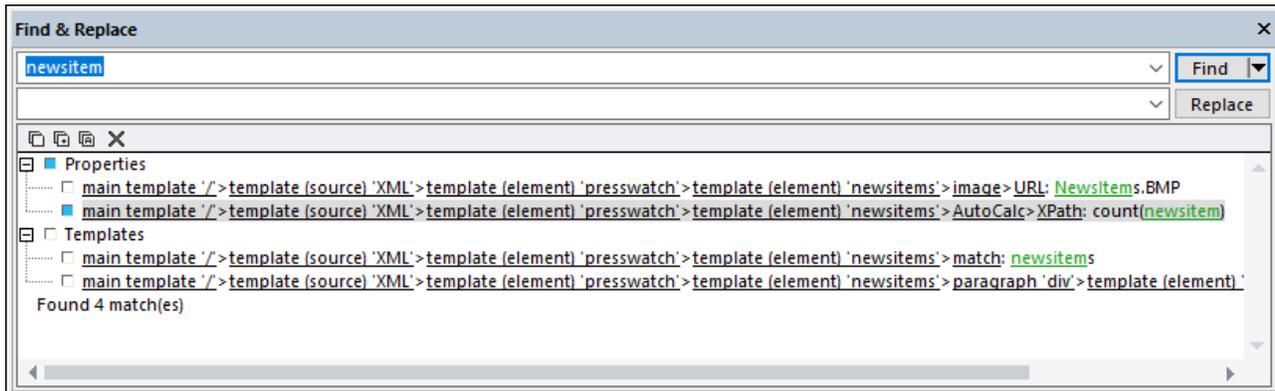
When StyleVision is opened for the first time, the Messages sidebar (*screenshot below*) is displayed below the Main Window of the GUI. To toggle the Messages sidebar on and off, click **View | Messages**.



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.

2.2.8 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](#)⁴⁴⁷ and [Edit | Replace](#)⁴⁴⁷ 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 and XSLT stylesheets), see the menu-reference topic [Find, Find Next, Replace](#)⁴⁴⁷.

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](#)⁴⁴⁸.

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](#)²³, C:\Documents and Settings\\My Documents\Altova\StyleVision2023\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](#) ⁵¹
- [How to add a schema source for the SPS](#) ⁵³
- [How to select the XSLT version of the SPS](#) ⁵⁴
- [How to assign the Working XML File](#) ⁵⁴
- [How to specify the output encoding](#) ⁵⁴
- [How to save the SPS document](#) ⁵⁴

Files in this section

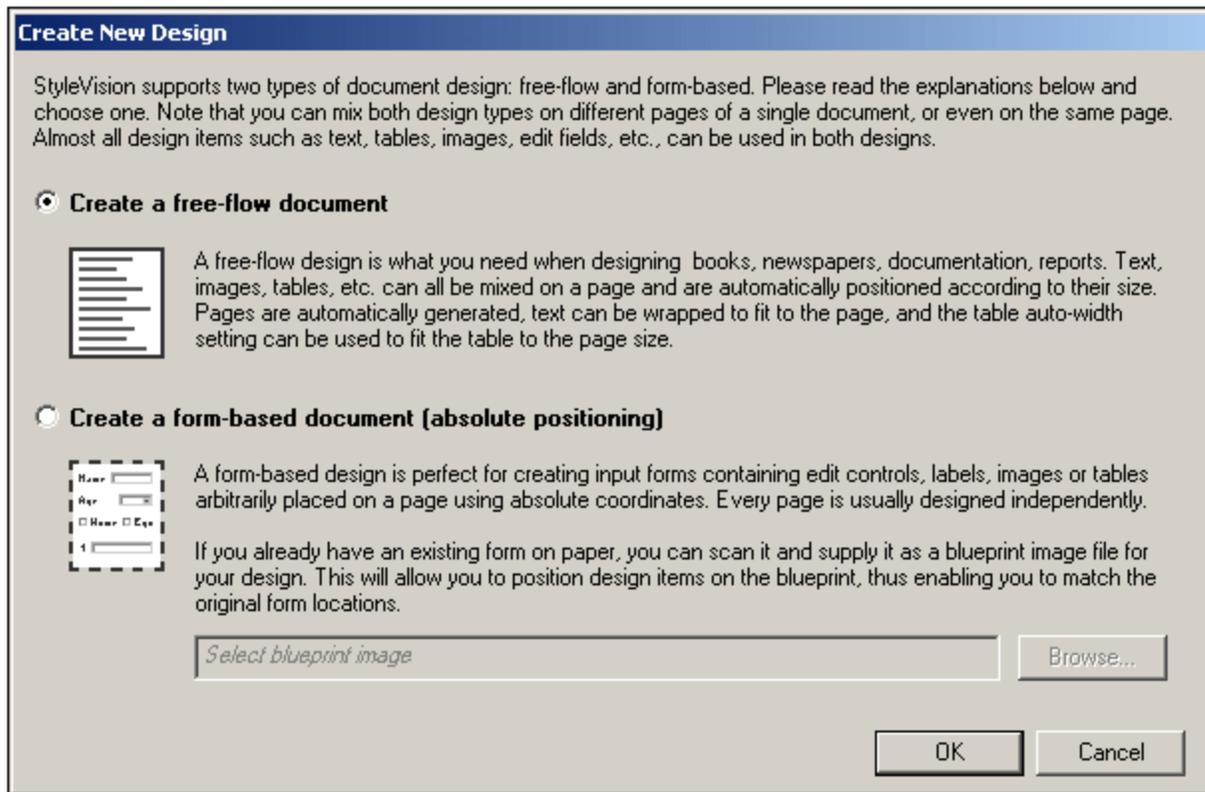
Files referred to in this section are located in the [\(My\) Documents folder](#) ²³, C:\Documents and Settings\\My Documents\Altova\StyleVision2023\StyleVisionExamples\Tutorial\QuickStart:

- 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 \(Empty\)](#) ⁴²³ or select **New (Empty)**  in the dropdown list of the [New icon](#) ⁴²¹  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).

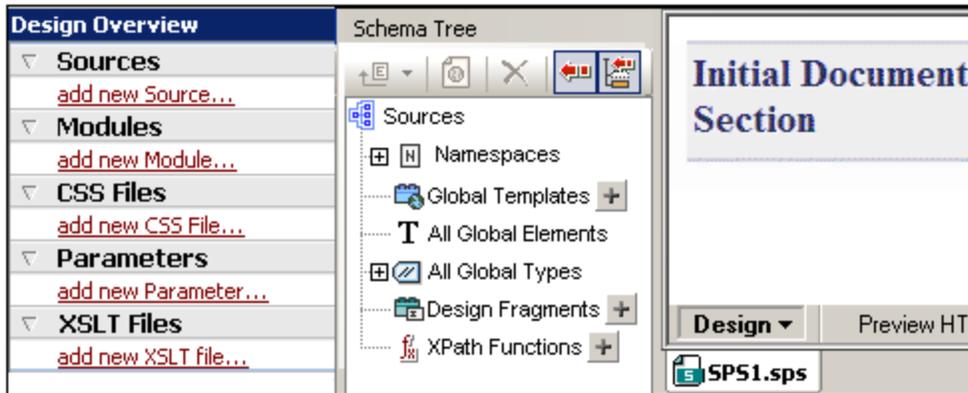


In a free-flowing document design, document content is laid out to fit the output media object or viewer (paper or screen). Items in the document content can only be placed relative to each other, and not absolutely. This kind of design is suited for documents such as reports, articles, and books.

In a form-based document, a single [Layout Container](#)¹⁵⁹ 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](#)¹⁵⁹. 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](#)²⁷ (screenshot below).

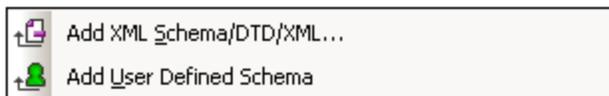


In [Design View](#)²⁷, an empty main template is displayed. In the [Design Overview](#)³² and [Schema Tree](#)³⁵ sidebars, there are no schema entries.

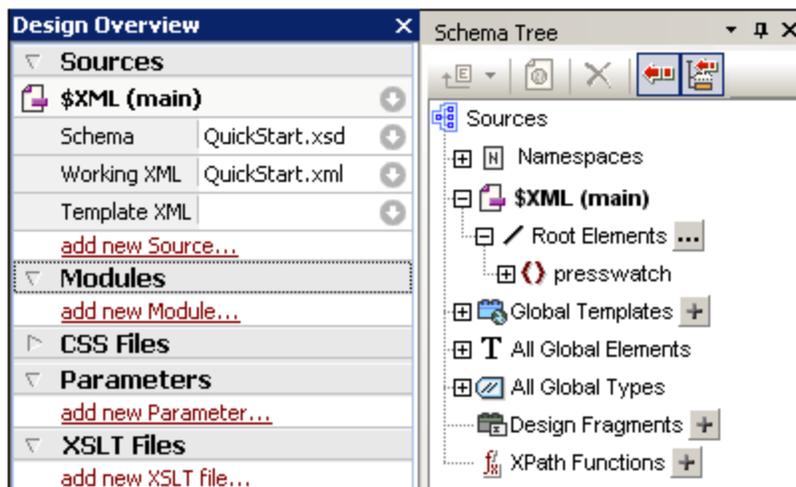
Adding a schema source

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](#)⁵¹ (*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](#)⁵¹ (*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](#)²²; (ii) In the Schema Tree sidebar, the Root Elements tree would list the one or more [root elements \(document elements\)](#)²¹ you select from among the [global elements](#)²¹ defined in the schema. In the case of this schema, the element `presswatch` is selected by default because it is the one [global element](#)²¹ in the schema that lies clearly at the top of the hierarchy defined in the schema; (iii) All [global elements](#)²¹ in the schema are listed in the [All Global Elements tree](#)³⁵.

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](#)²² 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](#)²² 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](#)²² 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](#)⁵¹, as the Working XML File.

Specifying the encoding of output

In the Default Encoding tab of the Options dialog ([Tools | Options](#)⁵¹³), set the HTML encoding to Unicode 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](#)⁵¹. Do this via the menu command [File | Save Design](#)⁴³⁴ 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..

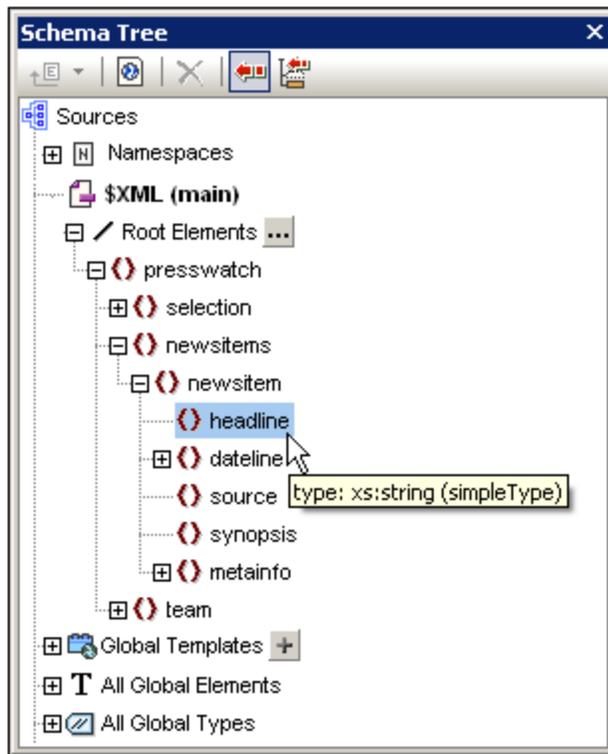
3.2 Inserting Dynamic Content (from XML Source)

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.

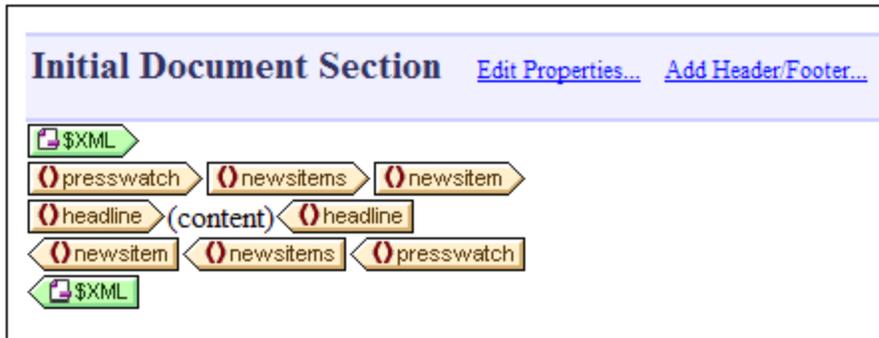
Inserting element contents

In your SPS, do the following:

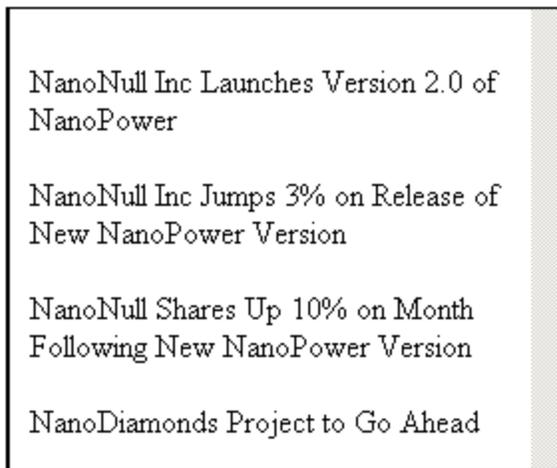
1. In the [Schema Tree sidebar](#)³⁵, expand the schema tree up to the children of the `newsitem` element (*screenshot below*).



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](#)³⁰, 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](#) ²⁸ (screenshot below). The HTML preview shows the contents of the `headline` child elements of `newsitem`, each as a text string.

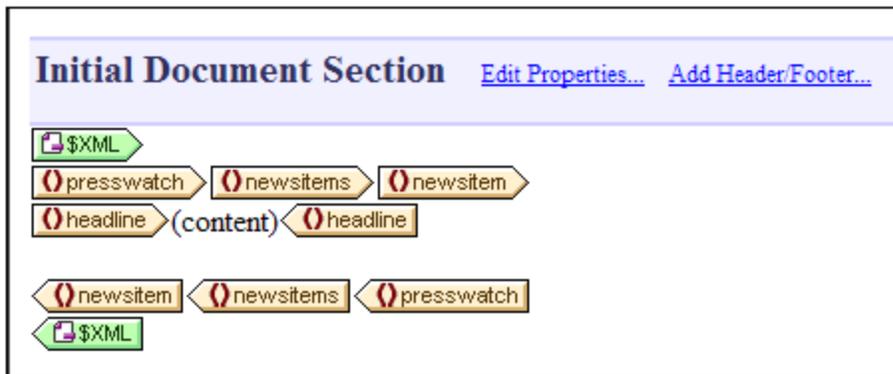


Note: You can also create the contents of a node by using the following steps: (i) Click the the Insert Contents icon in the [Insert Design Elements toolbar](#) ⁴¹⁸, (ii) Click the location in the design where you wish to insert the content, (iii) Select, from the Schema Selector tree that pops up, the node for which you wish to create contents.

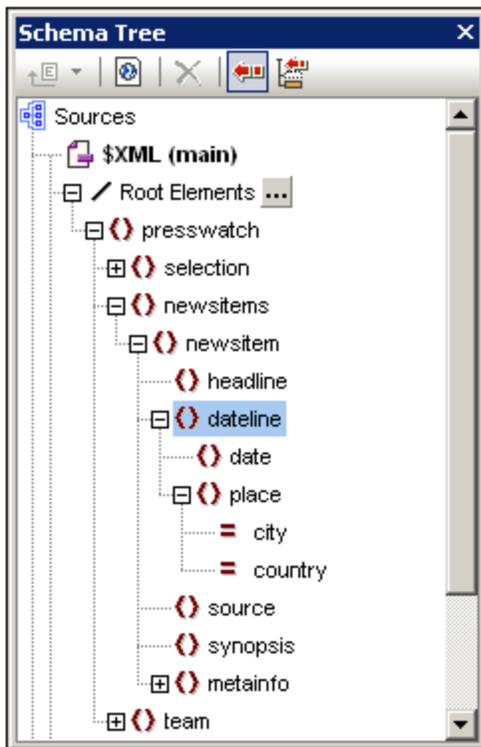
Inserting attribute contents

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

1. Place the cursor after the end tag of the `headline` element and press **Enter**. This produces an empty line (screenshot below).

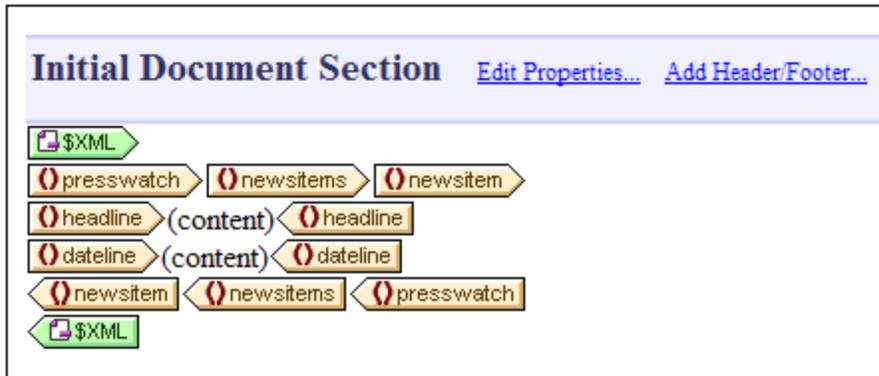


2. In the Schema Tree sidebar, expand the `dateline` element (*screenshot below*).



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

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

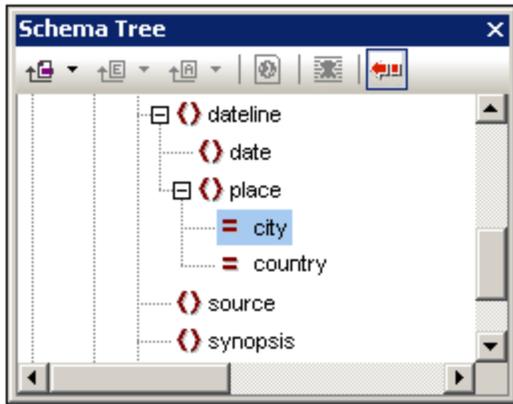


4. Switch to [HTML Preview](#)²⁸ and look carefully at the output of `dateline` (*screenshot below*).



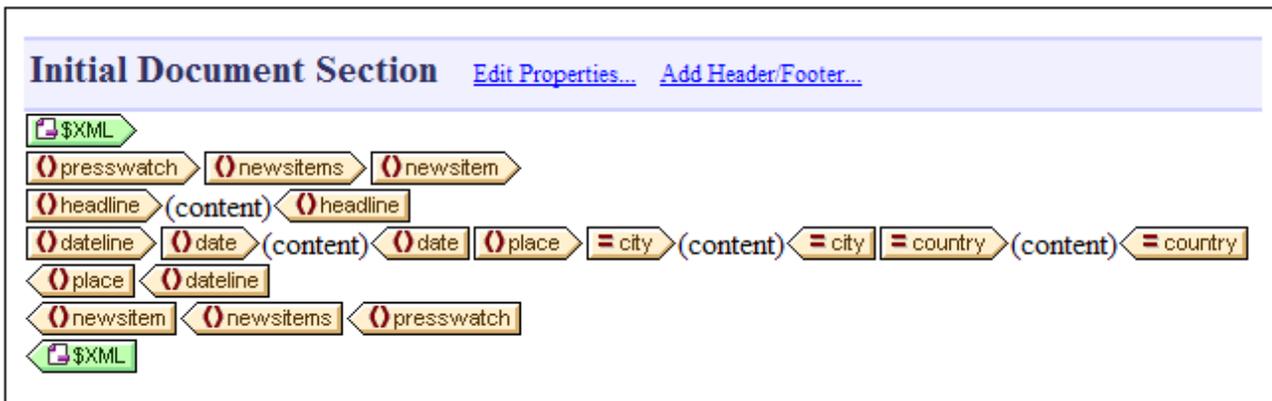
Notice that while the contents of the `date` children of `dateline` elements have been output, no contents have been output for the `place` children of `dateline`. This is because the `place` data is contained in the attributes of the `place` element (in the attributes `city` and `country`) and attribute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

5. Drag the `date` element from the [Schema Tree sidebar](#)³⁵ 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](#)³⁵.



7. Drag the @city attribute node into [Design View](#)²⁷, and drop it (create as contents) just after the end tag of the date element.
8. Drag the @country attribute node into [Design View](#)²⁷, 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:



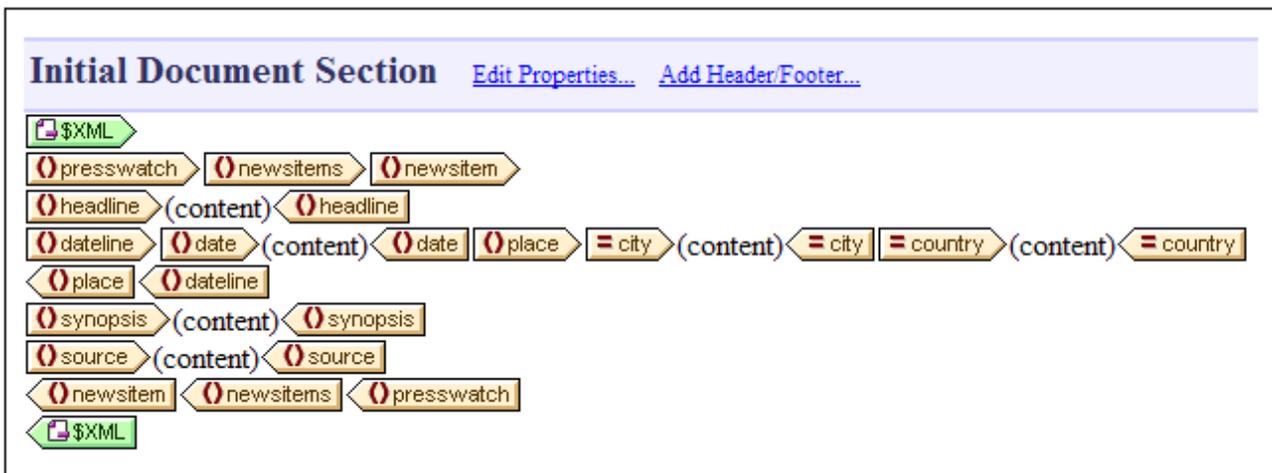
The [HTML Preview](#)²⁸ will look like this:



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](#)²⁸ to see the output. This is an important point to note: That the order in which nodes are placed in the [main template](#)²¹ is the order in which they will appear in the output (see the section, [Templates and Design Fragments](#)²¹⁵, 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](#)²⁸, 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](#)⁷⁷, you will learn how to create a node as a combo box, and in the section, [Using Global Templates and Rest-of-Contents](#)⁸⁴, 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

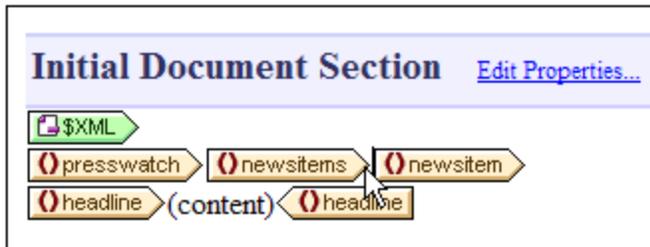
Static content is content you enter or insert directly in the design—as opposed to content that comes from the XML source. A variety of static components can be placed in an SPS design. In this part of the tutorial, you will learn how to insert the following static components:

- [An image](#)⁶²
- [A horizontal line](#)⁶³
- [Text](#)⁶³

Inserting a static image

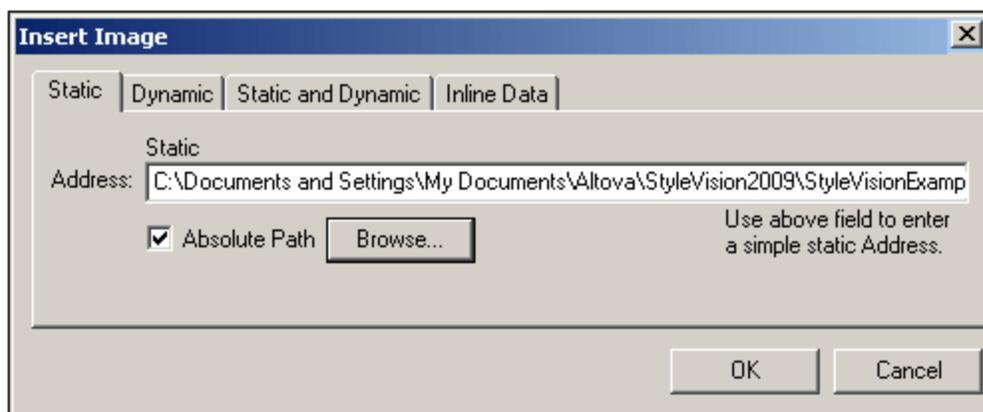
The static image to insert is in the [\(My\) Documents folder](#)²³: C:\Documents and Settings\\My Documents\Altova\StyleVision2023\StyleVisionExamples\Tutorial\QuickStart\NewsItems.BMP. It will be used as the header of the document. To insert this image at the head of the document, do the following:

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



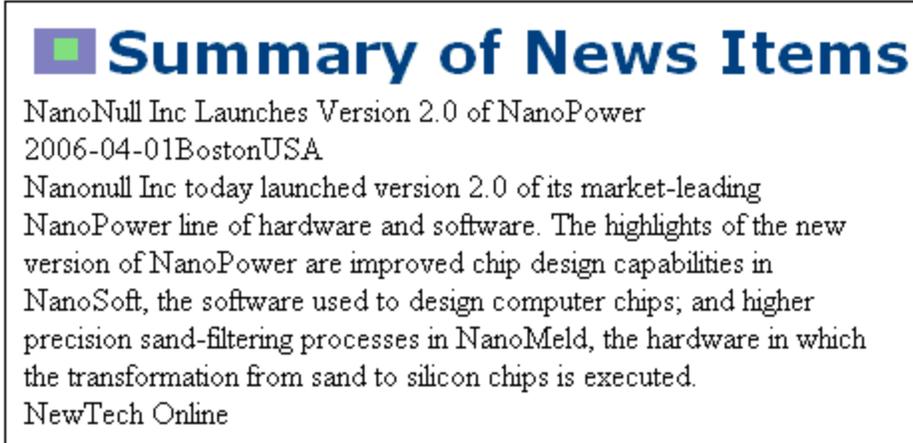
Notice that the cursor is within the `newsitems` element but outside the `newsitem` element. It will therefore be inserted in the output once, at the start of processing of the `newsitems` element (because there is only one `newsitems` element defined in the schema).

2. Right-click, and select [Insert | Image](#)⁴⁶¹. 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](#)⁴⁶³. A horizontal line is inserted.

Set properties for the line as follows:

1. With the line selected in [Design View](#)²⁷, in the [Properties sidebar](#)⁴⁴, 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](#)²⁷, in the [Styles sidebar](#)⁴³, 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](#)²⁸.

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 \(from XML Source\)](#)⁵⁵), 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*)



This part of the output will now look like this:

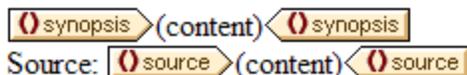
Summary of News Items

NanoNull Inc Launches Version 2.0 of NanoPower

2006-04-01: Boston, USA

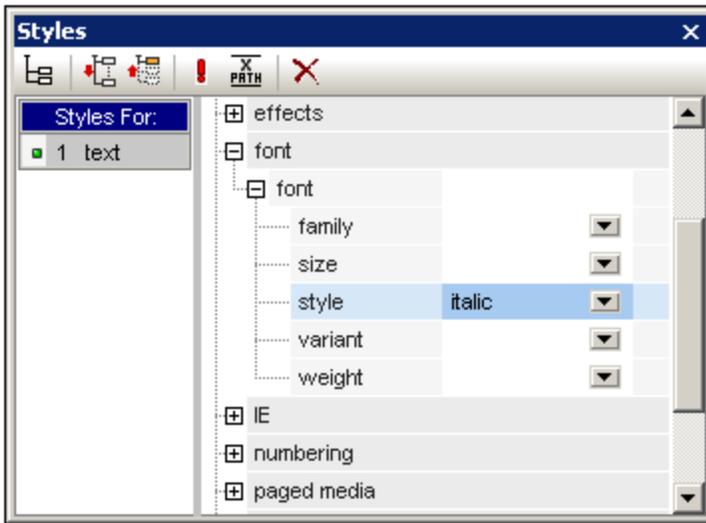
Notice the colon, spacing and comma in the `dateline` output. All of these text items are static text items that were inserted directly in the design.

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*).

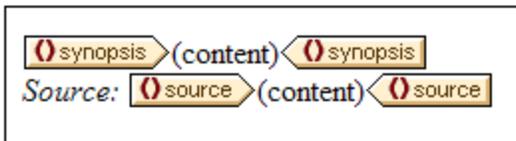


Formatting static text

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](#)⁴³ (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 give 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](#)³¹⁹, 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](#)³⁰⁶ 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](#)³²⁰ or [global](#)³²³ style rules.
- You can specify styles at three levels. These are, in increasing order of priority: (i) style rules in [external stylesheets](#)³²⁰, (ii) [global style rules](#)³²³, and (iii) [local style rules](#)³²⁵.

In this section, you will learn how to:

- [Assign predefined formats](#)⁶⁷
- [Assign a component a class attribute](#)⁶⁸
- [Define styles in an external CSS stylesheet](#)⁶⁹ and add this stylesheet to the style repository of the SPS
- [Define global style rules](#)⁶⁹
- Define [local styles for a selection of multiple design components](#)⁷¹
- Define [local styles for a single component](#)⁷¹

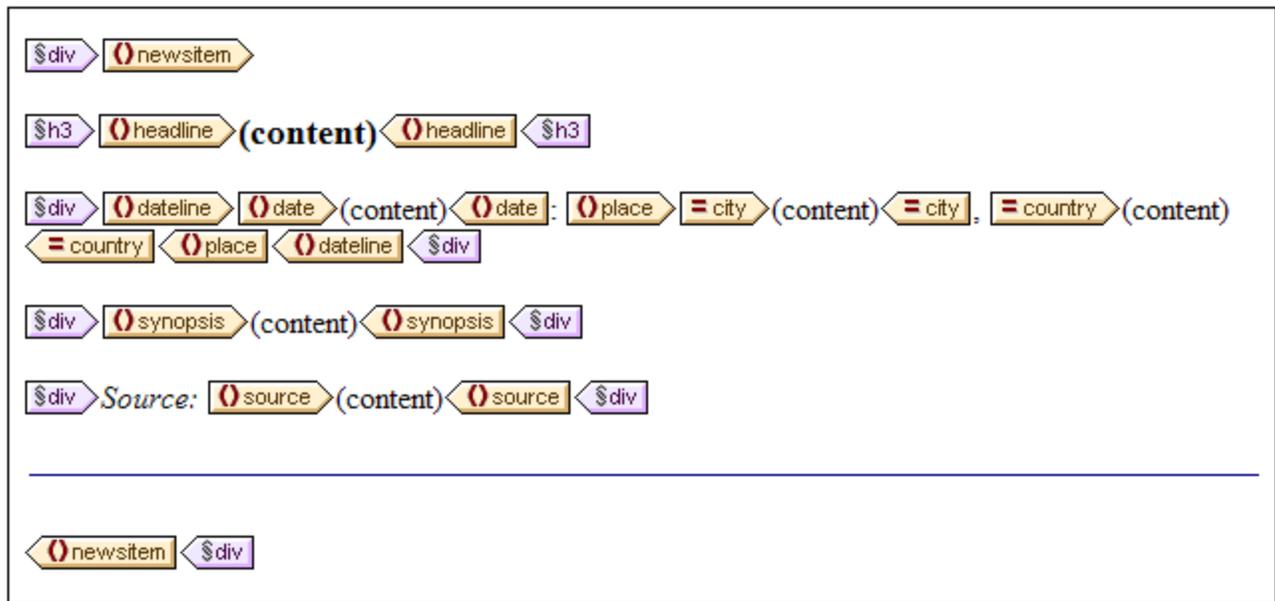
Assigning predefined formats

One reason to assign a [predefined format](#)³⁰⁶ is to give a component the inherent styling of that [predefined format](#)³⁰⁶. 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 `newstitem`, `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 `newstitem` element is inside a `div` component.

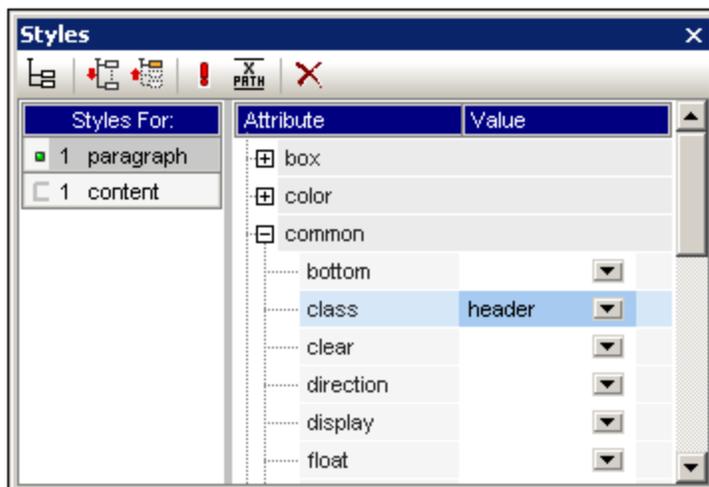


You have now grouped components together in different `div` blocks. [Later in this section](#) ⁷¹, you will learn how to assign styles to such blocks of grouped components.

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](#)⁴¹ (in Design View), do the following:

1. Select the `External` item.
2. Click the **Add** button in the toolbar of the [Style Repository sidebar](#)⁴¹. This pops up the Open dialog.
3. **Browse for the file** `C:\Documents and Settings\\My Documents\Altova\StyleVision2023\StyleVisionExamples\Tutorial\QuickStart\QuickStart.css`, which is in the [\(My\) Documents folder](#)²³, 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.

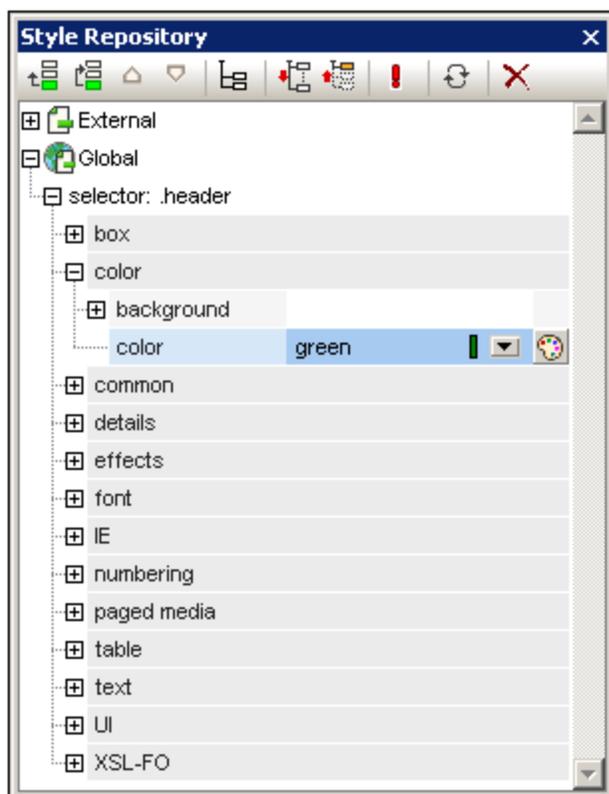
Source: NewTech Online

Defining global style rules

[Global style rules](#)³²³ can be defined for the entire SPS using CSS selectors. The rules are defined directly in the [Style Repository sidebar](#)⁴¹. Create a global style rule for the `header` class as follows:

1. With [Design View](#)²⁷ active, in the [Style Repository sidebar](#)²⁷, 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

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](#)⁴³, 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](#)⁴³, 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](#)⁴³, 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](#)⁴³, 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](#)³⁰⁶ 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.

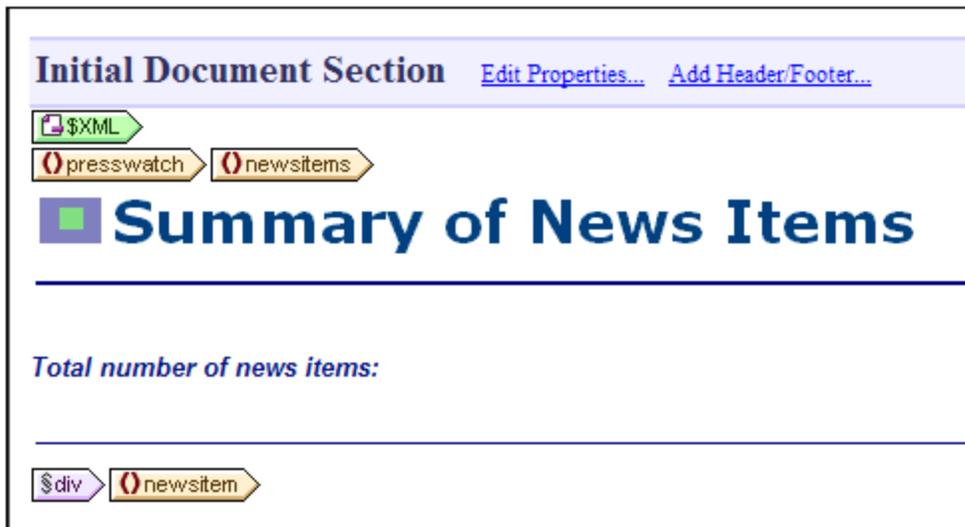
3.5 Using Auto-Calculations

[Auto-Calculations](#)²⁴⁰ 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.

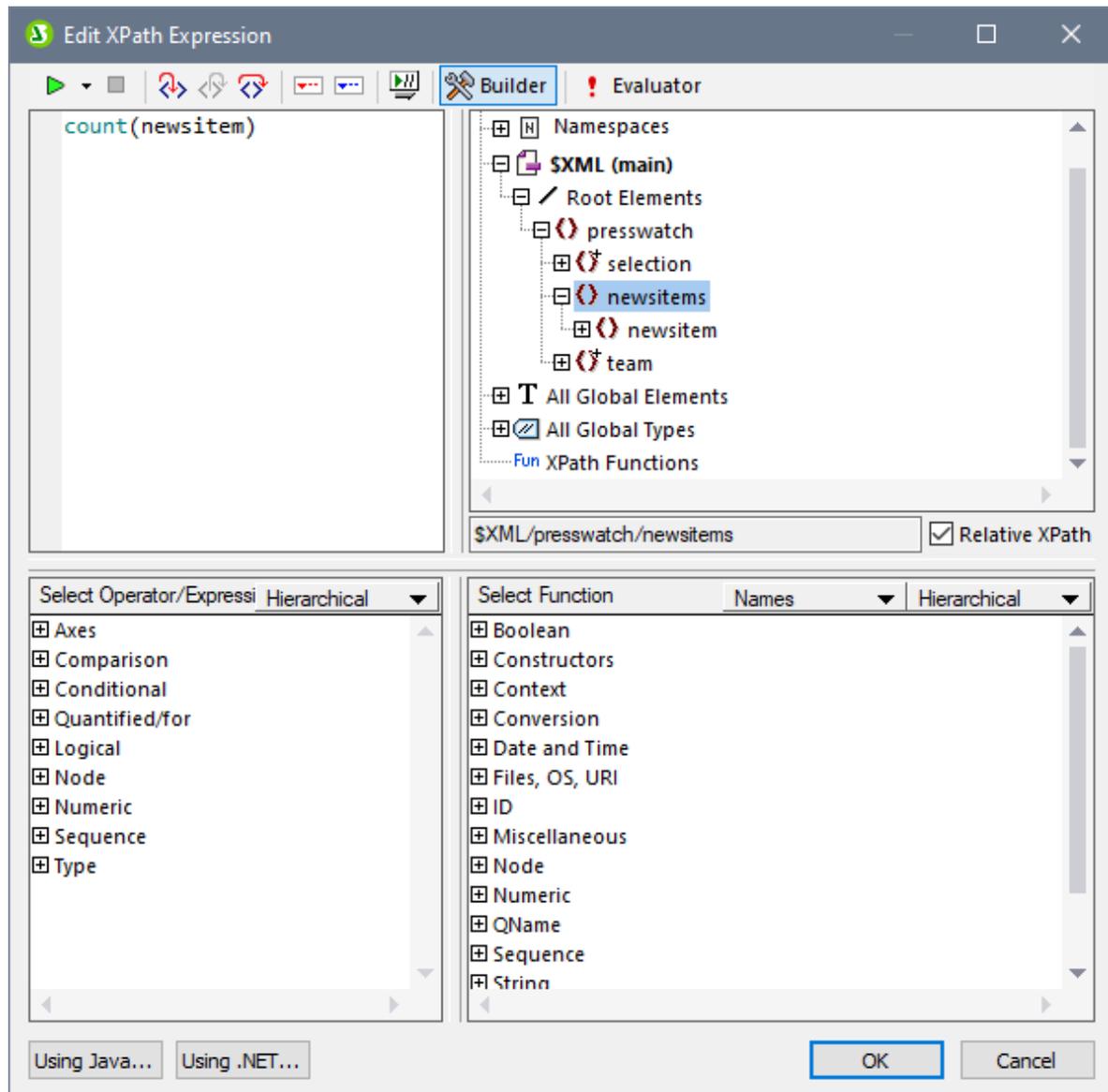
Counting the news item nodes

In the design, do the following:

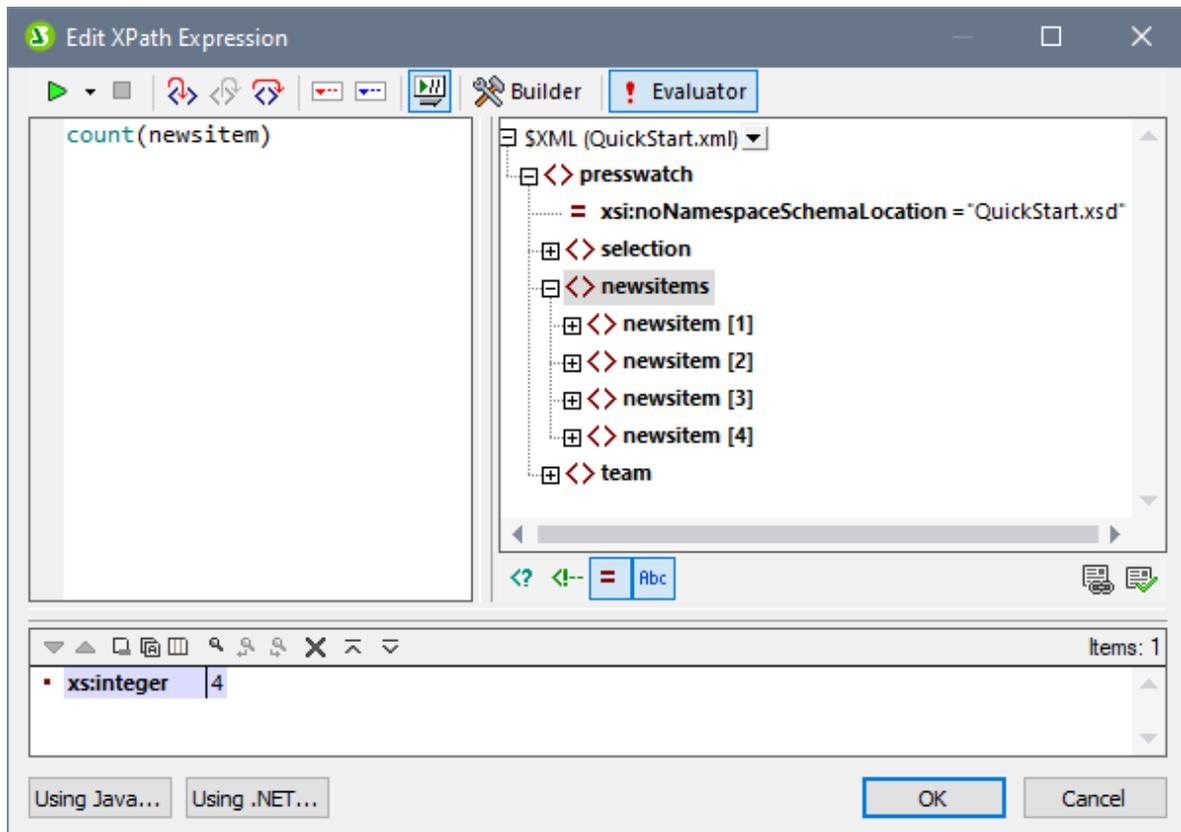
1. Create space, as shown in the screenshot below, for a line of static text (on which the Auto-Calculation will also be placed). Use the **Return** key to add new lines and insert a horizontal line below the space you create (see *screenshot*).



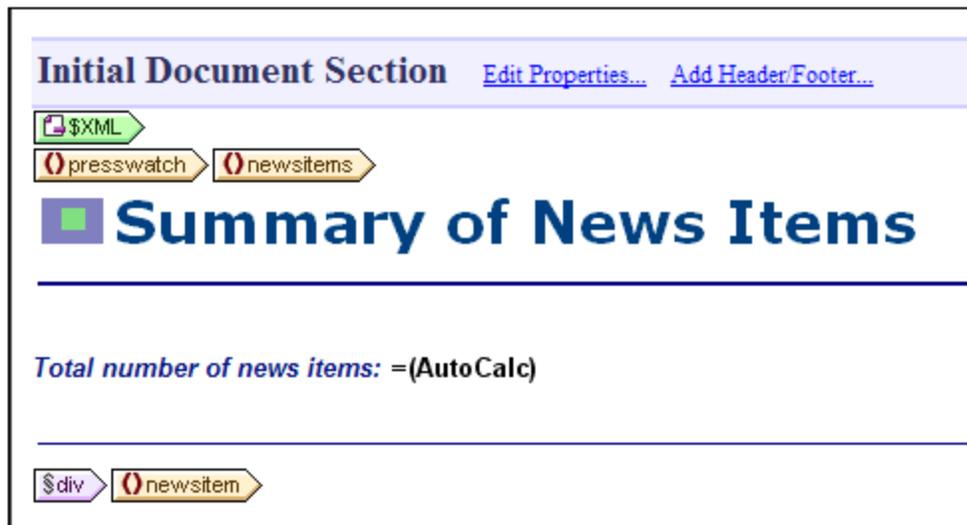
2. Type in the static text "Total number of news items:" as shown in the screenshot above.
3. Apply local styling of your choice to the static text. Do this as described in the section [Formatting the Content](#)⁷¹.
4. Place the cursor after the colon and select **Insert | Auto-Calculation | Value**. This pops up the [Edit XPath Expression dialog](#)³⁹⁷ (*screenshot below*). (Alternatively, you can right-click and select the command in the context menu.)



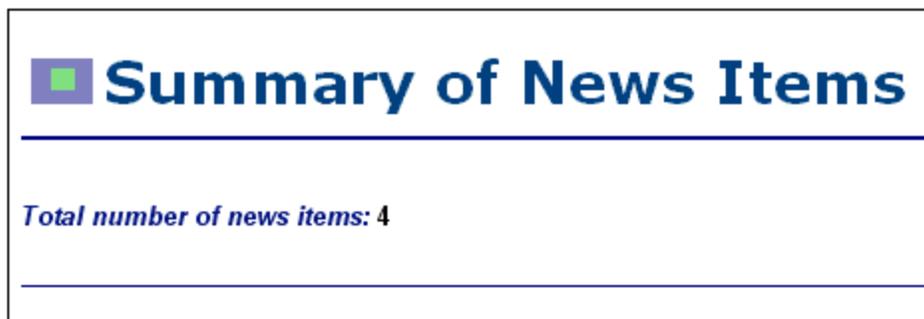
- In the schema tree, note that the context node is `newsitems`, which is highlighted. Now, in the *Expression* text box either type in the expression `count(newstitem)` or build the expression using the entry-helper panes below the Expression text box. (Double-click the `count` function (found in the *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](#) ³⁹⁷.



- 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](#)⁷¹.



Your HTML output will look like this:

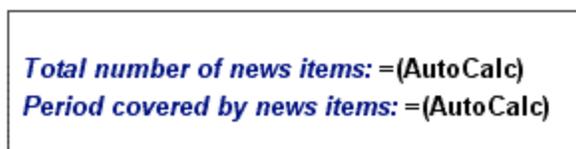


Displaying the period covered by news items

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. Each `relevance` node contains a heading under which the relevance of the news item is indexed. Further, there is a node `/presswatch/selection/byrelevance`. The content of this node contains one of the relevance headings and determines what news items are displayed. For example, if the content of the `byrelevance` node is `NanoPower`, then all news items that have a `relevance` node containing `NanoPower` are displayed. A condition can test what the content of the `byrelevance` node is (by looking up that node) and provide appropriate processing (displays) in the conditional template. In this section, you will create a conditional template that displays those news items that have a `relevance` element that matches the content of `byrelevance`.

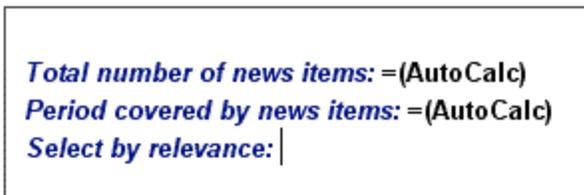
We will proceed as follows:

1. Create a combo box which displays the value of the `byrelevance` node. 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.

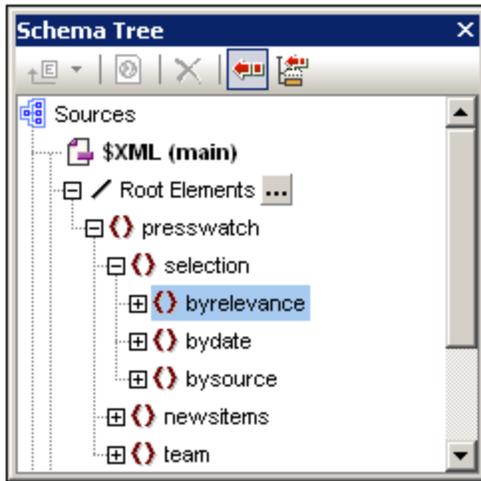
Creating the combo box to select unique node values

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

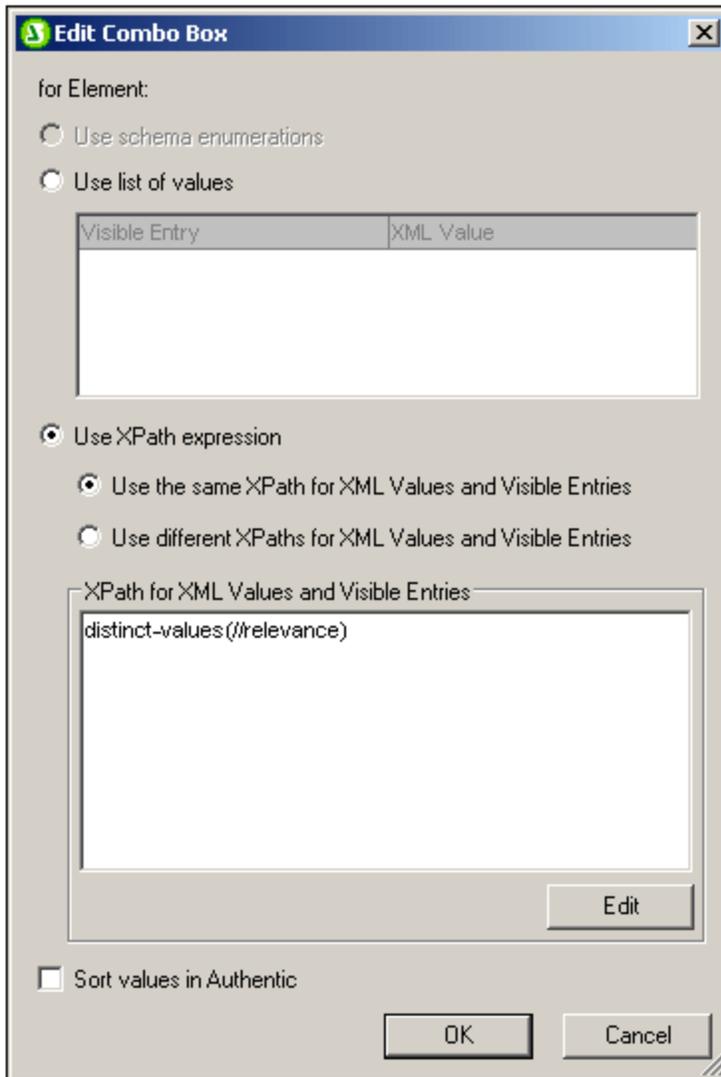
1. Insert the static text "Select by relevance: " at the head of the document and just below the [second Auto-Calculation](#)⁷³ (screenshot below).



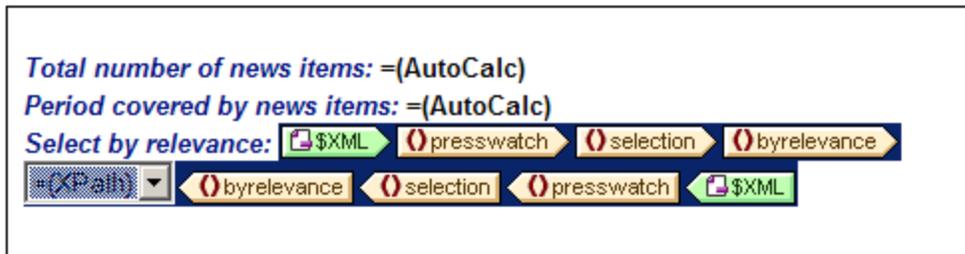
2. Drag the `byrelevance` node from the [Schema Tree sidebar](#)³⁵ (screenshot below), and drop it after the newly entered static text.



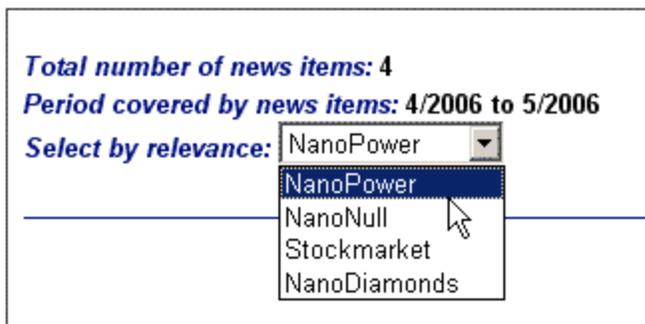
3. In the context menu that appears, select Create Combo Box. This pops up the dialog shown below.



- 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. Note that although the values of all `relevance` nodes will appear in the HTML combo box, selecting one of them in HTML Preview will have no effect on the content of the node in the XML document (which is what the SPS acts on). The HTML document is an output obtained by transforming the XML document; it does not accept input. The combo box is used here to demonstrate alternative ways of presenting content.
- Click **OK** to finish. The combo box is inserted and the design will look something like this:



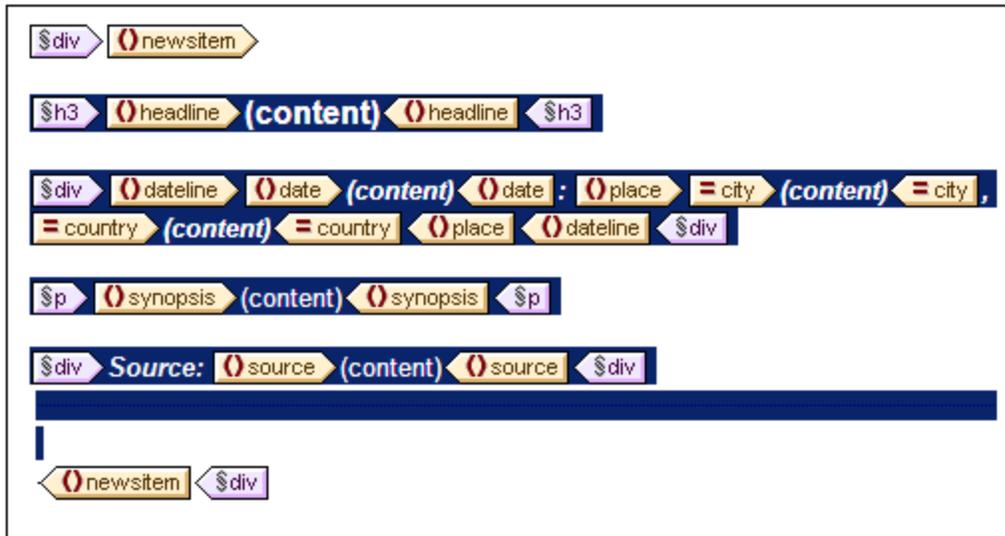
- Switch to [HTML Preview](#) ²⁸. When you click the dropdown arrow of the combo box, notice that the list contains the unique values of all `relevance` nodes (*screenshot below*). Check this against the XML document. This is a dynamic listing that will be augmented each time a new `relevance` value is added to the XML document.



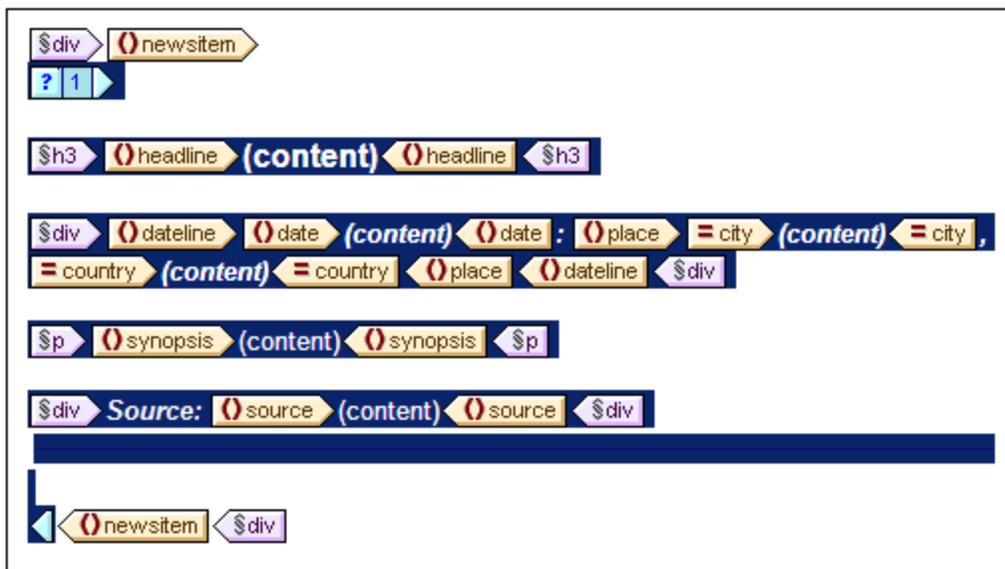
Inserting a condition to display news items having the selected `relevance`

The condition selects `newsitem` elements that have a `metainfo/relevance` element with a value that is the same as that in 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** ⁴⁷⁸. This pops up the **Edit XPath Expression dialog** ³⁹⁷.
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*).

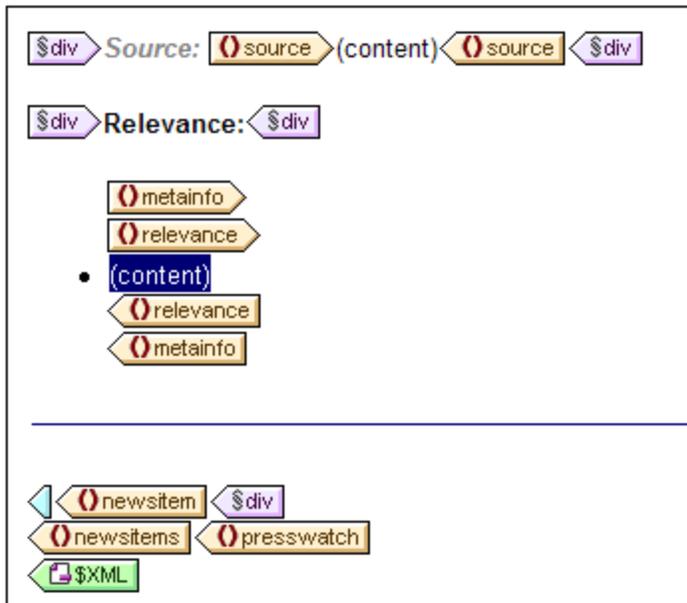


Note that there is a single branch in this condition. News items for which the condition test evaluates to true are displayed, those for which the condition test does not evaluate to true are not displayed. The condition in this case, therefore, works as a filter. Later in this section, you will add 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](#)³⁵ 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:

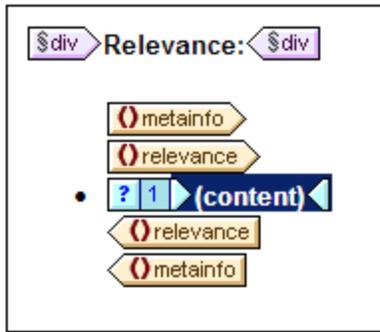


Now, in HTML Preview, check the results for different selections of `relevance`; Do this by: (i) changing the value of the `byrelevance` node in the XML document; (ii) saving the XML document; (iii) and then re-opening the SPS file in StyleVision.

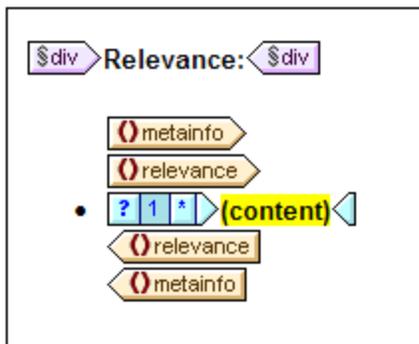
Making the selected `relevance` element bold

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](#)³⁹⁷ 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.



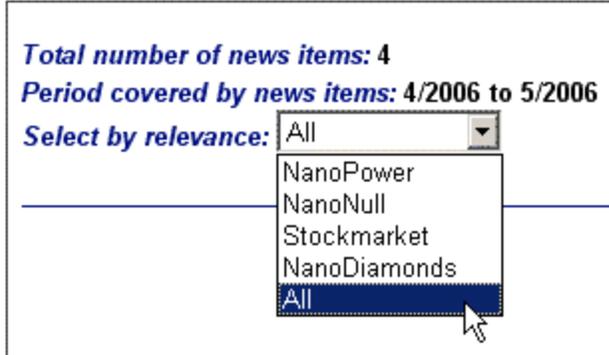
7. Notice that the contents of the *Otherwise* branch are a copy of the first branch; the `contents` placeholder is bold and has a yellow background. Remove this formatting (bold and background-color) from the `contents` placeholder.

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 HTML Preview.

Modifying the combo box and inserting a second condition branch

In the combo box, 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](#)¹⁵¹ 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 HTML Preview (*screenshot below*).



The value `All` can now be entered in the `byrelevance` node. 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](#)³⁹⁷ 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 content of the `byrelevance` node is `All`.

After you have completed this section, save the design.

3.7 Using Global Templates and Rest-of-Contents

[Global templates](#)²¹ 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.

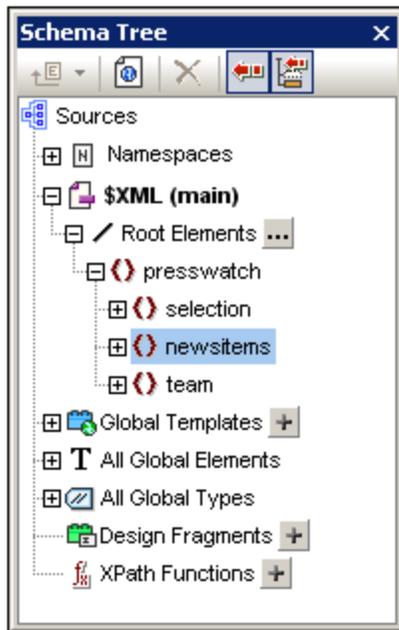
There are two mechanisms that are used to invoke a global template from the main template:

- 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](#)²¹ `member`.

Inserting the rest-of-contents instruction

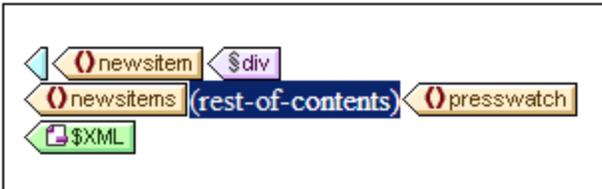
The broad structure of the schema is shown in the screenshot below.



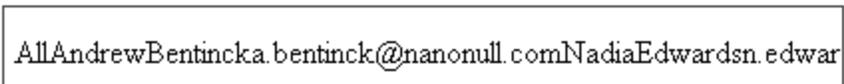
The document element `presswatch` contains three children: (i) `selection`; (ii) `newsitems`; and (iii) `team`. The main template you have created this far processes the `/presswatch` element. Within the `presswatch` element,

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**⁴⁵⁸. 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*):

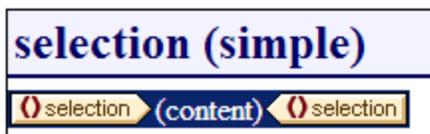


This string is the result of the application of the built-in templates to the `selection` and `team` elements. The built-in template for elements processes child elements. The built-in template for text nodes outputs the text in the text node. The combined effect of these two built-in templates is to 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`).

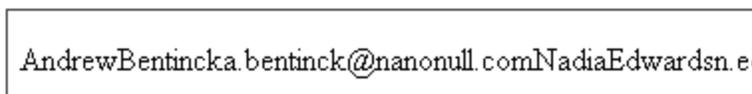
Creating a global template for `selection`

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](#)³⁵.
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*).

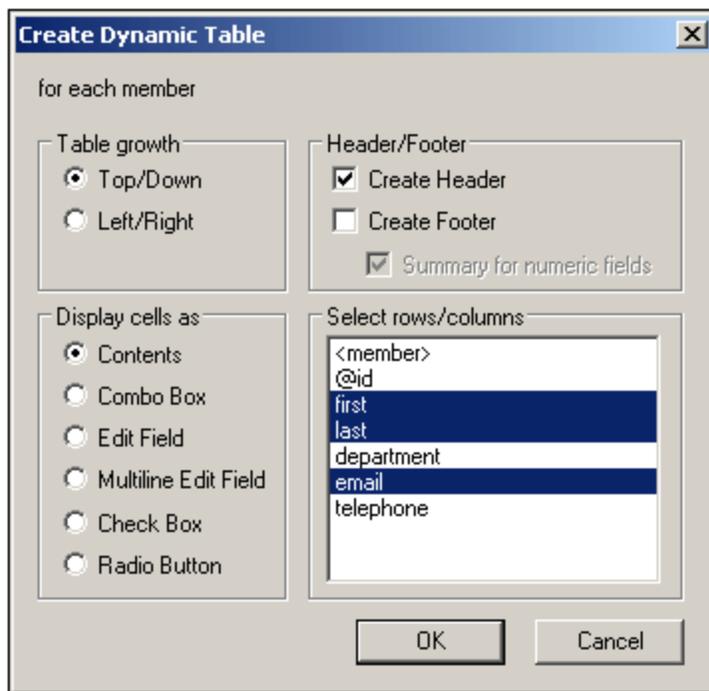


Since the global template for `selection` is empty, the child elements of `selection` are not processed.

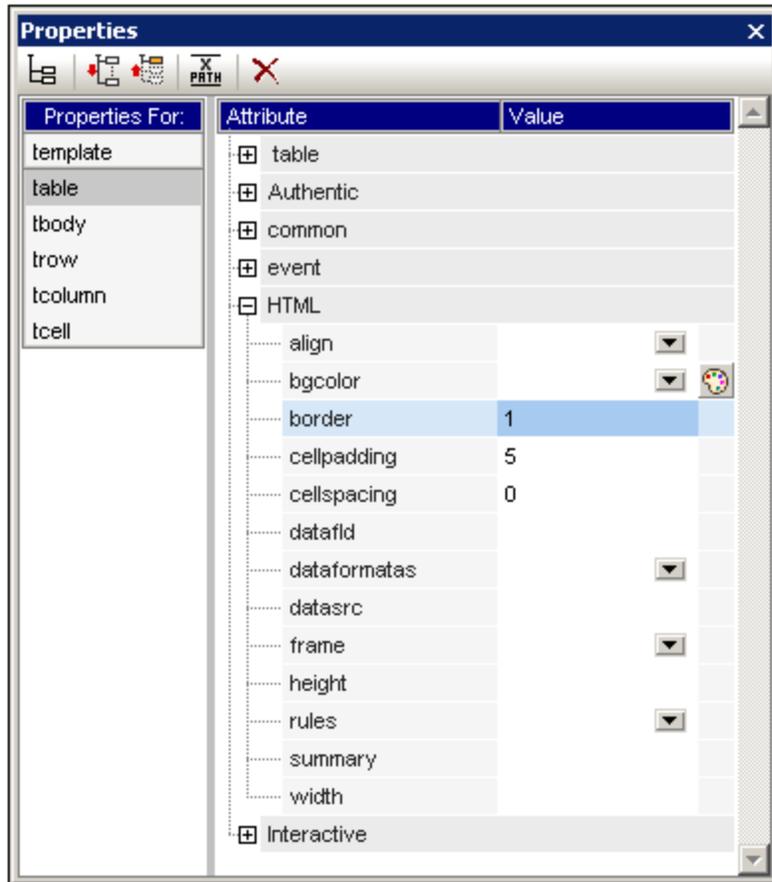
Creating a global template for `team/member`

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](#)⁴⁴, 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 `trow` 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](#)⁵⁵ of the document ([main template](#)⁵⁵ and [global templates](#)⁸⁴).
- How to insert [dynamic](#)⁵⁵ and [static](#)⁶² content in the design, using a variety of dynamic and static SPS components..
- How to use [CSS styles](#)⁵⁵, in [external stylesheets](#)⁶⁹, in [global style rules](#)⁶⁹, and in [local style rules](#)⁷¹.
- How to use [Auto-Calculations](#)⁷³ to derive additional information from the available XML data.
- How to use [conditions](#)⁷⁷ to filter the XML data and how to obtain different outputs depending on values in the XML data.
- How to use [global templates](#)⁸⁵ and [rest-of-contents](#)⁸⁴.

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

- [SPS File: Content](#)¹⁰²
- [SPS File: Structure](#)¹⁷²
- [SPS File: Advanced Features](#)²³⁹
- [SPS File: Presentation](#)³⁰⁵
- [SPS File: Additional Functionality](#)³³⁷

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

4 Usage Overview

Objectives

SPS documents that you create in StyleVision can be used to generate XSLT stylesheets for HTML. A stylesheet generated from an SPS can be used to transform any XML document 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](#)³⁵ 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](#)³². Alternatively, a new SPS can be created directly with a schema via the **File | New** command.
2. [Assign a Working XML File](#)³⁵ to the SPS. The [Working XML File](#)²² provides the XML data processed by the SPS when generating output previews. The [Working XML File](#)²² is assigned in the [Design Overview sidebar](#)³². The Working XML File enables you to preview output in StyleVision.
3. [Select the required XSLT version](#)⁹². In order to generate Text output, the XSLT version must be XSLT 2.0 or XSLT 3.0
4. Select the [Internet Explorer Compatibility](#)⁹³ to match the installed Internet Explorer version.
5. The SPS document is designed in [Design View](#)²⁷ using the various design components available to the designer. The [design process](#)⁹¹ consists of creating a document structure and defining [presentation properties](#)³⁰⁵.
6. The 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](#)⁹⁵ are required, these are [generated](#)⁹⁵.

4.1 SPS and Sources

Creating a new SPS file

To create a new SPS document, select an option from under the [File | New \(Ctrl+N\)](#)⁴²³ command or click the **New Design** icon  in the [Standard toolbar](#)⁴²¹. 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 two types of source files that can be assigned to an SPS:

- [Schema sources](#)⁹⁰
- [Working XML File](#)⁹⁰

These source file assignments are made in the [Design Overview sidebar](#)³². How to make the assignments is described in the section, [Design Overview](#)³². 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](#)³². A schema may be an XML Schema file (`.xsd` file), an XML Schema generated from an XML file, a DTD, or a user-defined schema. For each schema, one optional [Working XML File](#)⁹⁰ 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

can, optionally, have a [Working XML File](#)²² associated with it. The function of the [Working XML File](#)²² 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](#)²² is assigned in the [Design Overview sidebar](#)³².

4.2 Creating the Design

In the SPS design, you specify:

1. [What content](#)⁹¹ (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](#)⁹¹; and
3. [What presentation \(formatting\) properties](#)⁹¹ are applied to the various parts of the output.

Content for output

The content for the output can come from:

1. The XML document to which the SPS is applied. Content from the [XML document](#)²² is included in the SPS by dragging the required XML data node from the relevant schema tree in the [Schema Tree sidebar](#)³⁵ 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). 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](#)³⁵ 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](#)²⁴⁰.
5. For the HTML output, [JavaScript functions](#)³⁶² can be used to generate content.

Structure of output

In the SPS design, the [structure of the output](#)²¹⁵ can be controlled by using either: (i) a procedural approach, in which the output structure is specified in an [entry-level template](#)²¹⁵ (StyleVision's [main template](#)²¹⁵) and can be independent of the structure of the XML document; (ii) a declarative approach, in which [template rules are declared for various nodes](#)²¹⁵ (StyleVision's [global templates](#)²¹⁵), 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](#)²¹⁵ and [global templates](#)²¹⁵ to obtain the desired structure for the output document. The use of [Modular SPSs](#)²⁰¹ and [Design Fragments](#)²²⁵ 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](#)³⁰⁶. Specifying presentation properties is described in detail in the section, [Presentation Procedures](#)³⁰⁵.

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](#)²⁸ in the interface; and (ii) to [generate and save output files](#)⁹⁵ (HTML) from [within the interface](#)⁴⁴⁰ 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](#)²⁴⁰ and [Conditional Templates](#)²⁴⁵—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 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](#)¹⁴⁵.
- Support for new CSS styles (including CSS3 styles), which are listed below. Application of these styles is limited to 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](#)¹¹⁵.

Design View and IE versions

You can set up Design View for a specific IE version by specifying, in the [Properties](#)⁴⁴³ 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](#)¹⁰⁵ or as [user-defined elements](#)¹¹⁵. The HTML element will be rendered in 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 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](#)⁴⁴³ 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 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](#)⁴⁴³ dialog, the IE compatibility back to that of the previously selected IE version.

4.5 Generated Files

In StyleVision, XSLT stylesheets and output files can be generated using the [File | Save Generated Files](#)⁴⁴⁰ command or [StyleVision Server](#).

The following files can be generated from StyleVision:

- XSLT stylesheets based on the SPS design.
- Output files, generated by processing the [Working XML File](#)²² 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. 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](#)²⁸.

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. (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](#)

4.6 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](#)⁹⁶ 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](#)⁹⁷ and [Customizing your Catalogs](#)⁹⁸. The section [Variables for Windows System Locations](#)¹⁰⁰ 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](#)⁹⁶
- [Catalog Structure in StyleVision](#)⁹⁷
- [Customizing your Catalogs](#)⁹⁸
- [Variables for Windows System Locations](#)¹⁰⁰

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

4.6.1 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.

4.6.2 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\Common2023\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.

4.6.3 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.

4.6.4 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\Common2023\Schemas</code>
<code>%ApplicationWritableDataFolder%</code>	<code>C:\ProgramData\Altova</code>
<code>%AltovaCommonFolder%</code>	<code>C:\Program Files\Altova\Common2023</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.

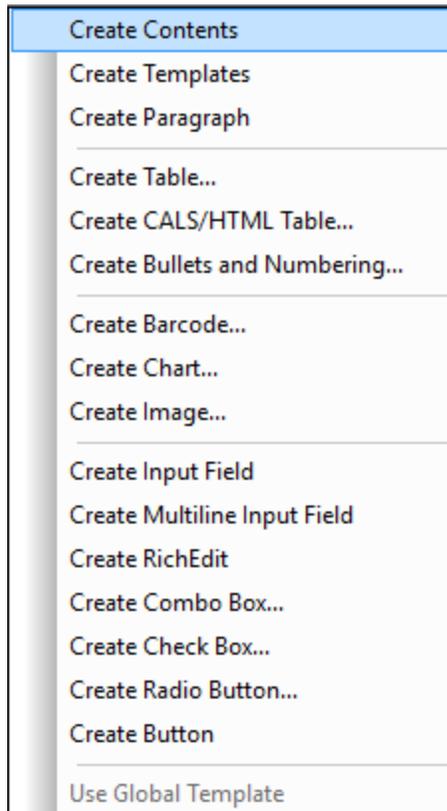
5 SPS Content

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](#)²¹⁵, [global](#)²¹⁵, or [named](#)²²⁵.

- [Inserting XML Content as Text](#)¹⁰³. 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](#)¹⁰⁷
- [User-Defined Templates](#)¹¹²
- [User-Defined Elements, XML Text Blocks](#)¹¹⁵
- [Working with Tables](#)¹¹⁸. 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](#)¹³⁸. 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](#)¹⁴³: 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 \(or Form Controls\)](#)¹⁴⁸. 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](#)¹⁵⁴
- [Barcodes](#)¹⁵⁵
- [Layout Modules](#)¹⁵⁹
- [The Change-To Feature](#)¹⁶⁹. 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*).



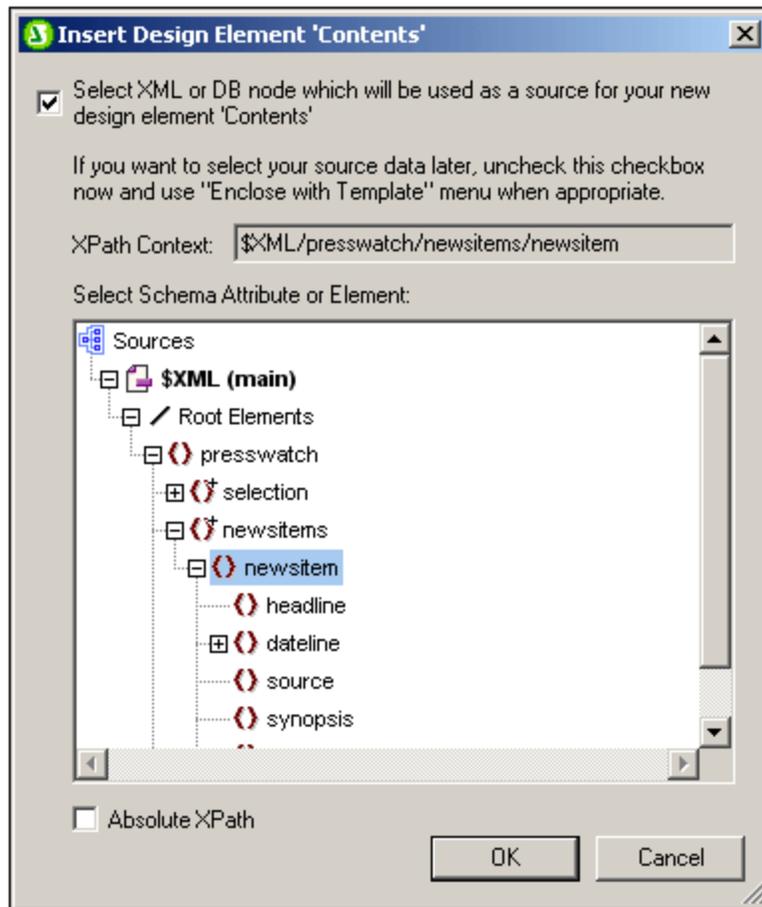
Types of schema nodes

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](#)⁴¹⁸ 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.



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](#) ¹⁰⁵ when the node is inserted.
- How descendant nodes not explicitly included within a node can be included for processing. See [Rest-of-Contents](#) ¹⁰⁶.

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](#) ⁴¹⁶ (*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](#) ²²³ **Enclose With | Special Paragraph** ²²³.

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 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](#)²¹⁵ 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

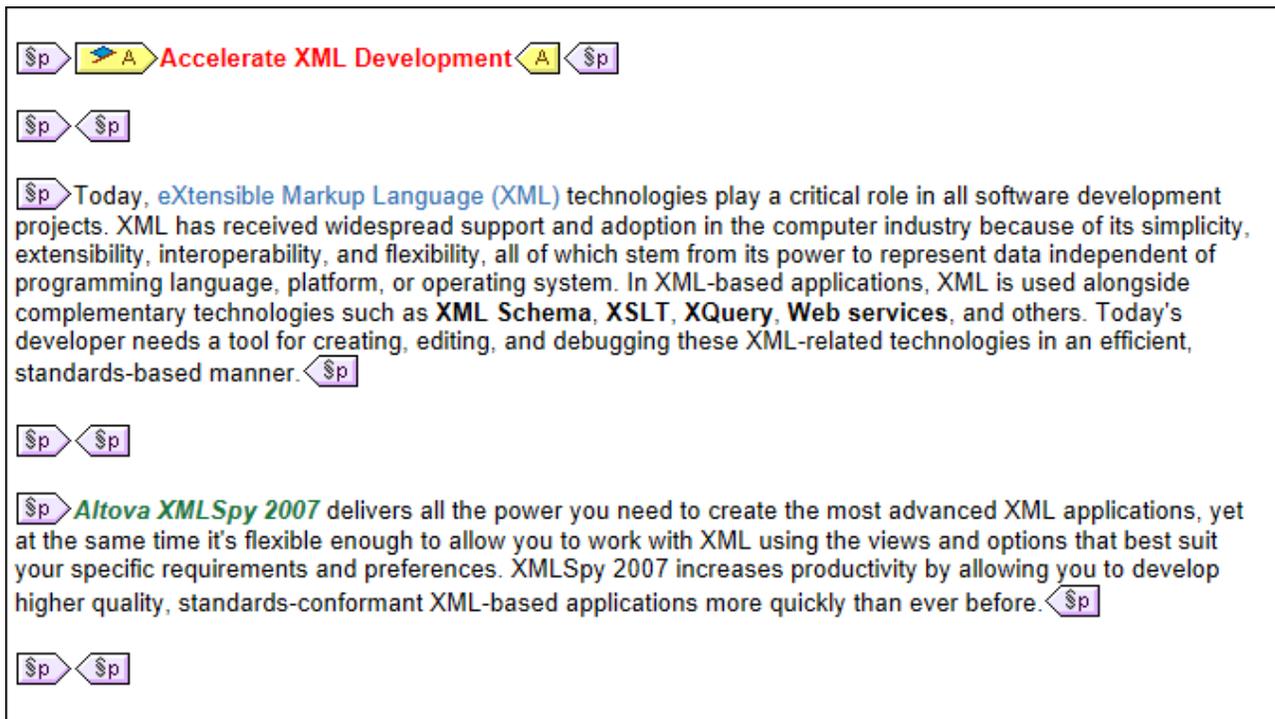
If Microsoft Word 2007+ is installed on your machine, then content can be pasted from Word documents into the design as **static content**. The Word content will be inserted within suitably corresponding design components, and text formatting properties will be carried over from the Word content. For example, text content that is in a Word paragraph block will be inserted within a [Paragraph component](#)¹⁰⁵, 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+](#)⁴²³ 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

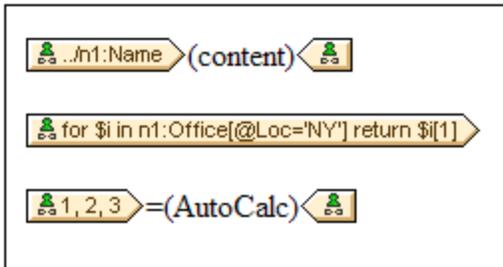
Excel content imported into a design as a static table with text formatting preserved.

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+](#)⁴²³ command.

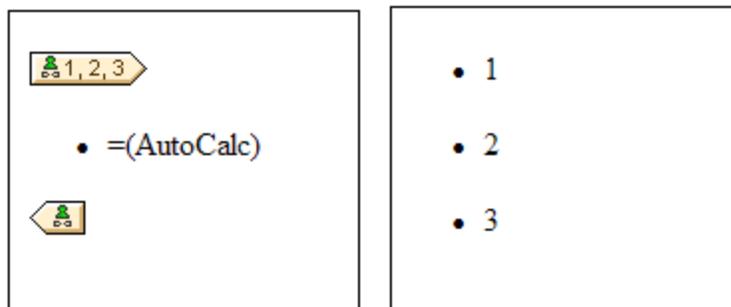
5.4 User-Defined Templates

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.



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. 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:

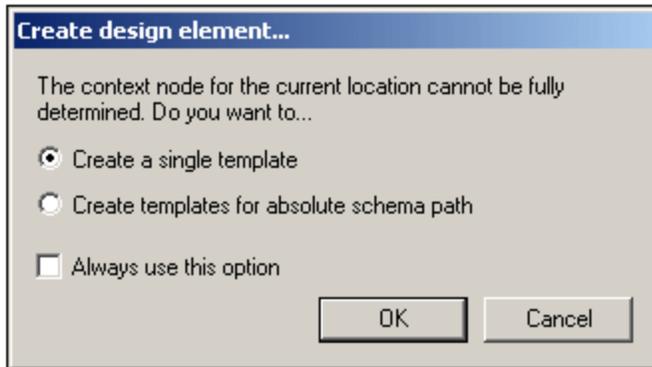
1. Click the **Insert User-Defined Template** icon in the Insert Design Elements toolbar and then click the design location where you wish to insert the template. Alternatively, right-click the design location where you wish to insert the template and, from the context menu that appears, select the **Insert User-Defined Template** command.
2. In the [Edit XPath Expression](#)³⁹⁷ 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](#)²²³.

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](#)⁵¹³.

5.5 User-Defined Elements, XML Text Blocks

[User-Defined Elements](#)¹¹⁵ and [User-Defined XML Text Blocks](#)¹¹⁶ 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

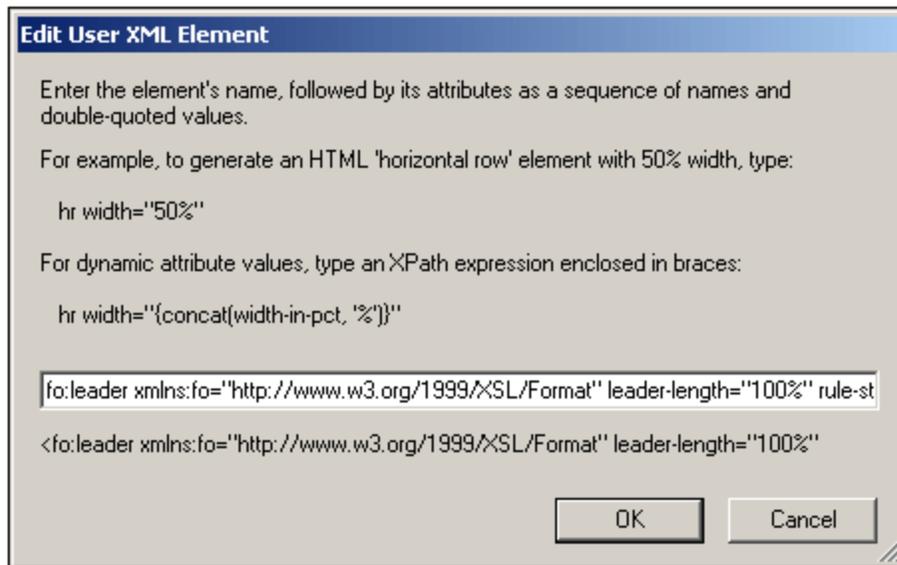
User-Defined Elements are elements that you can generate in the output without these elements needing to be in any of the schema sources of the SPS. This means that an element from any namespace (HTML or XSL-FO for example) can be inserted at any location in the design. SPS design elements can then be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

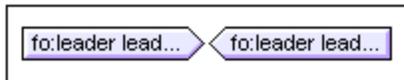
The mechanism for using User-Defined Elements is as follows:

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.

- 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*).



- 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 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:

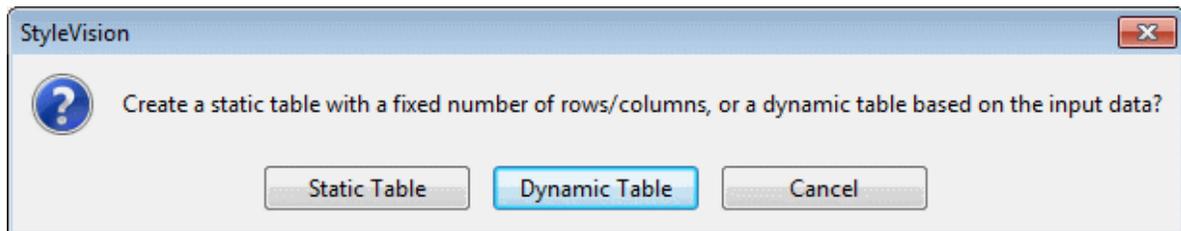
- **The structure** of an SPS table is defined in the SPS. The structure of a CALS/HTML table on the other hand is specified in the schema and must follow that of the CALS/HTML table model; the element names in the schema may, however, be different than those in the CALS or HTML table models.
- **Colspans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

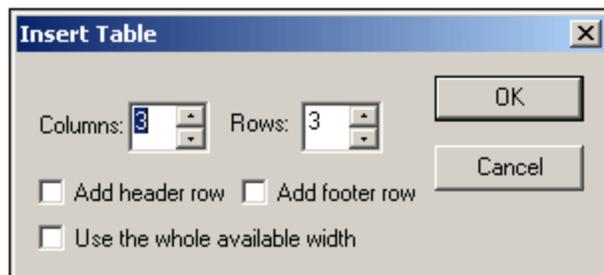
To create a static table, do the following:

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.

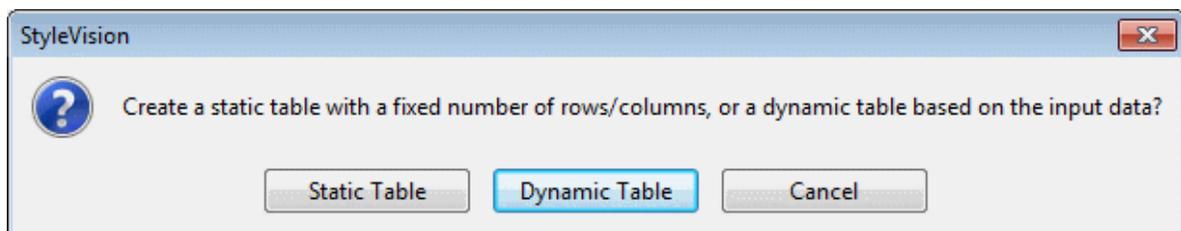
Toolbar table editing icons

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.

5.6.2 Dynamic Tables

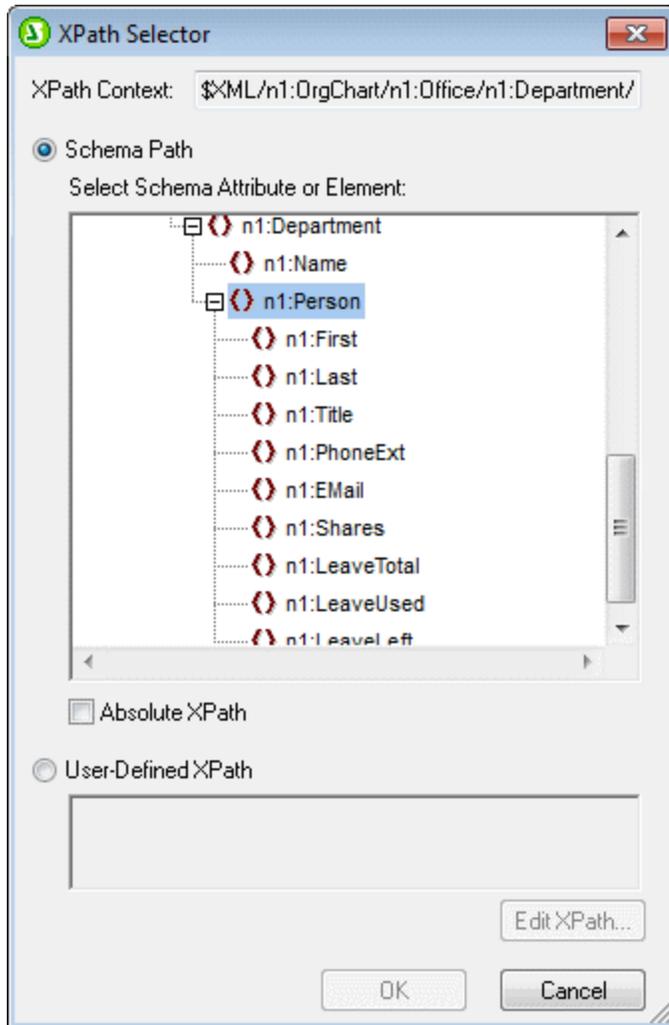
To insert a dynamic table, do the following:

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.



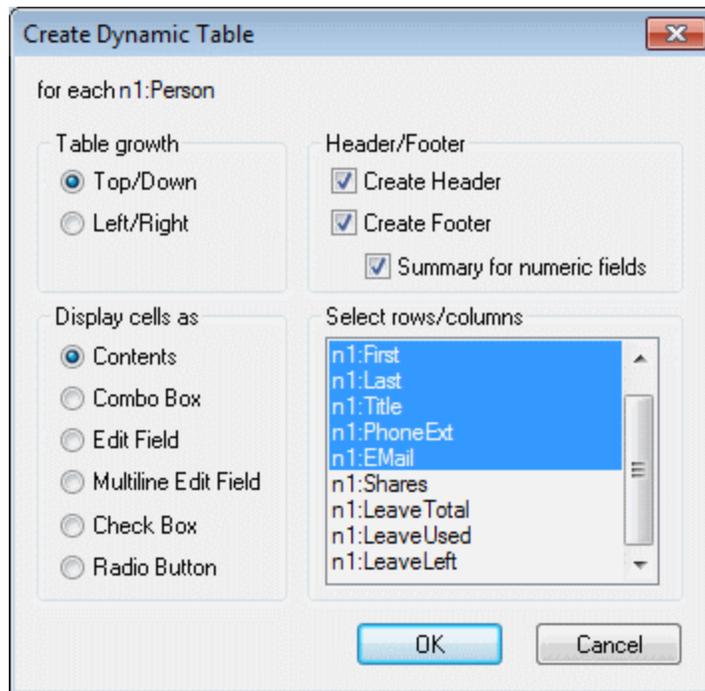
Click **Dynamic Table**.

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.

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.)

First	Last	Email
first (content)	last (content)	email (content)

Clicking the **Edit Conditional Processing** command pops up the [Edit XPath Expression dialog](#)³⁹⁷, 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](#)³⁹⁷ 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**.

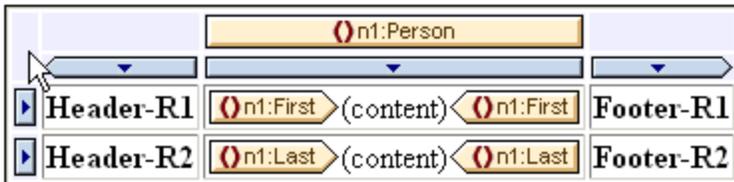
5.6.4 Tables in Design View

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.

Header-C1	Header-C2
n1:First (content)	n1:Last (content)
Footer-C1	Footer-C2

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*).



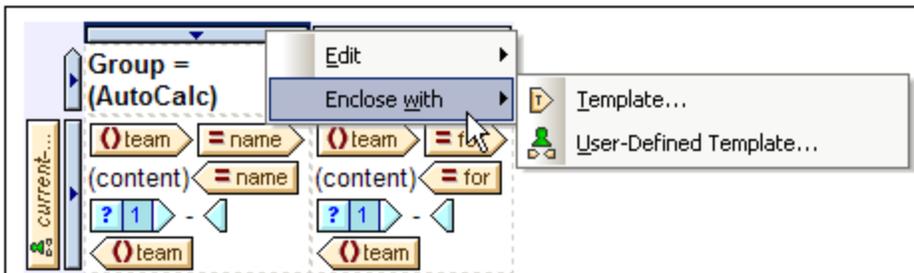
After a column or row or table has been selected, styles and/or properties can be set for the selection in the Styles and Properties Windows.

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](#)²¹⁹. 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](#)²¹⁹ also allows the use of interesting template-match results within the row or column (via Auto-Calculations, for example).

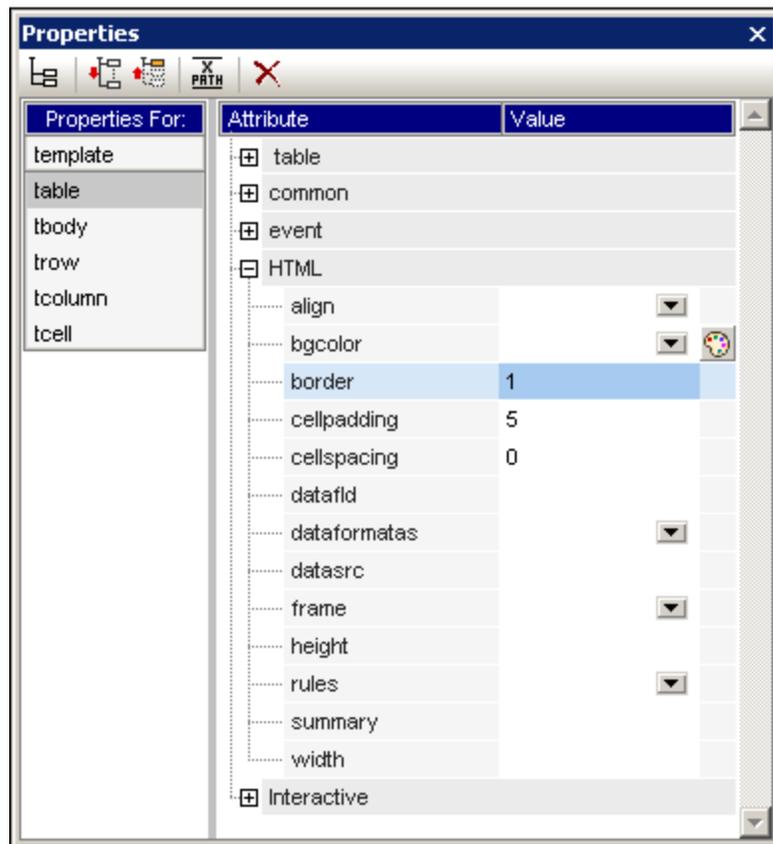
5.6.5 Table Formatting

Static and dynamic tables can be formatted using:

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

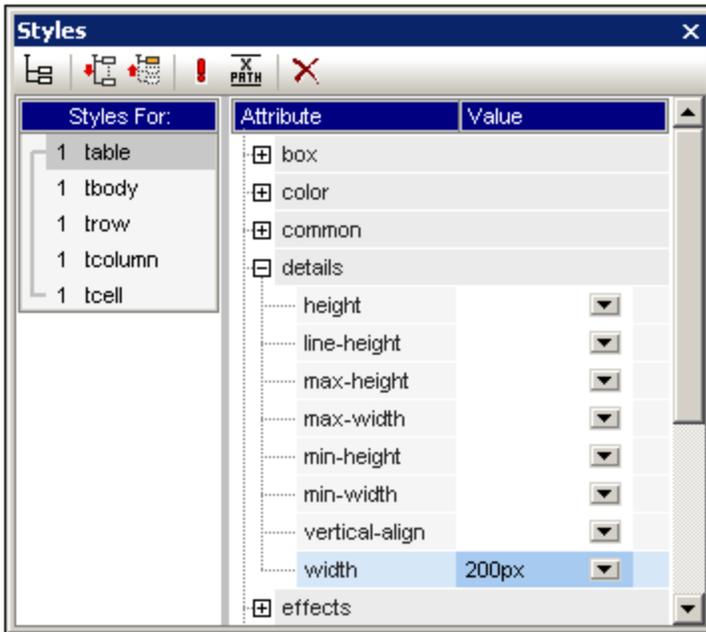
Properties sidebar

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

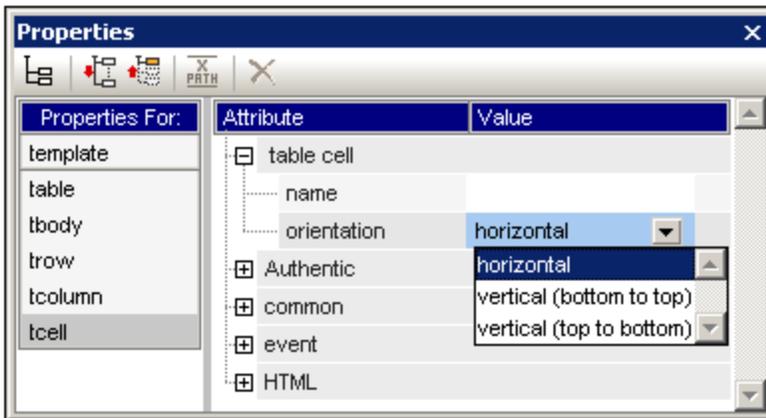
The CSS table formatting properties are available in the Styles sidebar (*screenshot below*). CSS properties are available for the table component and its sub-components (body, row, column, and cell).



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](#)¹⁶⁴.

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](#)¹¹⁵. 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](#)³⁹⁷ 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

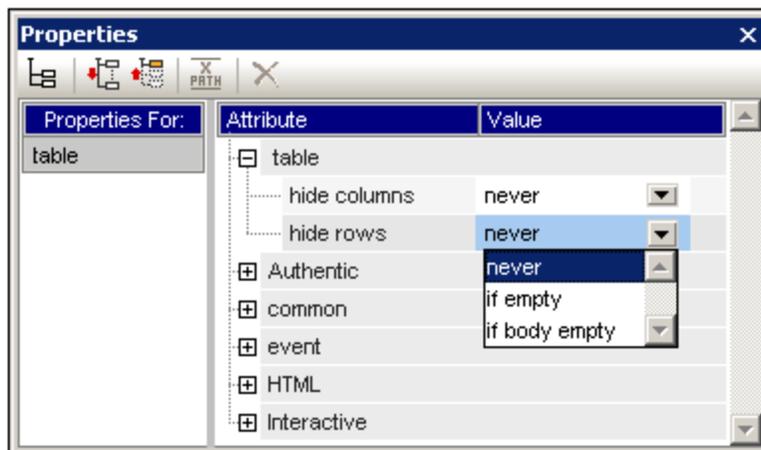
For tables, the following row and column display options are available in the **HTML output only**. These features are **not supported in Authentic View** and they require XSLT 2.0 or XSLT 3.0 to be selected as the XSLT version of the SPS.

- 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.

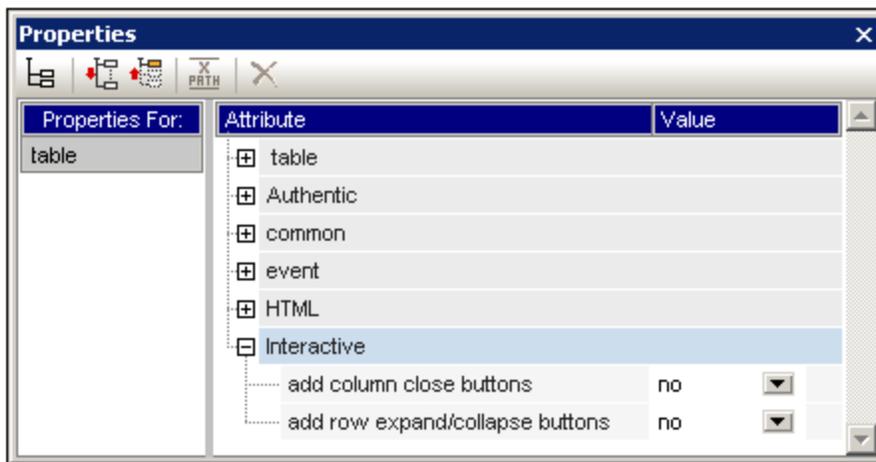
User interaction to hide columns expand/collapse rows

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](#) ¹³⁴
2. [Specify formatting styles for the table](#) ¹³⁶
3. [Insert the CALS/HTML table in the SPS design](#) ¹³⁶

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>
```

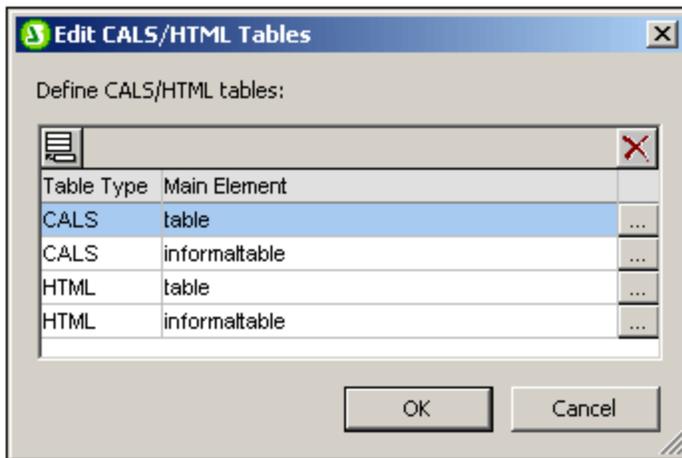
Alternatively, the XML data structure could have a structure corresponding to the HTML table model but different element names than in the HTML table model. For example:

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

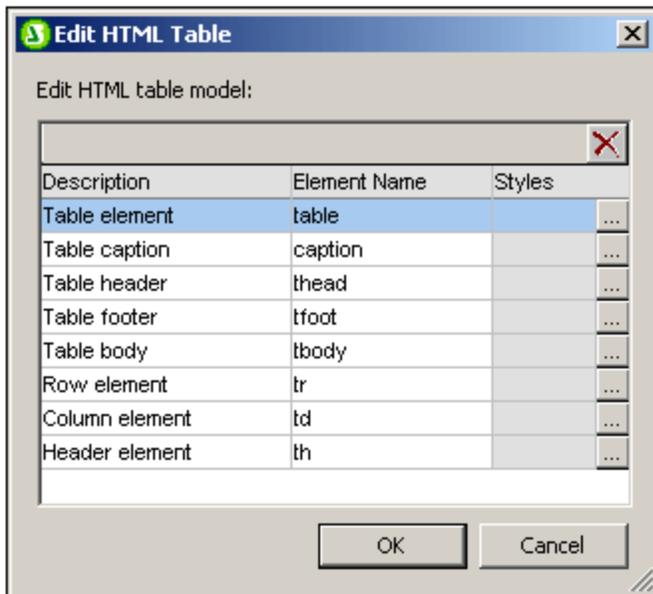
This table structure, which is defined in the XML document, can be used to directly generate a table in the various output formats. To do this you need to define this XML data structure as a CALS or HTML table. If the XML data structure is not defined as a CALS or HTML (the default), the elements in 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.

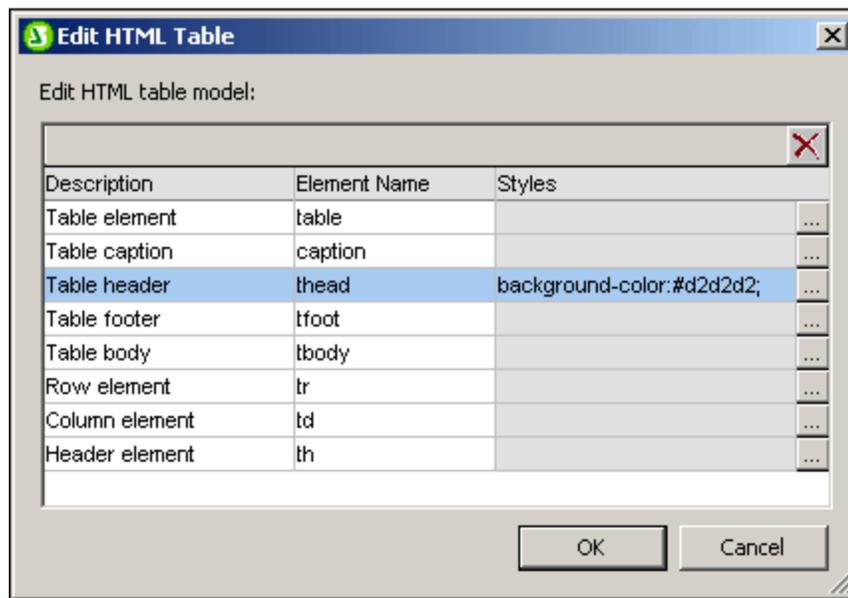
If a CALS/HTML table has been defined and the XML data structure is [correctly inserted](#)¹³⁶ as a CALS/HTML table, then the data structure will be sent to the output as a table. To **remove a CALS/HTML table**

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](#)⁴³ 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.

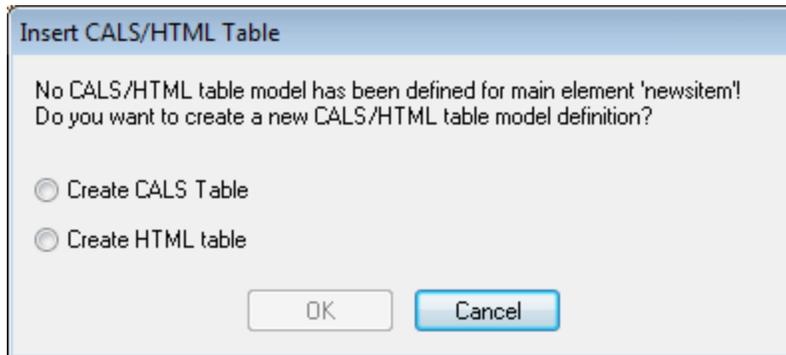
To remove a style that has been assigned to an element in the CALS/HTML table definition, select that element (for example in the screenshot above the `thead` element has been selected) and click the **Delete** button. The styles for that element will be removed.

Inserting a CALS/HTML table in the design

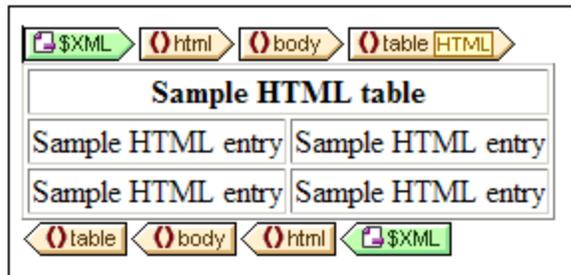
A CALS/HTML table structure can be inserted in the design in two ways:

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](#)¹³⁴, 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](#)²¹⁵ 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:

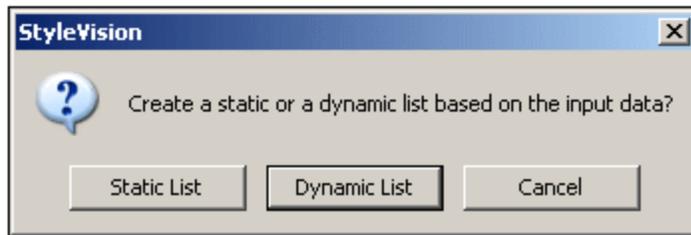
- [Static lists](#)¹³⁸, which are lists, the contents of which are entered directly in the SPS. The list structure is not dynamically derived from the structure of the XML document.
- [Dynamic lists](#)¹⁴⁰, which are lists that derive their structure and contents dynamically from the XML document.

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

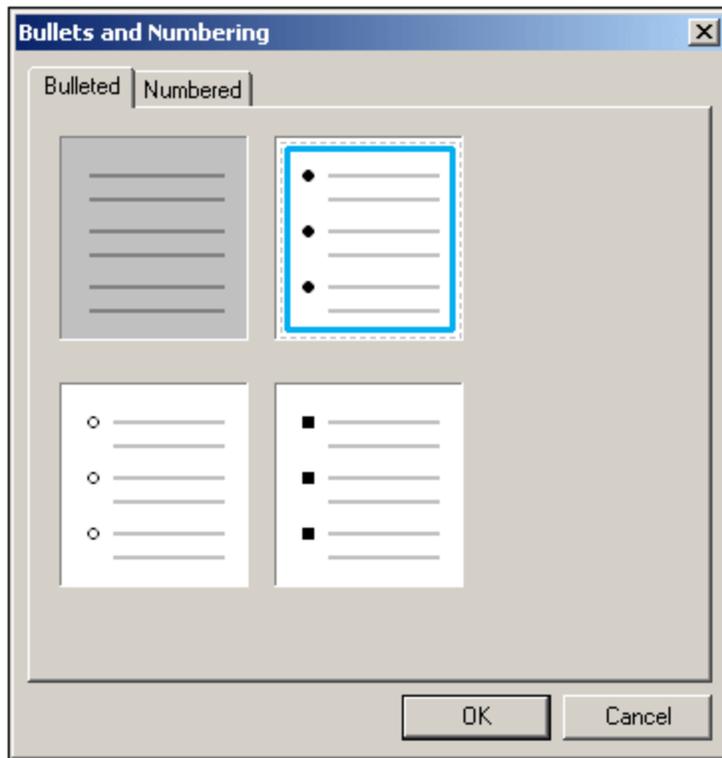
5.7.1 Static Lists

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](#)⁴⁶⁴ menu command (or click the Bullets and Numbering icon in the [Insert Design Elements toolbar](#)⁴¹⁸). 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](#)⁴⁶⁴ 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](#)⁴¹⁶. 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](#)⁴⁷⁷, 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](#)⁴⁷⁷ command can also be accessed via the context menu.

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.

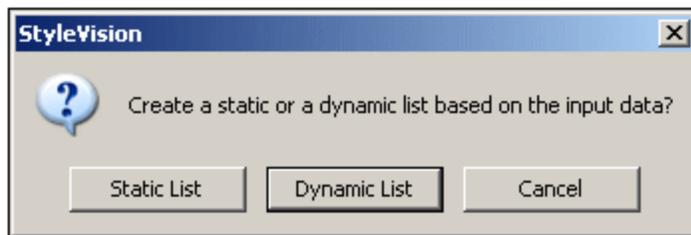
General usage mechanism

- 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.

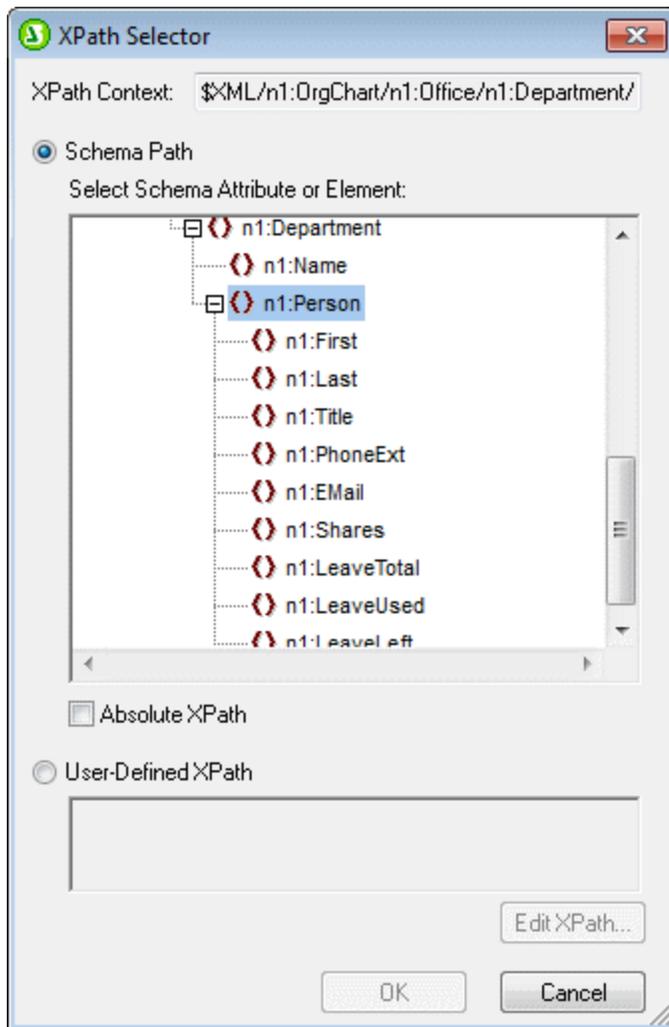
Creating a dynamic list

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** ⁴⁶⁴ menu command. This pops up a dialog asking whether you wish to create a static list or dynamic list (*screenshot below*).

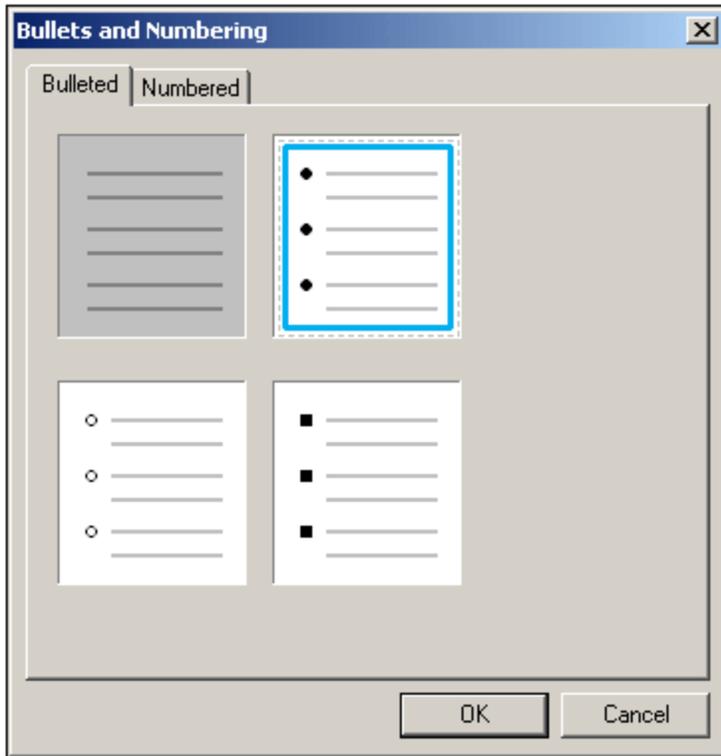


2. Click **Dynamic List**. This pops up the XPath Selector dialog (*screenshot below*).
3. In the XPath Selector dialog, notice that the XPath Context is the context of the insertion location, and that it cannot be changed in the dialog. Select the node that is to be created as the dynamic list. In the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



5.8 Graphics

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](#)¹⁴³. What type of images are supported in the various outputs are listed in the section [Image Types and Output](#)¹⁴⁵.

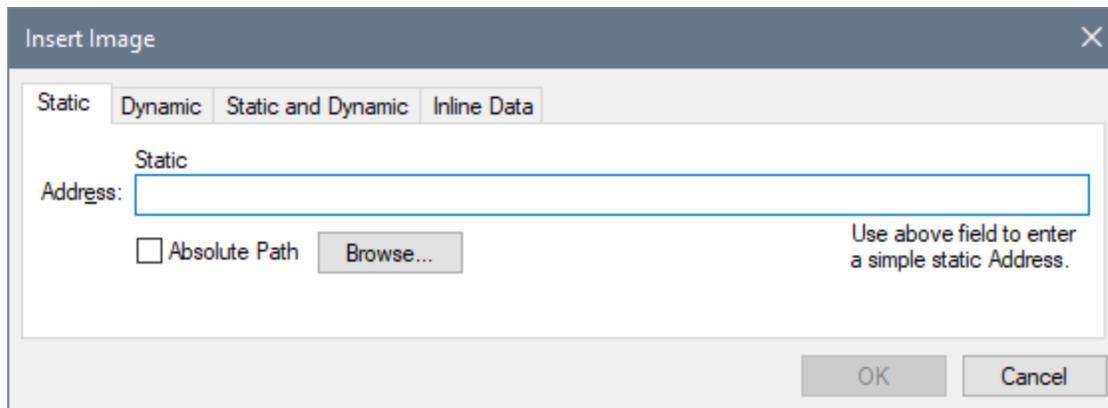
Image properties

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 the output documents; in Design View, inserted images are indicated with thumbnails or placeholders.

To insert an image, click the [Insert | Image](#)⁴⁶¹ 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

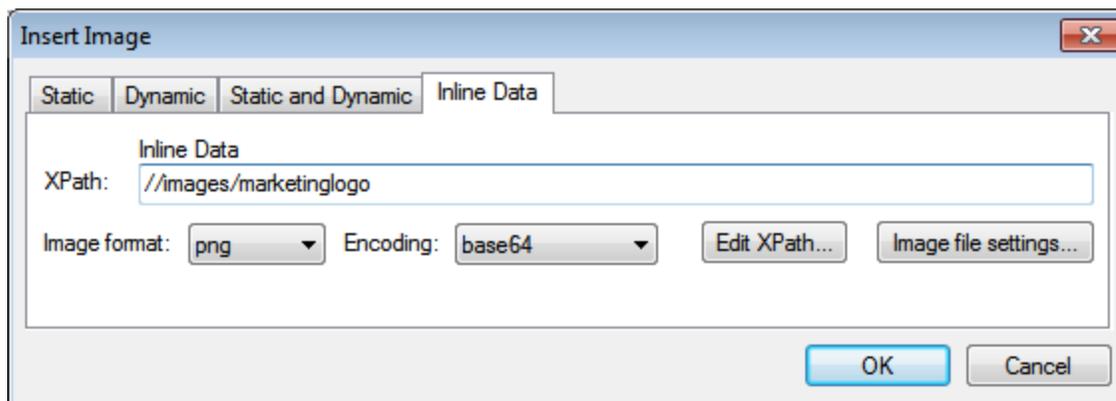
An image file is inserted in the design by specifying its URI. This file is accessed at runtime and placed in the document. There are three ways in which the URI of the image can be entered in the Insert Image dialog (*screenshot above*):

- In the Static tab, the URI is entered directly as an absolute or relative URI. For example, `nanonull.gif` (*relative URI*; [see section below](#)¹⁴⁴), 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](#)³³⁸. 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](#)³³⁸.
- 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.

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, you can set, in the [Properties dialog](#)⁴⁴³, 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.

- 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:

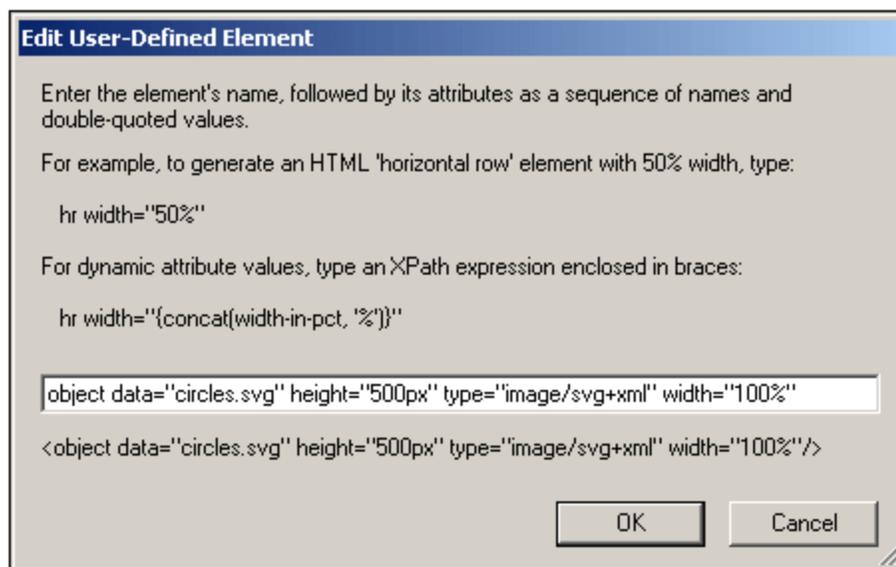
```
C:\Documents and Settings\\My
  Documents\Altova\StyleVision2023\StyleVisionExamples\Tutorial\Images
```

SVG images in HTML

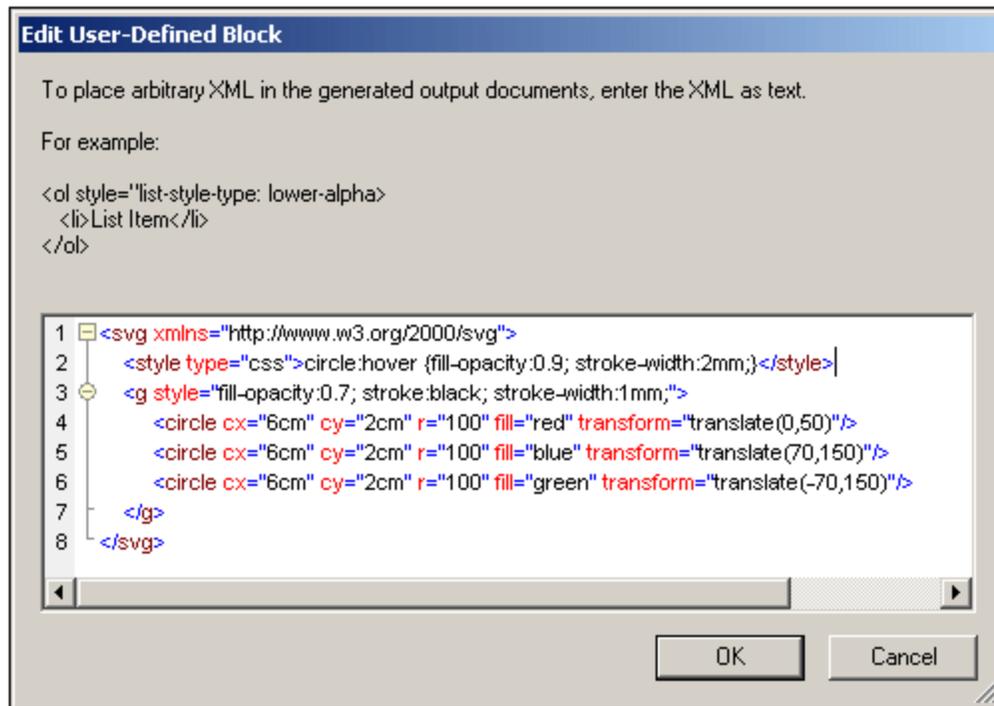
When an external SVG file with code for mouse events is used as an image, the SVG file is rendered within the image and will no longer be interactive. This limitation can be overcome by using the external SVG image file as an object or by including the SVG code fragment as a User-Defined XML Block.

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

1. External SVG [inserted as image](#)¹⁴³: 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](#)¹¹⁵ (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](#)¹¹⁶. 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](#)²³: C:\Documents and Settings\

- The second part contains a table showing which image formats are supported in the various StyleVision output formats. Note that the RTF, PDF and Word 2007+ output formats are available only in the Enterprise Edition and Professional Edition (RTF only) of StyleVision. 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). In the HTML output, the data-entry device is rendered as an object that is the same as that displayed in Design View, or a near-equivalent. Note that data-entry devices 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.

Given below is a list of the data-entry devices available in StyleVision.

Data-Entry Device	Output to Plain Text Format
Input Field (Text Box)	Text entered by user
Multiline Input Field	Text entered by user
Combo box	Selected value
Check box	[] (unselected); [x] (selected)
Radio button	() (unselected); (o) (selected)
Button	Button text

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 cannot be entered in data-entry devices in the HTML output. In the HTML output, data-entry devices are merely used as an alternative way of presenting content.
- 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 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.

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 content of that node is displayed in the input field or multiline input field.

Editing the properties of input fields

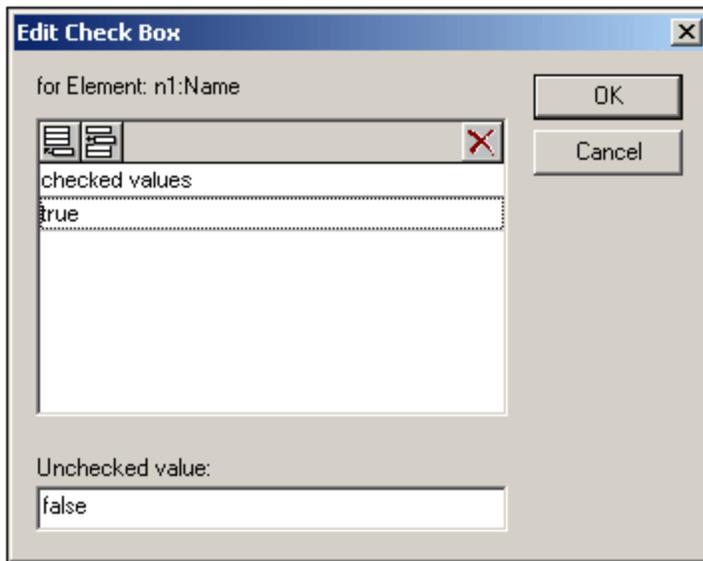
You can modify the HTML properties of input fields by selecting the input field and then modifying its *HTML* properties in the Properties sidebar.

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. In Basic edition, you can leave the settings in the Edit Check Box dialog at their default settings (since Basic edition does not support Authentic View, as a result of which no value can be entered in the XML file.)



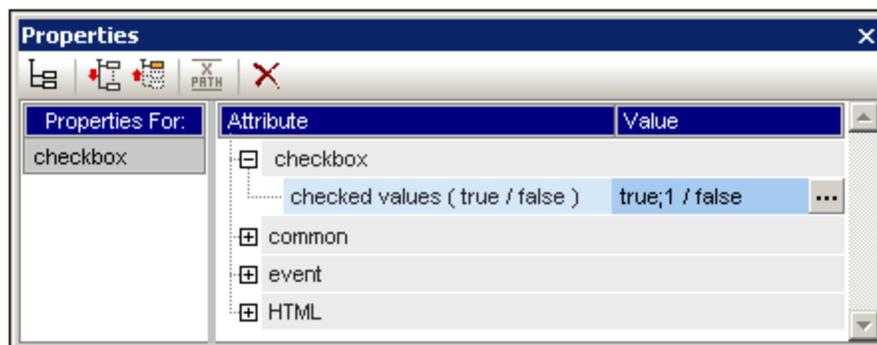
In the above screenshot, an element called `Name` has been created as a check box. If the Authentic View user checks the check box, a value of `true` will be entered as the value of the element `Name`. If the value is unchecked, then the value `false` is entered as the XML value of `Name` (as defined in the dialog).

Note: 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

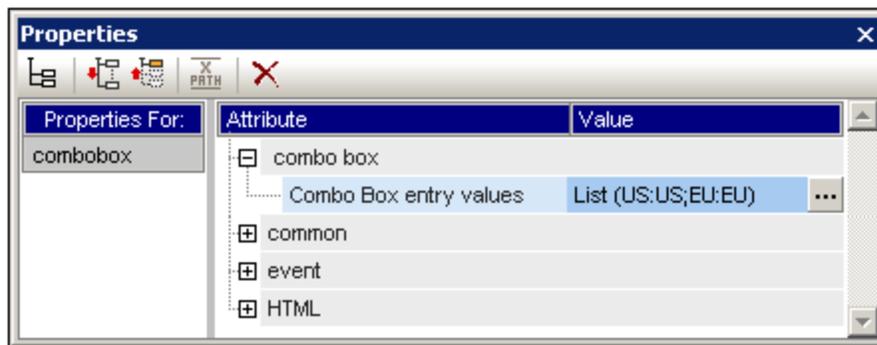
A combo box presents items in a dropdown list. The items in the list can be selected in one of the three ways listed below. This list can be used in the generated HTML document for any required purpose; for example the generated HTML can be post-processed so that the combo box list provides entries for an HTML form.

- From the schema enumerations for the selected node.
- From a list defined in the Edit Combo Box dialog. You enter the visible entry and the corresponding XML value, which may be different. The XML value applies to the Enterprise and Professional editions, and refers to the XML value to which the Authentic View user-selection maps. Basic edition users can leave this column blank (since Authentic View is not supported in Basic edition).
- 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. 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 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:

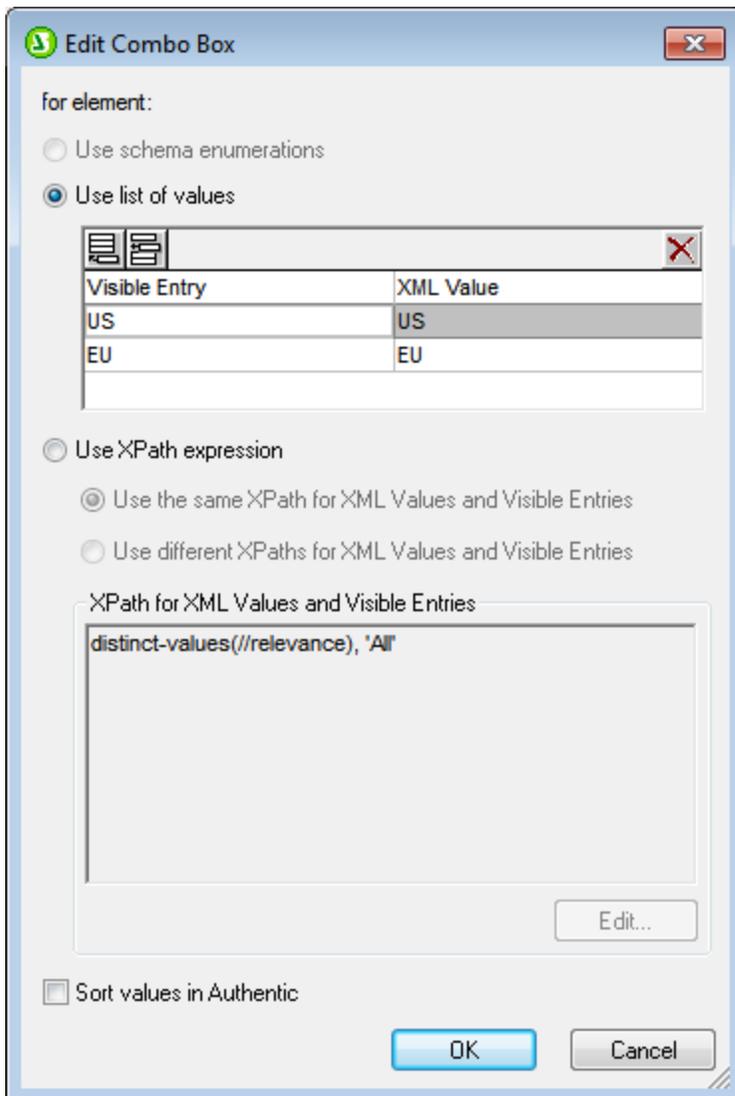
1. Select the combo box in the design.
2. In the Properties sidebar, select the combo box item and then the *combo box* group of properties (see *screenshot below*).



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

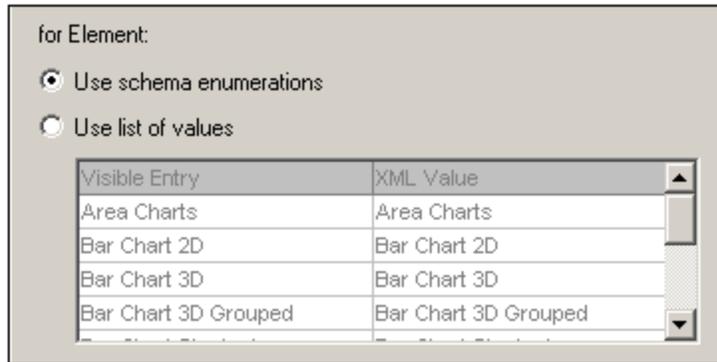
Using the Edit Combo Box dialog

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.
- 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 and buttons can be useful for input into forms or triggering events in the HTML output. The latter is done by associating scripts with the button event.

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](#)²⁹⁸, 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](#)¹⁵⁵ and specify its [properties](#)¹⁵⁷.



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.

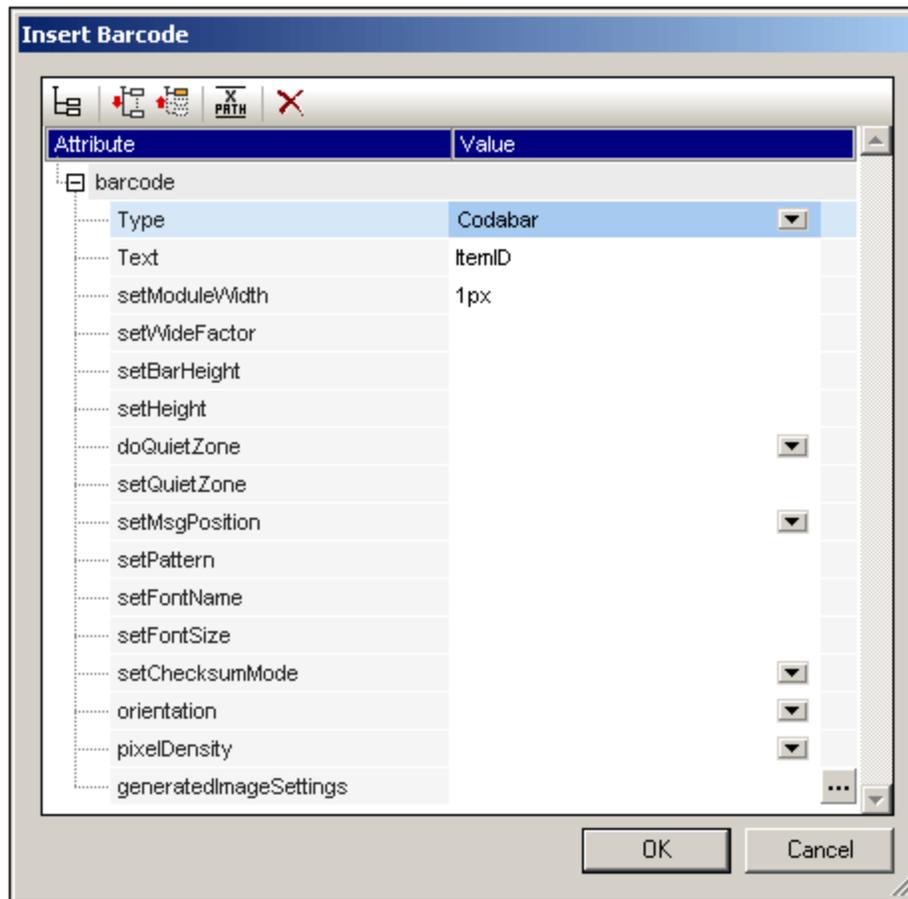
Important: For barcodes to be generated in the output, you **must use Altova's XSLT processor** to generate the output. This is because the barcodes in an SPS are generated by calling special Java extension functions that are not part of the XSLT standard. Altova's XSLT processors support these specific 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](#)¹⁵⁷. 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](#)⁴⁷ 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](#)³⁹⁷. 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

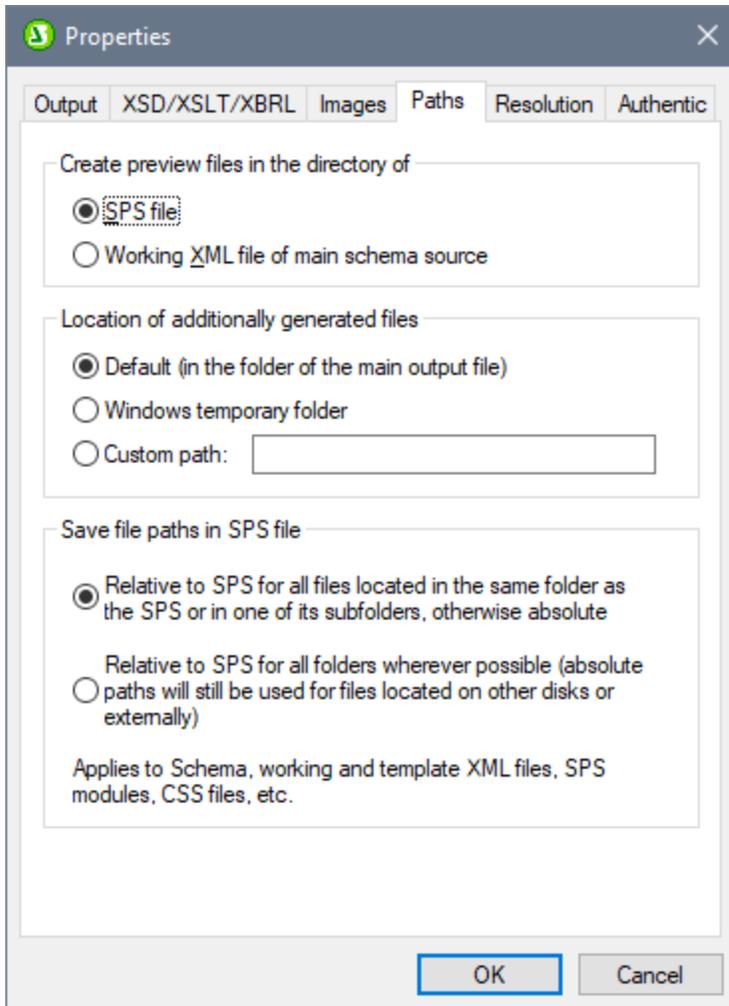
Patterns are used to make the input message string more readable in the barcode. In the pattern, each character of the input message text is represented by the underscore "_". Any other characters included in the pattern are inserted at the corresponding locations in the output message text. The backslash "\" 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](#)¹⁵⁹. 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](#)¹⁶² 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](#)¹⁶⁶ 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](#)⁴²³ 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](#)¹⁶⁰ 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](#)¹⁶⁰ section below for details.
- A [layout grid](#)¹⁶⁰ and a [zoom feature](#)¹⁶¹ make the positioning of objects in the Layout Container easier.
- It can have [style properties](#)¹⁶¹, such as borders, background colors, font-properties for the whole container, etc.
- It can [contain Layout Boxes and Lines](#)¹⁶¹, but no other design element. (All design elements must be placed within Layout Boxes.)
- It can contain a [blueprint](#)¹⁶², 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

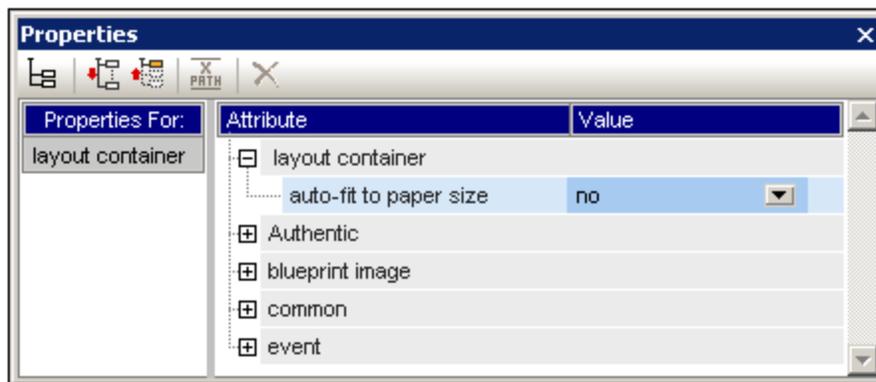
To insert a Layout Container, click the **Insert Layout Container** icon in the [Insert Design Elements toolbar](#)⁴¹⁸ and click the location in the design where the Layout Container is to be inserted. A dialog appears asking whether you wish to auto-fit the Layout Container to the page. If you click **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 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.

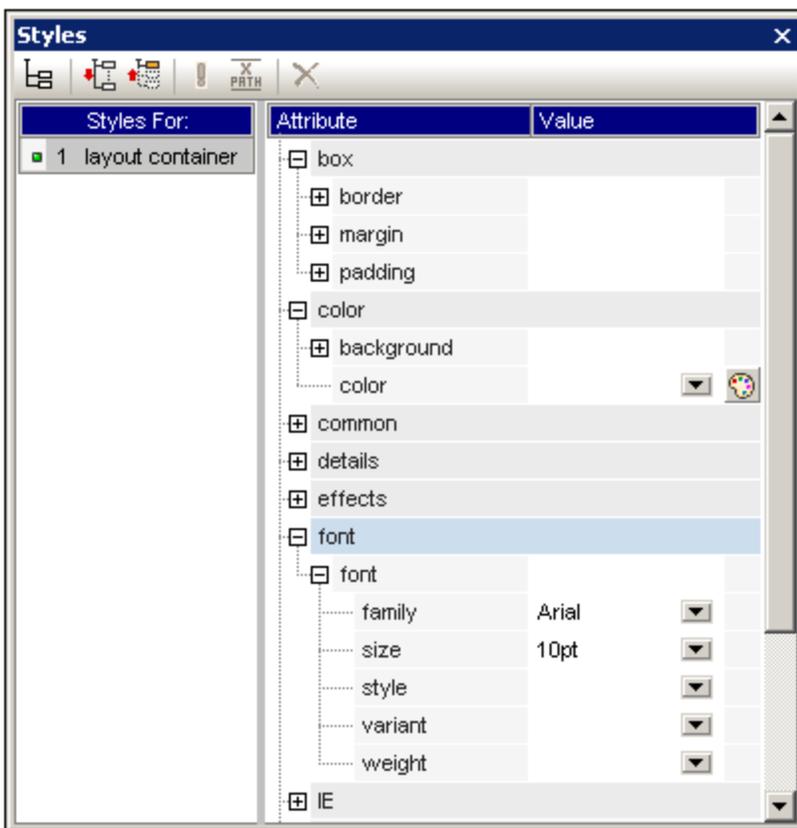
Zooming

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).

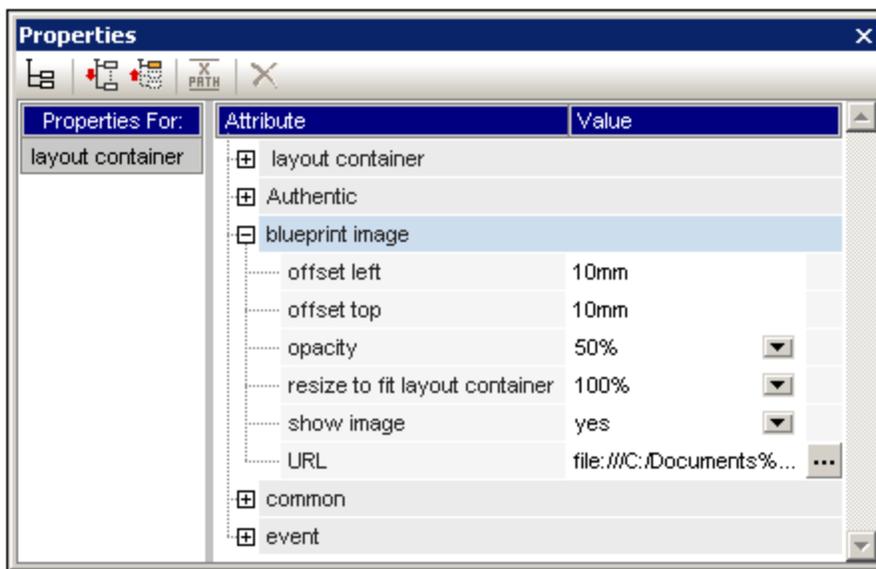
Layout Container contents

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](#) ¹⁶³
- [Selecting and moving Layout Boxes](#) ¹⁶³
- [Modifying the size of the Layout Box](#) ¹⁶³
- [Defining Layout Box style properties](#) ¹⁶⁴

- [Inserting content in the Layout Box](#) ¹⁶⁴
- [Stacking order of Layout Boxes](#) ¹⁶⁵

Inserting a Layout Box

A Layout Box can be inserted only in a [Layout Container](#) ¹⁵⁹. To add a Layout Box, first click the Insert Layout Box icon in the [Insert Design Elements](#) ⁴¹⁸ 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.

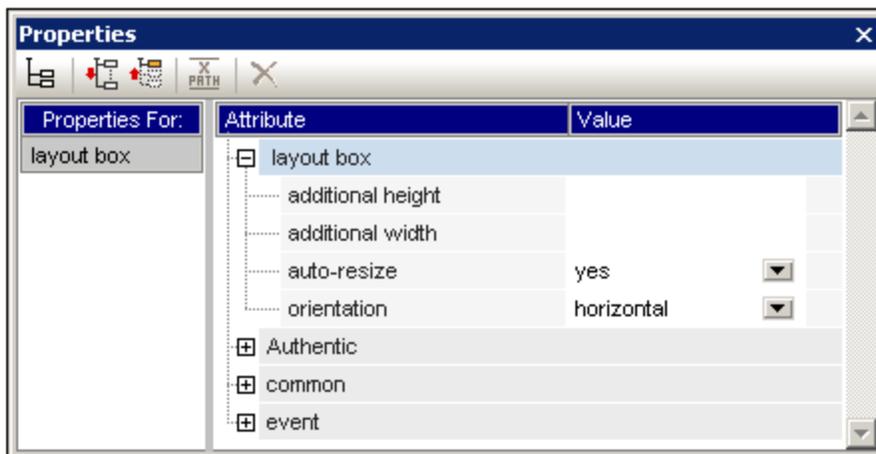
Selecting and moving a Layout Box

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 the 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](#) ¹⁶³. 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.

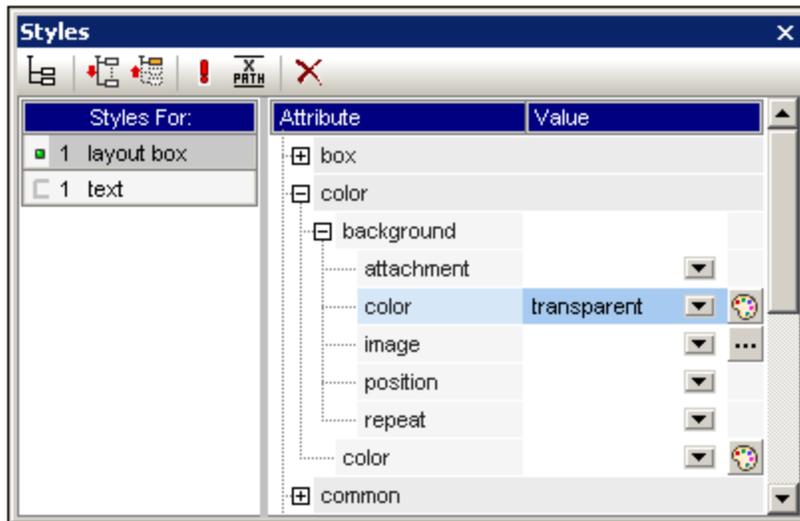


The *Additional Height* and *Additional Width* properties give the lengths that are additional to the optimal dimensions as determined by auto-resizing. The additional lengths are obtained when a Layout Box is manually resized. Conversely, by changing the values of these two properties, the size of the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

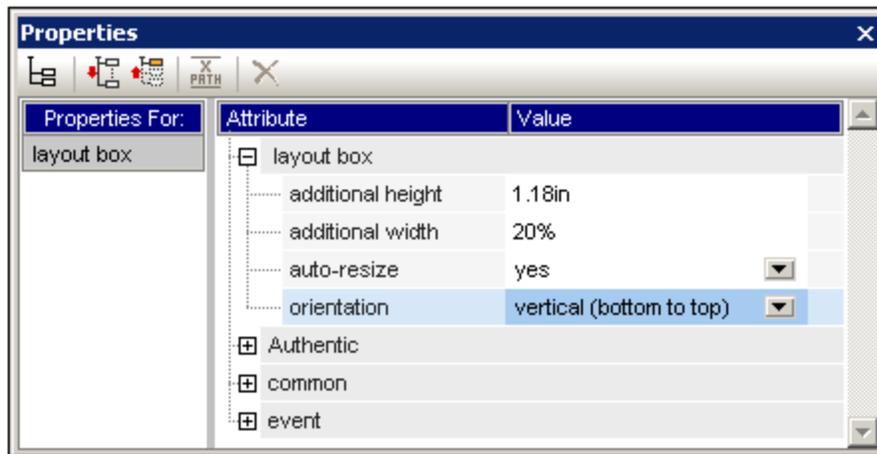


Note: The *background-color* value of `transparent` can be selected in the dropdown list of the property's combo box (it is not available in the color palette). The significance of this value in a situation where the Layout Box is part of a stack is explained below.

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](#)¹⁵⁹ nor a [Layout Line](#)¹⁶⁶ 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.

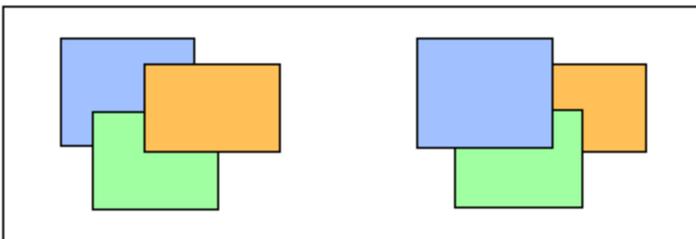


Note the following points:

- 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](#) ¹²⁸.

Stacking order of Layout Boxes

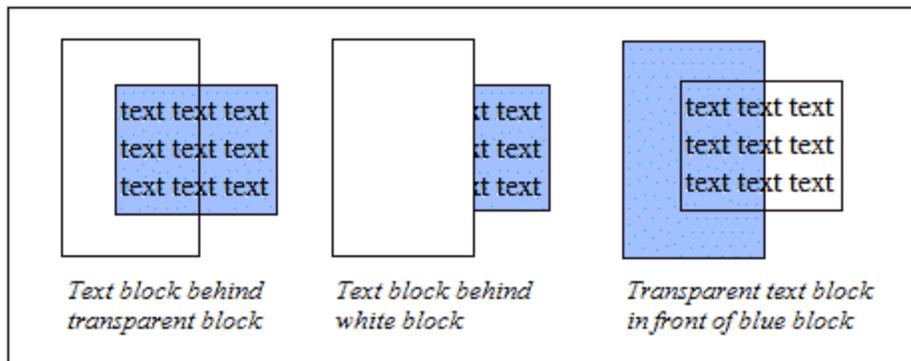
Layout Boxes can be placed one over the other. When one Layout Box is placed on top of another, then, if it is opaque, it hides that part of the Layout Box which it covers. This behavior can be extended to a stack of several Layout Boxes. In such a stack, only the topmost Layout Box will be fully visible; 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](#)¹⁶⁶ 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](#)¹⁶⁶ (but not in Layout Boxes), then [selected, re-sized and moved](#)¹⁶⁶ around within the Layout Container, [assigned properties](#)¹⁶⁷, and [moved backwards and forwards in a stack of layout items](#)¹⁶⁷ 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](#)⁴¹⁸ 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.

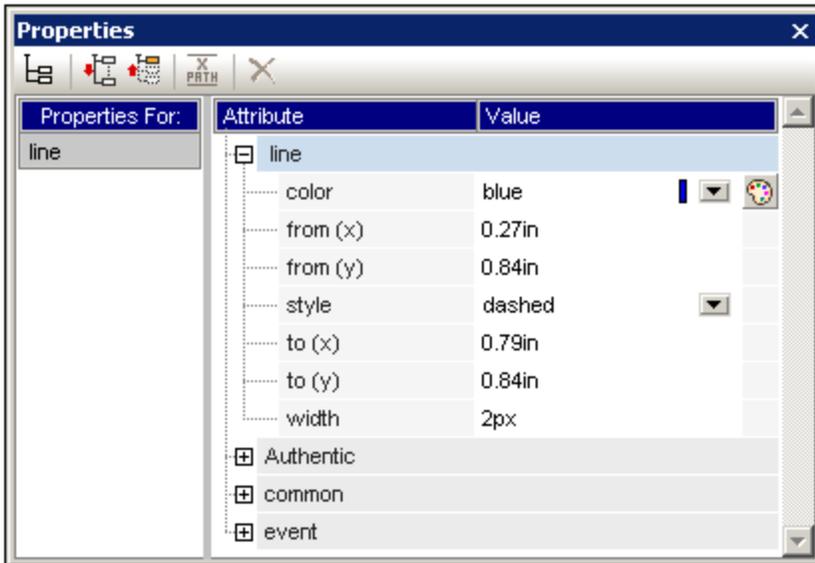
Selecting, moving, and sizing a Line

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.

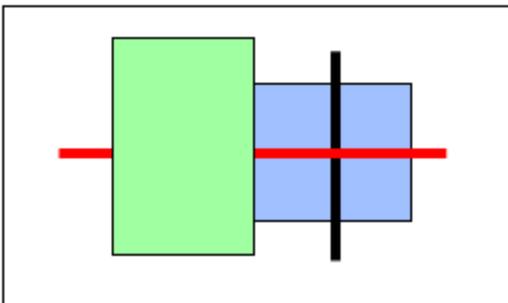


The following Line properties can be edited in the Properties sidebar:

- **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

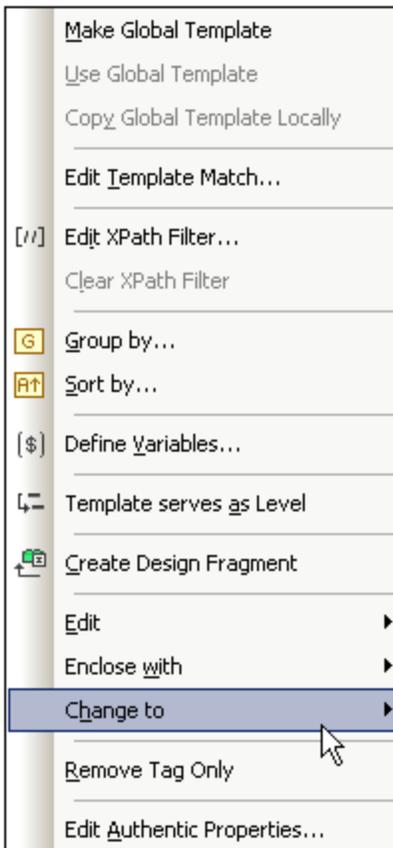
Either a node or its contents can be changed. In the image below left, the node is selected. In the image at right, the node's contents are selected.



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.

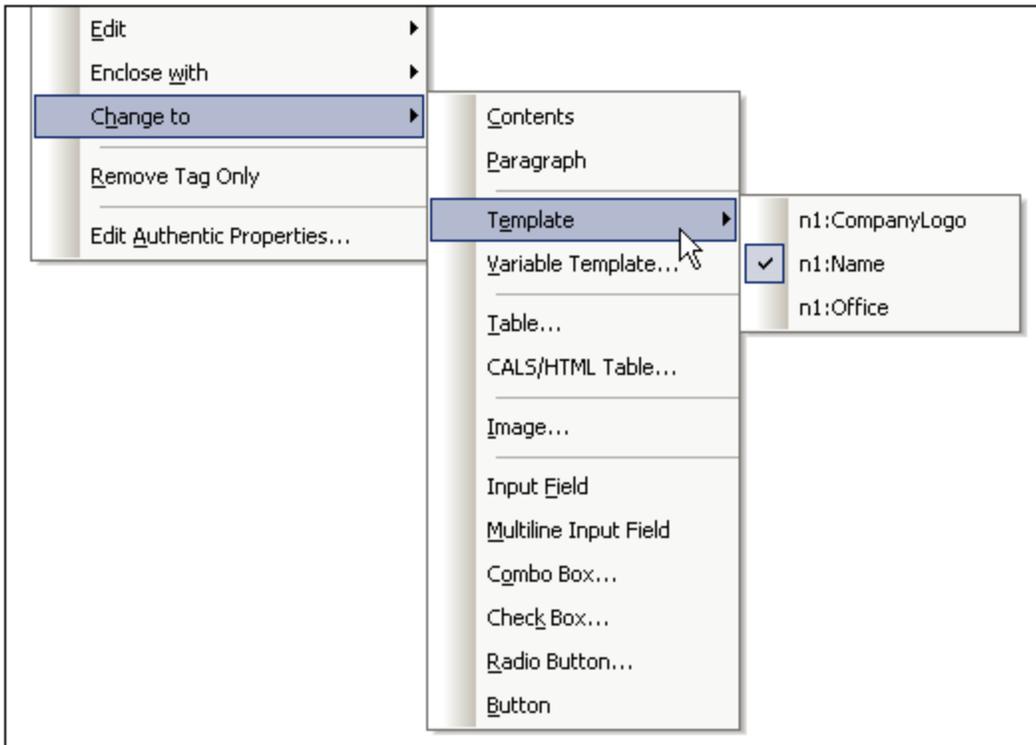
The Change-To command

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.

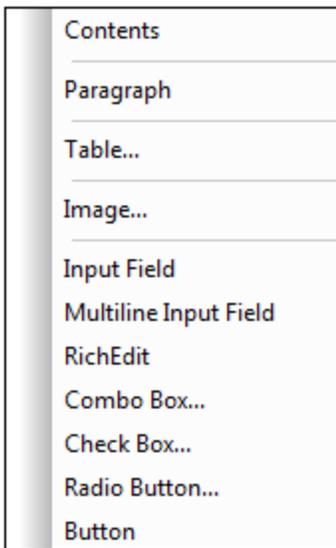


If the selected node has a content model that does not match that described in the template, there will be structural inconsistencies. Such inconsistencies are errors and are indicated with red strikethroughs in the tags of nodes that are invalid.

You can also change the template-match to match, not a node, but a [variable template](#)²²².

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](#)¹⁷⁴
- [Modular SPSs](#)²⁰¹
- [Templates and Design Fragments](#)²¹⁵

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](#)¹⁷⁴.

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](#)²⁰¹.

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](#)²¹⁵.

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](#)²²⁵.

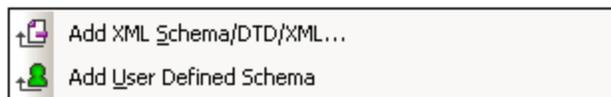
6.1 Schema Sources

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](#)³², 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.



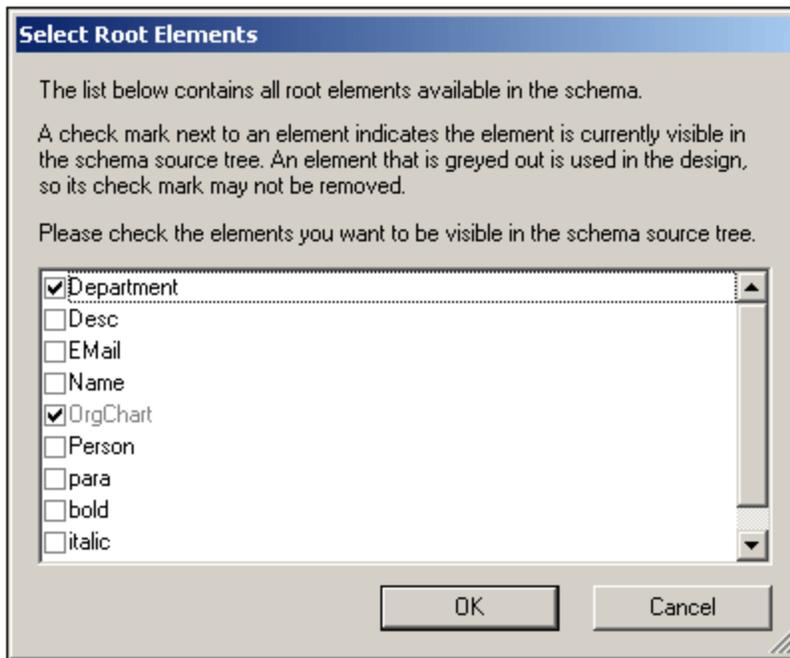
The schema source can be selected from a file or be user-defined. An important point to consider is whether you will be using global templates, and whether elements you wish to create as global templates are defined as global elements in the schema. When adding a DTD from file, remember that all elements 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.

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](#)²¹, then multiple root elements ([document elements](#)²¹) 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](#)²¹), 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](#)²¹) 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:

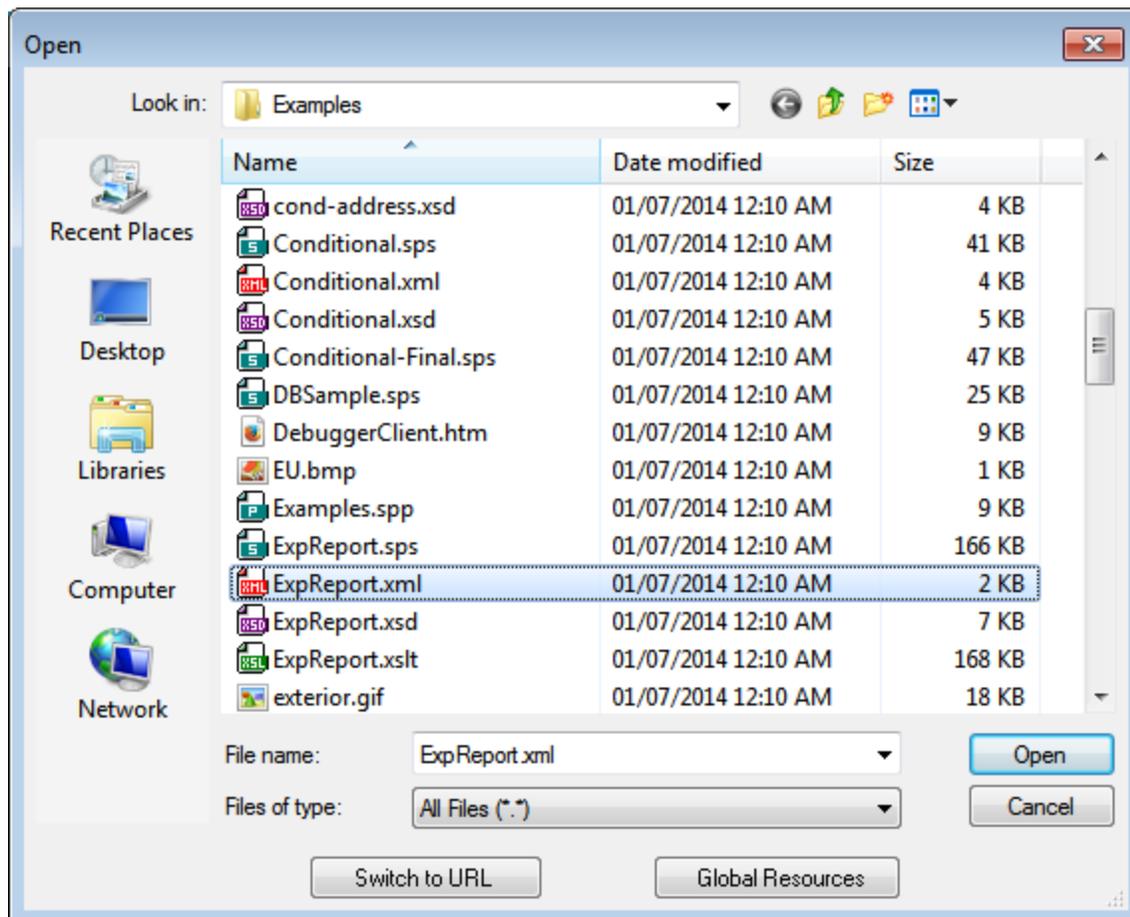
- The XML Schema or DTD is created as a schema source directly when the SPS is created (**File | New | New from XML Schema / DTD / XML**).
- The XML Schema or DTD is added to an empty SPS (in the [Design Overview sidebar](#)³²).

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

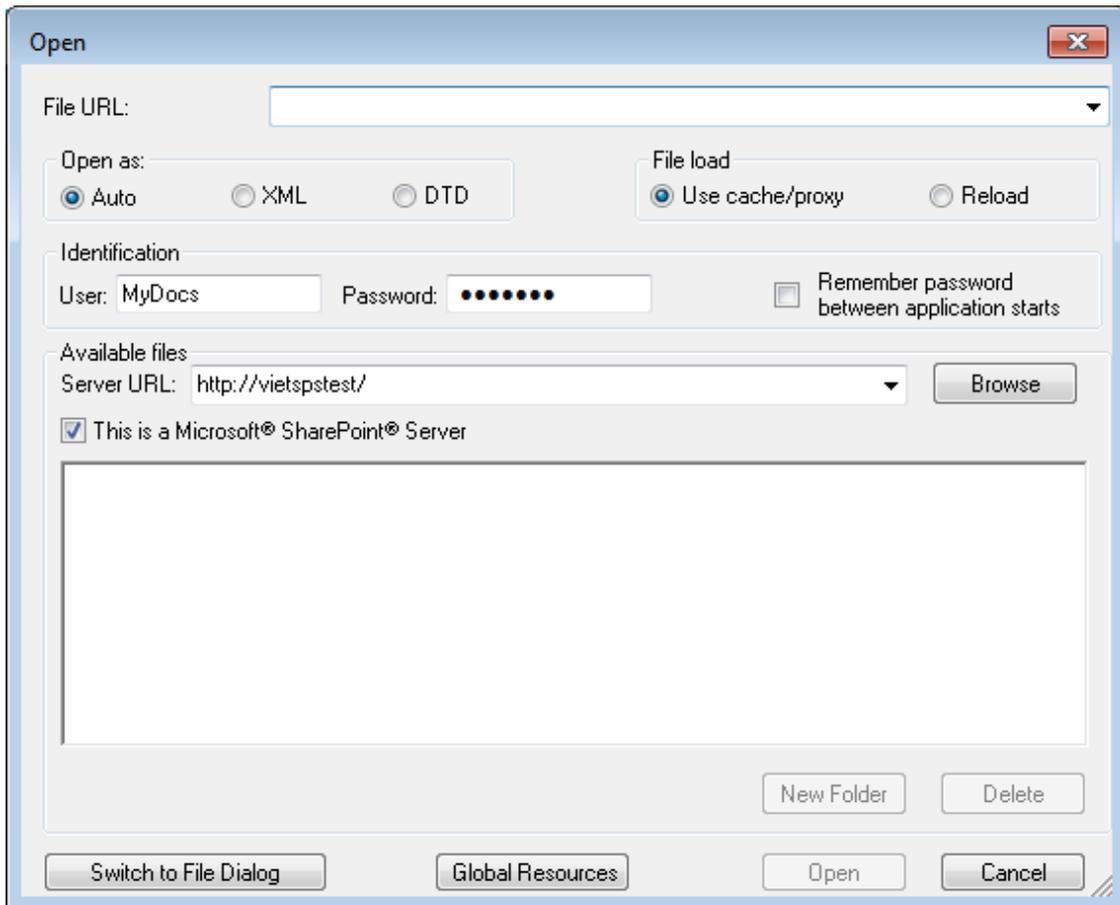
In several File Open and File Save dialogs, you can choose to select the required file or save a file via a URL or a global resource (see *screenshot below*). Click **Switch to URL** or **Global Resource** to go to one of these selection processes.



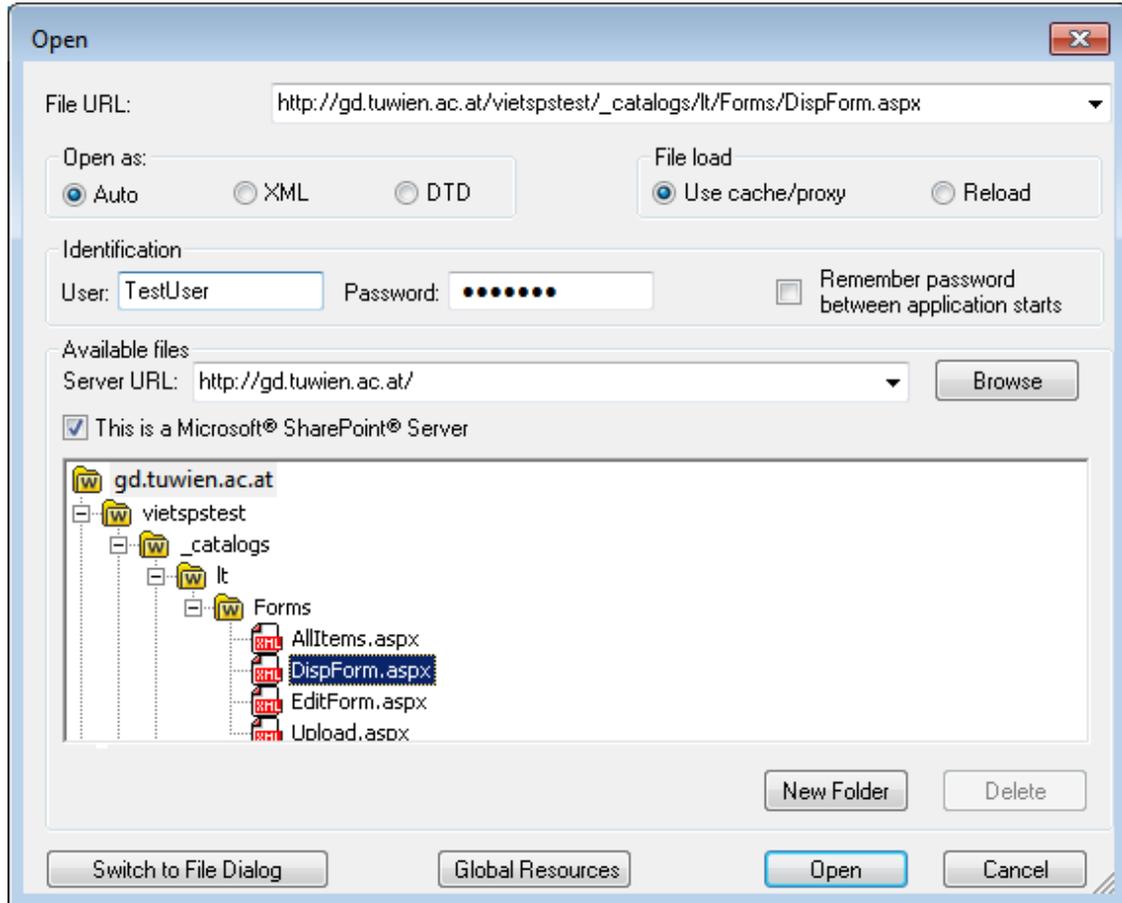
Selecting files via URLs

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.

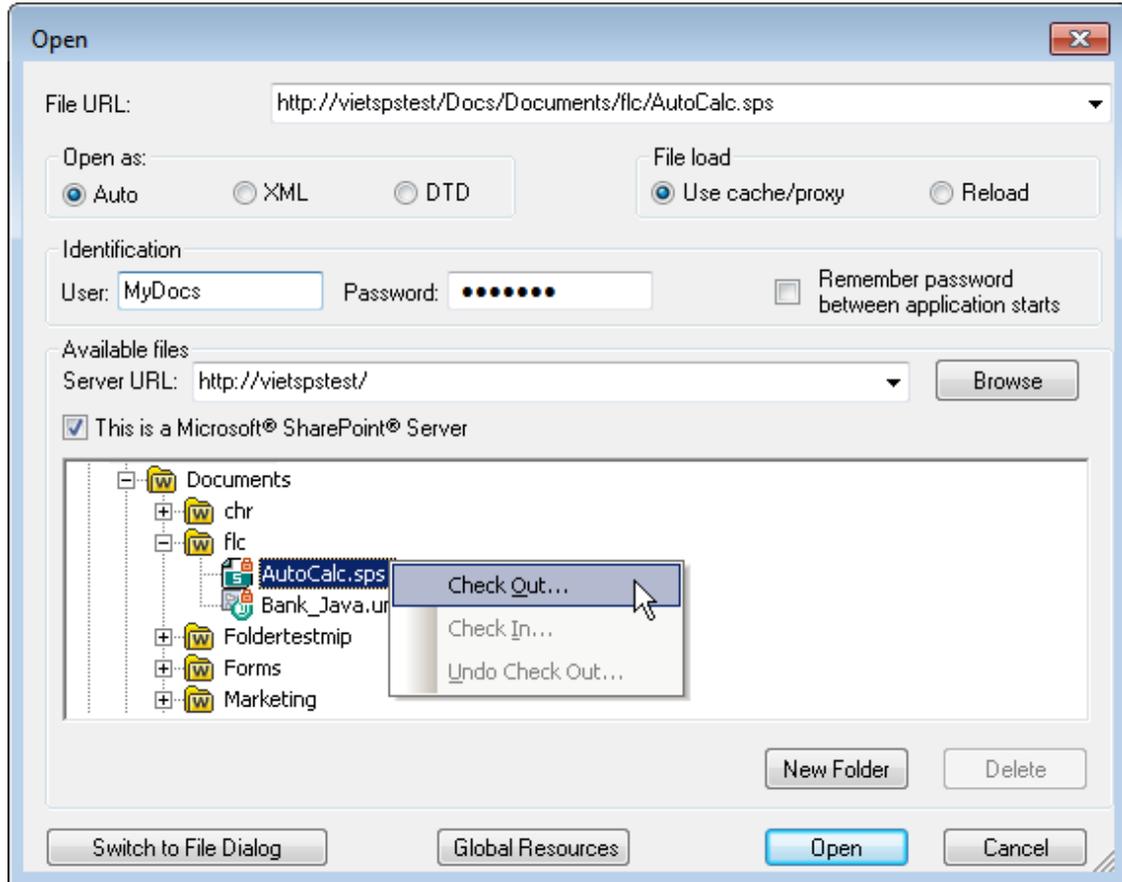
Note the following:

- 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,. For a general description of Global Resources, see the section in this documentation.

The anyType datatype of XML Schema

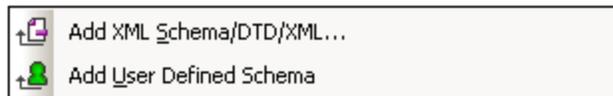
If an element in the XML Schema has been assigned the `anyType` datatype of XML Schema or if it has not been assigned any datatype, then the schema tree in the Schema Tree will show this element as having all the global elements of that schema as possible children. For example, if an element called `email` 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 User-Defined Schemas

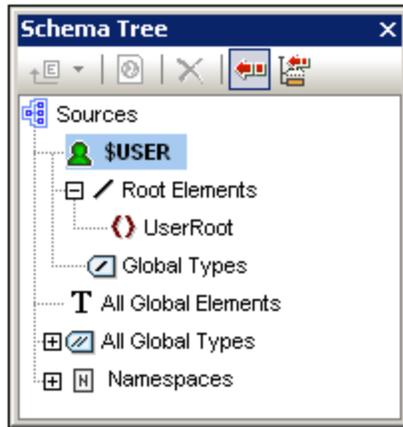
You can quickly create a user-defined schema in the [Schema Tree sidebar](#)³⁵. This is useful if you have an XML document that is not based on any schema and you wish to create an SPS for this XML document.

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

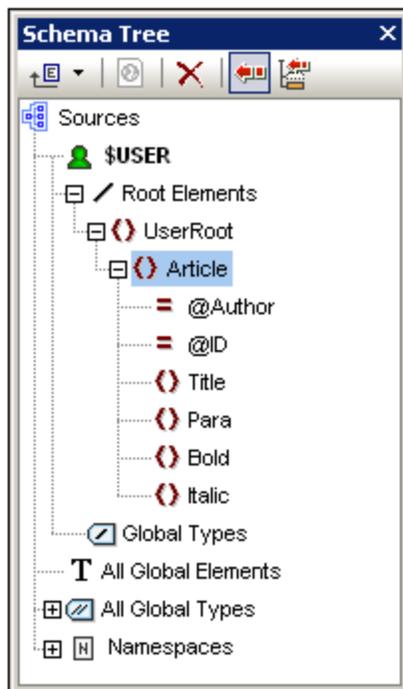
1. Click **File | New | New (empty)**. In the [Design Overview sidebar](#)³², click the **Add New Source** command (under the Sources heading), and select **Add User-Defined Schema** (*screenshot below*).



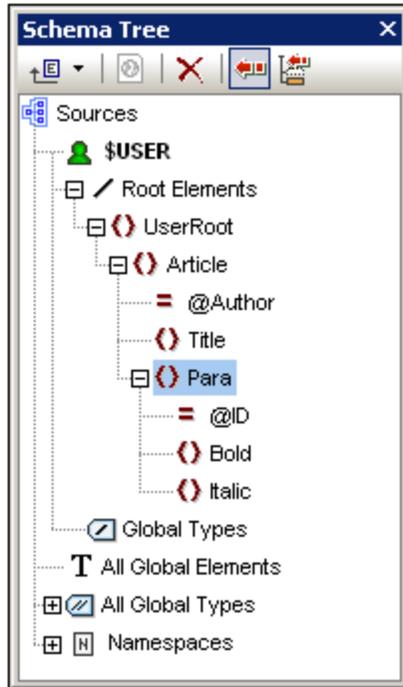
The new schema is created and is indicated with the parameter `$USER` (*screenshot below*).



2. In the Root Elements tree, there is a single [root element \(document element\)](#)²¹ called `UserRoot`.
3. Double-click `UserRoot` and rename it to match the [document element](#)²¹ 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](#)³⁵; 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`.



- 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.

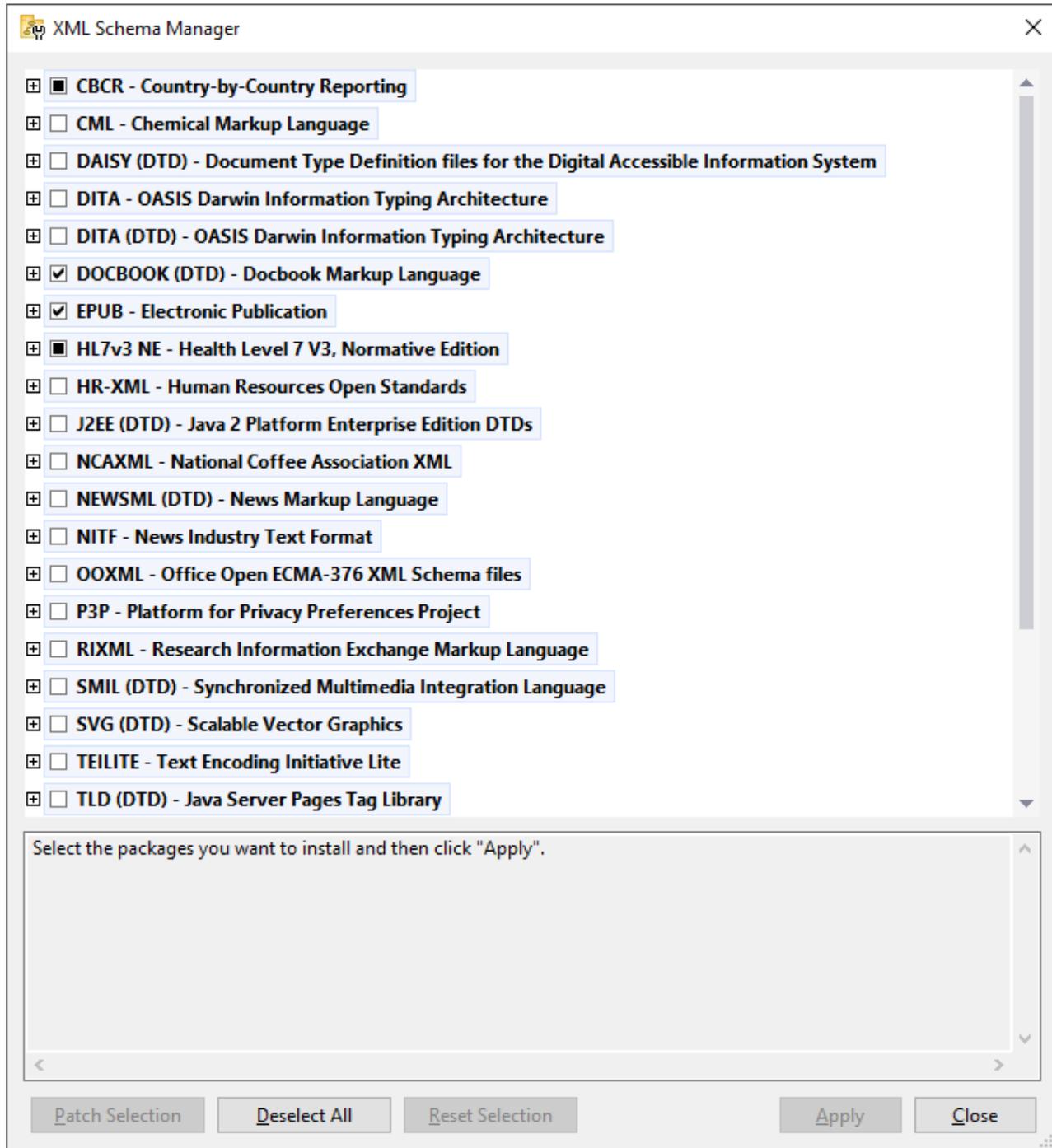
Editing node names and deleting nodes

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.

6.1.3 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 XBRL-enabled 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 Enterprise Edition 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.

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` ¹⁹³ 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:

Windows	C:\ProgramData\Altova\pkgs\.cache
---------	-----------------------------------

<i>Linux</i>	<code>/var/opt/Altova/pkgsg\cache</code>
<i>macOS</i>	<code>/var/Altova/pkgsg</code>

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](#)¹⁹⁶ 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](#)¹⁹⁴ command, and (ii) run the [initialize](#)¹⁹² command. (Alternatively, run the `reset` command with the `--i` option.)

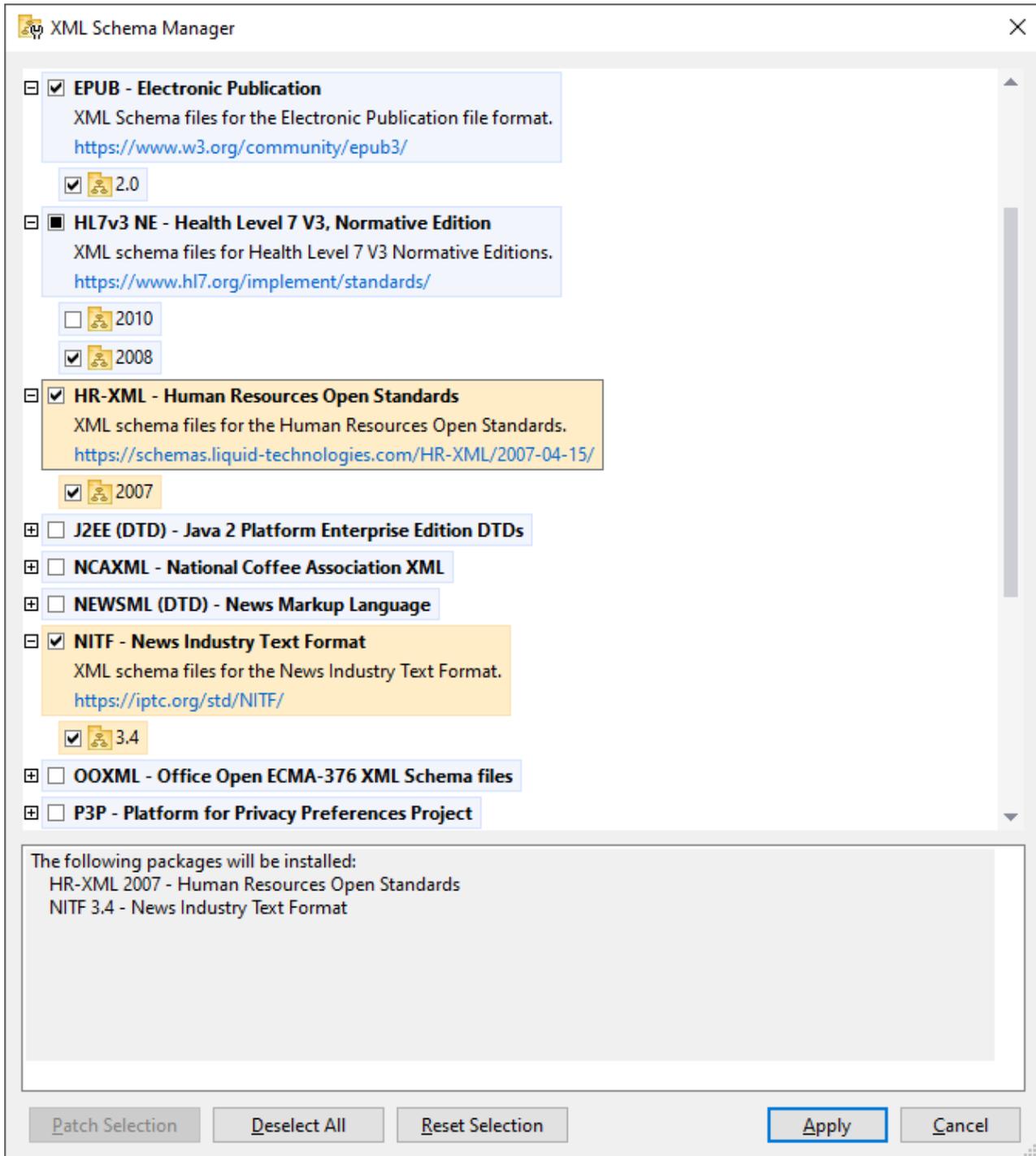
6.1.3.1 Run Schema Manager

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](#)¹⁹¹.

To display help for the commands, run the following:

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

6.1.3.2 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` ¹⁹³ 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`

6.1.3.3 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](#)¹⁸⁷.) 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`¹⁹³ command. This lists any schemas for which upgrades are available.
2. Run the `upgrade`¹⁹⁶ 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`¹⁹³ 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`¹⁹³ command.) You can enter as many schemas as you like. For details, see the description of the `install`¹⁹³ 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**.

6.1.3.4 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`¹⁹⁵ 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`¹⁹³ command.) You can enter as many schemas as you like. For details, see the description of the `uninstall`¹⁹⁵ 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`¹⁹⁴ command.

After running this command, make sure to run the `initialize`¹⁹² command in order to recreate the cache directory. Alternatively, run the `reset`¹⁹⁴ command with the `-i` option.

Note that `reset -i`¹⁹⁴ restores the original installation of the product, so it is recommended to run the `update`¹⁹⁶ command after performing a reset. Alternatively, run the `reset`¹⁹⁴ command with the `-i` and `-u` options.

6.1.3.5 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.

6.1.3.5.1 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
```

6.1.3.5.2 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](#)¹⁹³ 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
```

6.1.3.5.3 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
```

6.1.3.5.4 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](#)¹⁹³ 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](#)¹⁸².

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
```

6.1.3.5.5 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`

6.1.3.5.6 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](#)¹⁹² 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](#)¹⁹⁶ 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`

6.1.3.5.7 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`¹⁹³ 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](#)¹⁸².

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
```

6.1.3.5.8 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](#)¹⁹⁴ and [initialize](#)¹⁹².

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
```

6.1.3.5.9 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](#)¹⁹³ 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](#)²⁴⁰ that uses the `doc()` function can, therefore, also be used to merge XML data (see *example below*).

Example

The [\(My\) Documents folder](#)²³, `C:\Documents and Settings\\My Documents\Altova\StyleVision2023\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*).

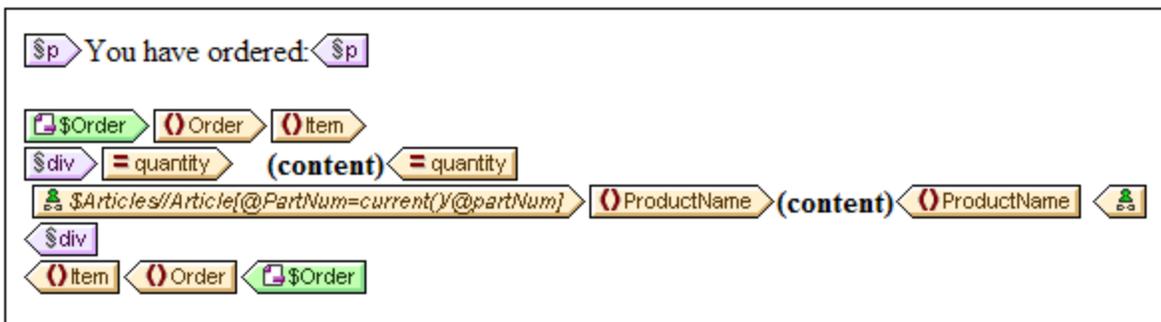
```
<?xml version="1.0" encoding="UTF-8"?>
<Articles xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="MergeArticles.xsd">
  <Article PartNum="833-AA">
    <ProductName>Lapis necklace</ProductName>
    <Price>99.95</Price>
  </Article>
  <Article PartNum="748-OT">
    <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](#)²¹⁹ 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](#)²⁴⁰ (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.

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.

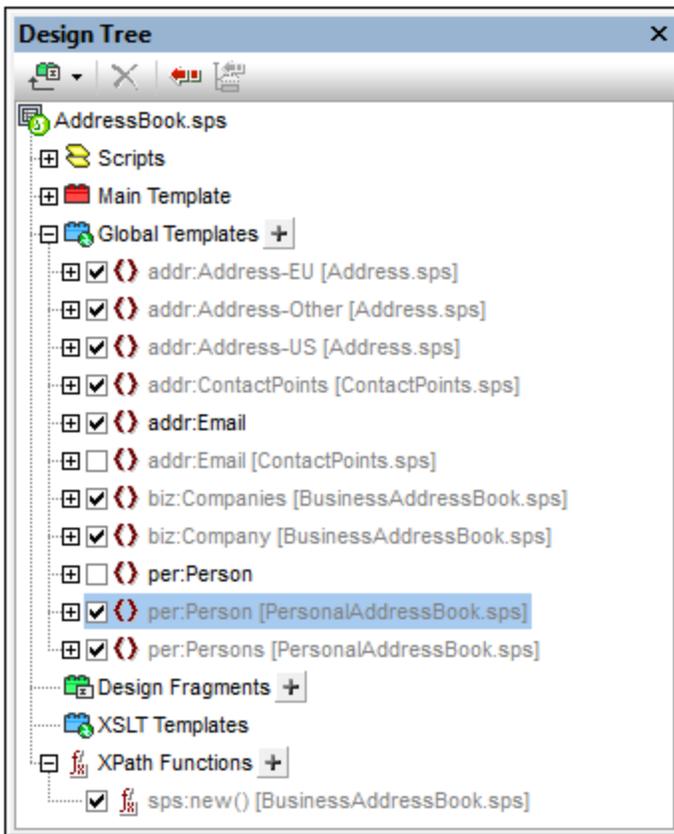
In any given SPS, one or more SPSs can be added as modules. Some types of components (or objects) in these modules are then available to the importing (or referring) SPS.

Available module objects

The section, [Available Module Objects](#)²⁰², 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](#)²⁰⁵ 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](#)²⁰⁵.

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.

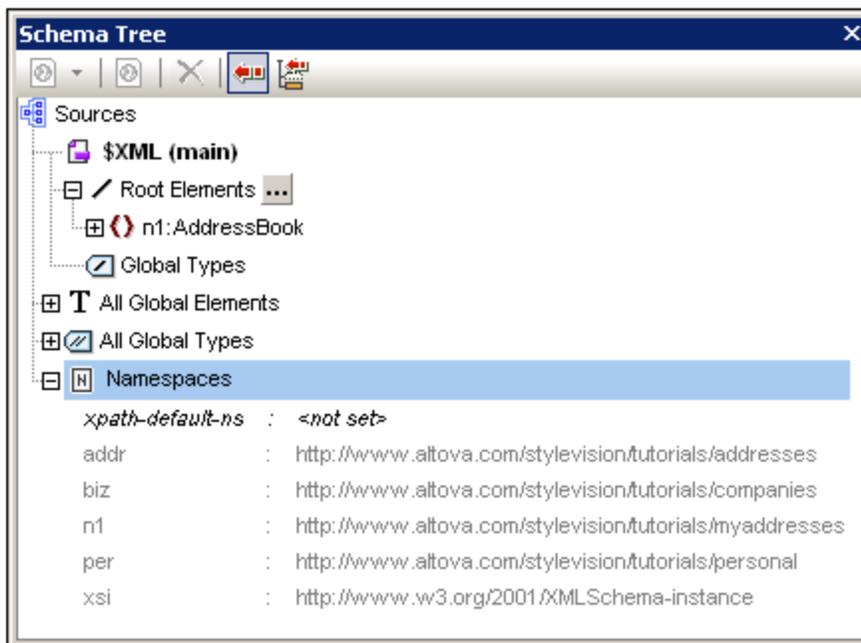
6.3.1 Available Module Objects

This section lists the objects in [added SPS modules](#)²⁰² that are available to the [referring SPS module](#)²⁰². 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](#)²⁰⁵. 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](#) ²⁰³
- [Global templates](#) ²⁰³
- [Design fragments](#) ²⁰⁴
- [Added modules](#) ²⁰⁴
- [Scripts](#) ²⁰⁵
- [CSS styles](#) ²⁰⁵
- [Page layouts](#) ²⁰⁵
- [Unavailable module objects](#) ²⁰⁵

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.

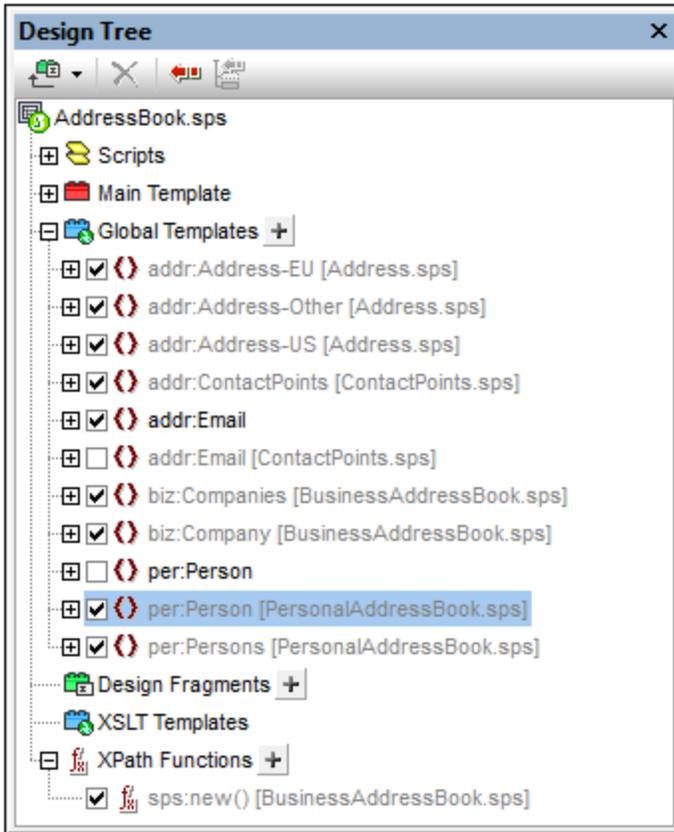


The screenshot above shows the various namespaces in an SPS, together with their prefixes, in the Schema Tree sidebar. These namespaces come from the source schema/s and cannot be edited.

Global templates

The [global templates](#) ²¹⁵ of the added SPS module are available to the referring SPS module and are displayed in the [Design Tree sidebar](#) ³⁸ (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](#)²²⁵ in the added SPS module are available to the referring SPS and are displayed in the [Design Tree sidebar](#)³⁸ (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](#)³².

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](#)³⁶². 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](#)³²⁰. The CSS files are also listed in the [Design Overview sidebar](#)³². 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](#)³⁸.

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](#)²⁰⁵.
2. [Add the module](#)²⁰⁶ 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](#)²⁰⁸ 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](#)²⁰² can only be [global templates](#)²¹⁵. 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](#)²⁰².

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](#) ²⁰⁵. 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:

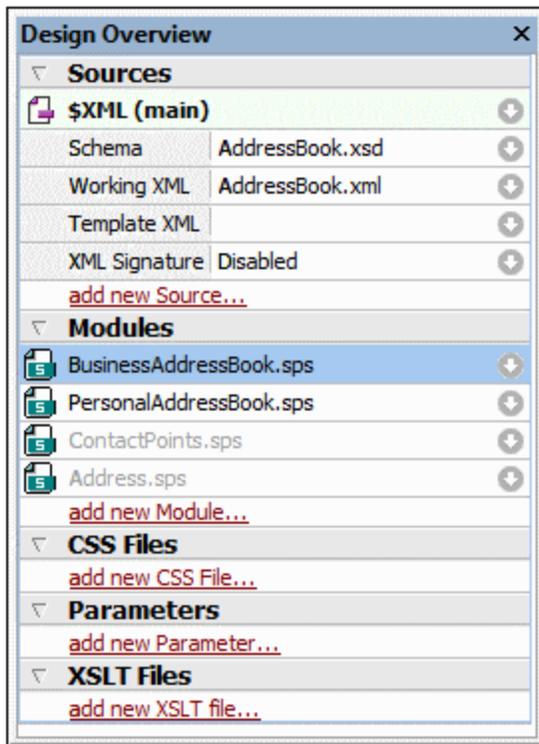
- The main source schema of the referring SPS. In this case, when the SPS is added, the added global templates will be part of the content model of the referring SPS's main schema. The output of these global templates in Authentic View is, therefore, editable.
- A schema which defines a content model 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](#) ²⁰³ used since these determine the expanded names of nodes.

You could use a [Working XML File](#) ²² 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](#) ²⁰⁵.

Adding the SPS module

To add a module to an SPS, in the [Design Overview](#) ³² (*screenshot below*), click the Add New Module command, browse for the required SPS file in the dialog that appears, select it, and click **Open**.



The module is added to the SPS and is listed under the Modules heading in the Design Overview. In the screenshot above, the `BusinessAddressBook.sps` and `PersonalAddressBook.sps` modules have been added to the `AddressBook.sps` module (the active SPS). All the added module objects are listed in the Design Tree sidebar; added CSS files, though, are also also listed in the Design Overview. If the added modules themselves refer to modules, these latter, indirectly imported modules are listed under the Modules heading, but in gray. Information about which modules import an indirectly imported module is available in a pop-up that appears when you mouseover the indirectly imported module.

To open one of the added modules or indirectly imported modules quickly in StyleVision, right-click that module, and select **Open Defining Module** from the context menu that pops up.

Order of added modules

The order in which modules are added and listed is significant for the prioritizing of CSS styles. In keeping with the CSS cascade order, CSS style rules in a relatively later module (lower down the list) have priority over style rules defined in a relatively earlier module (higher up the list). CSS styles in the referring SPS module have priority over those in any added module. To change the relative position of an added module, right-click it in the Design Overview and click, as required, the **Move Up** or **Move Down** command in the context menu.

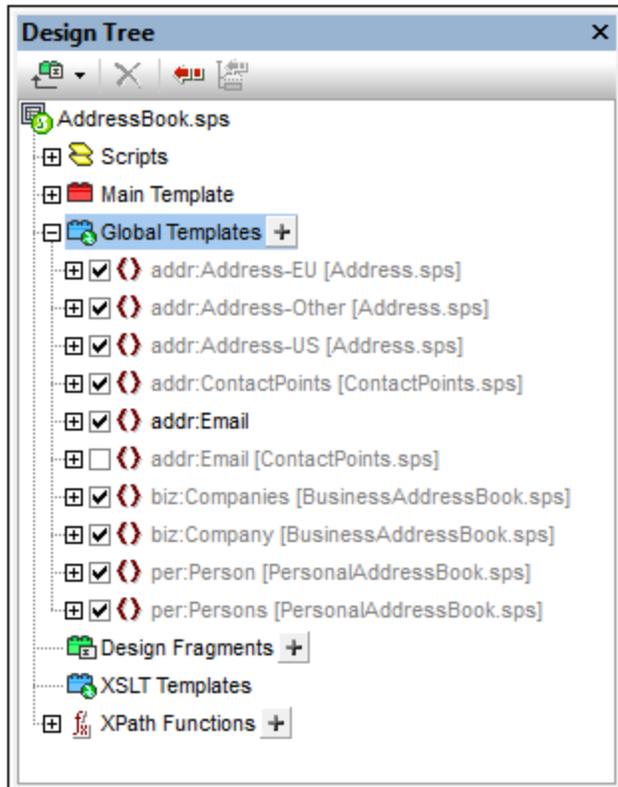
The module order is not significant for resolving conflicts among scripts, global templates, design fragments, and page layout items.

File modification alerts

If any added file (whether an SPS module, schema, or Working XML File) is modified after the referring SPS module has been opened, then a file modification pop-up will alert you to the change and ask whether the referring SPS module should be refreshed with the changes.

Activating/deactivating the added object

All module objects in all added modules (whether added directly or indirectly) are added to the referring SPS and are listed under the corresponding headings in the Design Tree: Scripts; Global Templates; Design Fragments; XSLT Templates; and XPath Functions. Next to each of these objects is a check box (see *screenshot below*), which you can check or uncheck to, respectively, activate or deactivate that object. When an object is deactivated, it is effectively removed from the SPS.



In the screenshot above, all the global templates used in the `AddressBook.sps` module are listed under the Global Templates heading. Those that have been added via other modules (whether directly or indirectly) are displayed in gray. Those that have been created directly in `AddressBook.sps` are displayed in black. The screenshot shows that only one global template, `addr:Email`, has been created in `AddressBook.sps` itself. All the other global templates have been added via other modules, and the file in which each of these is defined is listed next to its name.

Notice that there are two global templates for `addr:Email`, one created in the referring SPS (`AddressBook.sps`) itself, and the other created in the added module `ContactPoints.sps`. If more than one global template has the same (namespace-) expanded name, then only one of these will be active at a time. You can select which one by checking its check box. (Alternatively, you activate the global template from its context menu in Design View.) This mechanism is useful if you: (i) wish to override an added global template with one that you create in the referring SPS module, or (ii) wish to resolve a situation where a global template for one element is defined in more than one added module.

A global template that has been defined in the current SPS can be deleted by selecting it and clicking the **Remove** button. However, global templates that have been defined in an added module cannot be removed

from the referring SPS. They must be removed by opening the added SPS and removing the global template there.

Individual scripts, Design Fragments, and page layout items can be activated and deactivated in the same way.

Applying or using modular objects

In the [referring SPS module](#)²⁰², 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](#)²⁰⁸.

Global templates

When you wish to use a [global template](#)²¹⁵ 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](#)²²⁵.

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](#)³⁶⁴.

6.3.3 Example: An Address Book

The [\(My\) Documents folder](#)²³, C:\Documents and Settings\\My Documents\Altova\StyleVision2023\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

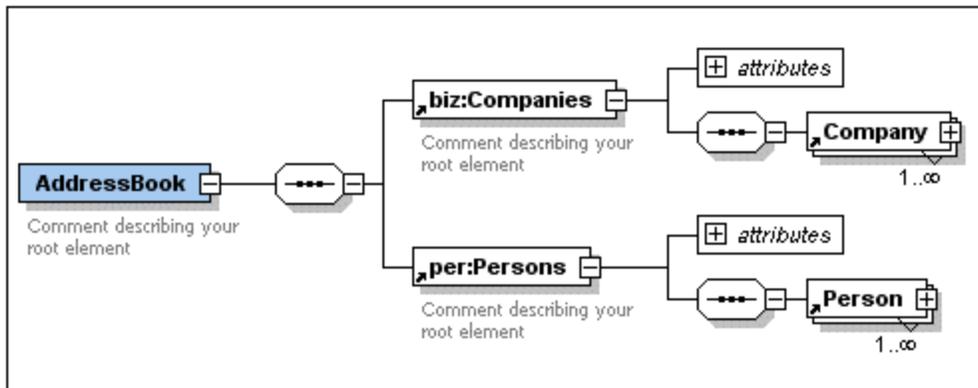
one of three content models, depending on whether the address is in the EU, US, or elsewhere. The output for all three content models is defined in a single SPS. However, they could have been defined in separate SPSs, which would have provided finer granularity. In this case, modularization would provide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

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

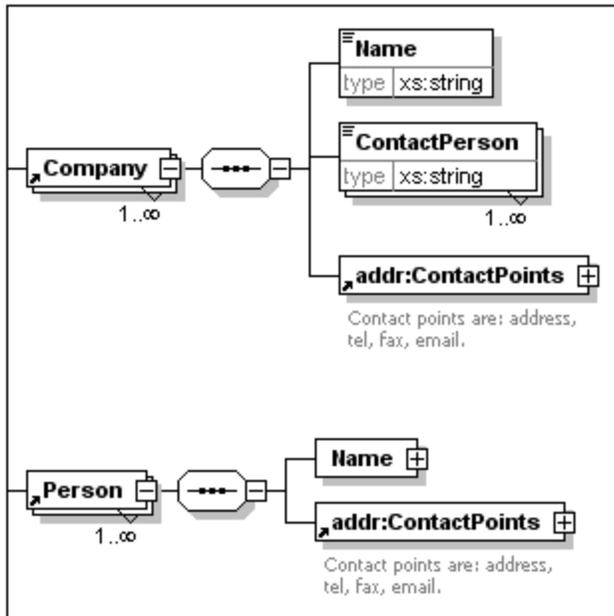
- [The schema files](#) ²¹⁰
- [The XML data sources](#) ²¹¹
- [The SPS files](#) ²¹²

The schema files

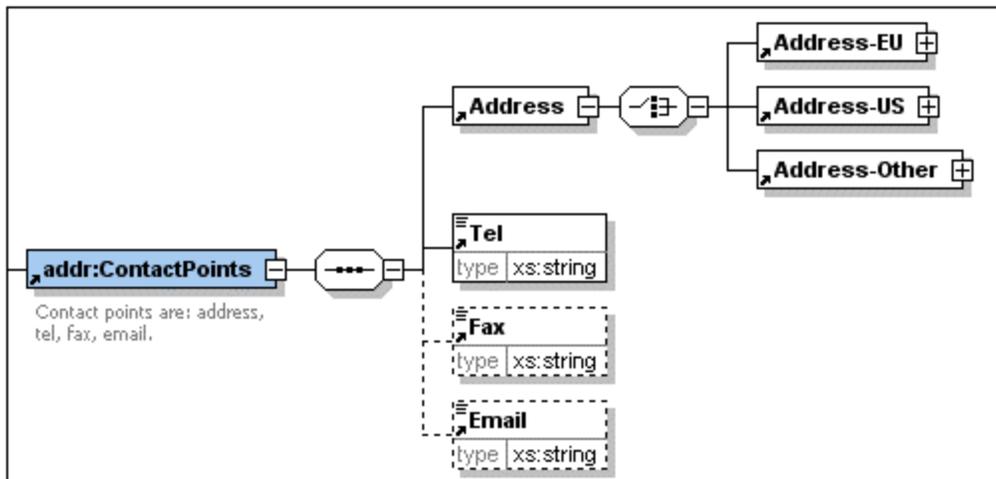
When creating schemas for modular SPSs, the most important thing to bear in mind is to create the elements that you wish to re-use as global elements. The schema for the address book is `AddressBook.xsd`. This schema has been constructed by importing the schemas for the business address book (`BusinessAddressBook.xsd`) 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](#) ²² of the following SPS modules:

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

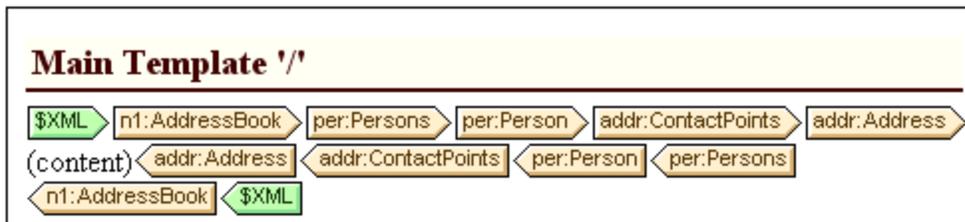
The SPS modules

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.

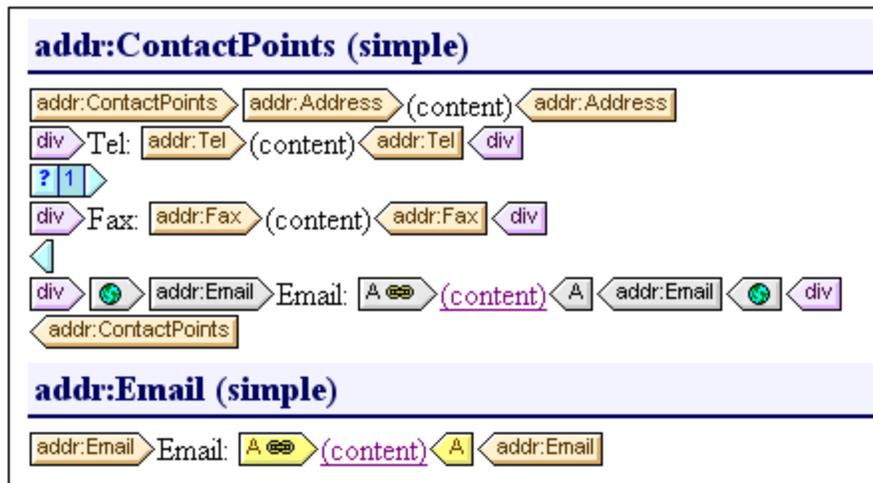


Because only the `Address` element is processed, the output previews show only the output of `Address`. When `Address.sps` is used as a module, the global templates are added and the main template is ignored.

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

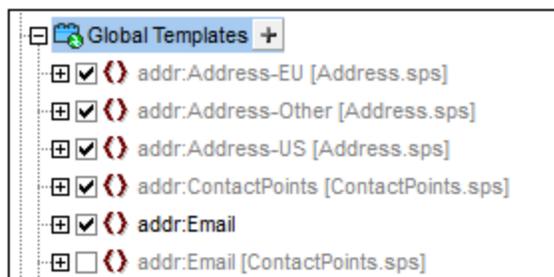
These SPSs each import one module, which in turn imports another. Note that the main template simply applies global templates.

- 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](#)²¹⁵ and, optionally, one or more [global templates](#)²¹⁵. 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](#)²²³ 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](#)³⁰⁵ in the design, only the structure.

Note: In Design View, the SPS can have several templates: the main template, global templates, and Design Fragments. You can control which of these template types is displayed in Design View by using [Template Display Filters](#)⁴²⁰, which are available as [toolbar icons](#)⁴²⁰. These display filters will help you optimize and switch between different displays of your SPS.

6.4.1 Main Template

The [main template](#)²¹ 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 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](#)²¹. 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](#)²¹ 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](#)²¹ of the schema. StyleVision offers the option of selecting multiple root elements ([document elements](#)²¹). This means that within the main template, there can be [local templates](#)²¹ 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](#)²¹ 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](#)²¹, [design fragments](#)²²⁵, or other global templates. The processing model is similar to that of

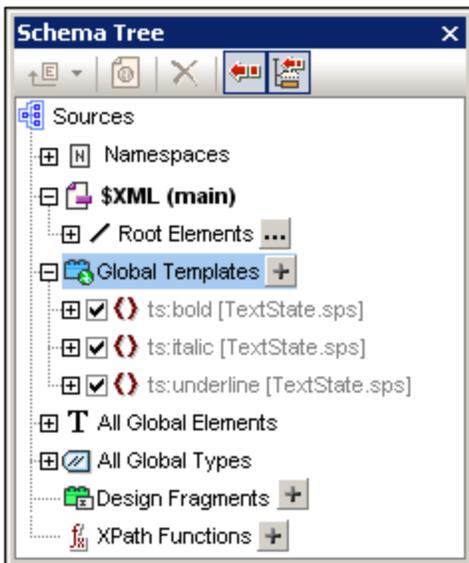
declarative programming languages, in that a single template is defined and invoked multiple times. In this way a single definition can be re-used multiple times. Global templates are invoked in two situations:

- When a node or type in the [main template](#)²¹ has been set to reference its global template (done by right-clicking the component in the design and selecting Make Global Template).
- When a [\(contents\)](#)¹⁰³ or [\(rest-of-contents\)](#)¹⁰⁶ is inserted within an element or type in a [local template](#)²¹, and the rest of the content of that element or type includes a node or type for which a [global template](#)²¹ 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](#)²¹ 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 [\(contents\)](#)¹⁰³ or [\(rest-of-contents\)](#)¹⁰⁶ 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

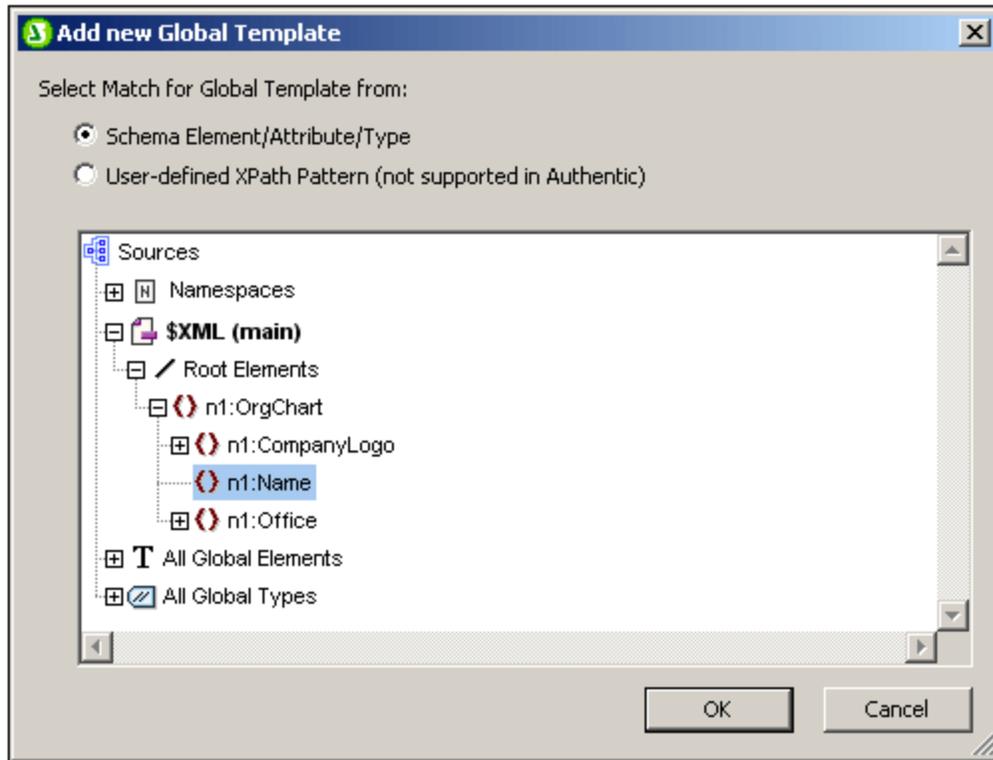
Global templates can be created for any node or type in the schema, or for a node specified in an XPath pattern., and are created from the Schema Tree sidebar (*screenshot below*).



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. 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](#)²²⁴ in the context menu that appears. This pops up the [Edit XPath Expression dialog](#)³⁹⁷, 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](#)⁴⁴³). 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.

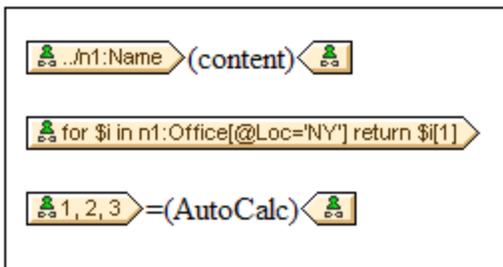
It is important to note that a simple global template will be automatically converted to a complex global template if a [predefined format](#)³⁰⁶ or newline is created **around** the element node for which the global template was created. This will result in the processing behaviour for complex global templates (described in the previous list item). To revert to the simple global template, the [predefined format](#)³⁰⁶ 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](#)³⁸), as the case may be. To avoid the automatic conversion from simple global template to complex global template, make sure that the [predefined format](#)³⁰⁶ 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](#)²⁰¹, 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](#)²⁰¹.

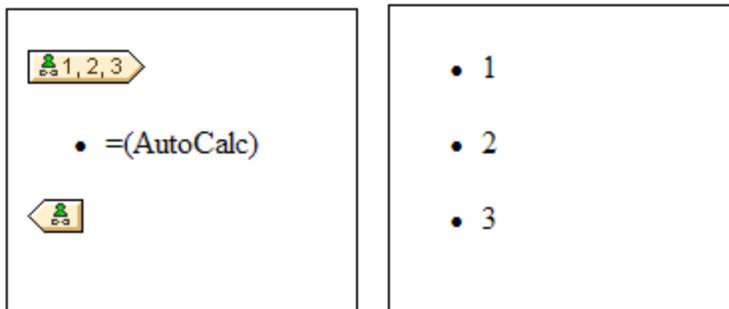
6.4.3 User-Defined Templates

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.



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. 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	<code><xsl:for-each select="Org"></code>	Each <code>Office</code> element has its own <code>Dept</code>

	<pre><xsl:for-each select="Office"> <xsl:for-each select="Dept"> ... </xsl:for-each> </xsl:for-each></pre>	population. So grouping and sorting can be done within each Office.
Yes	<pre><xsl:for-each select="/Org/Office/Dept"> ... </xsl:for-each></pre>	The Dept population extends over all Office elements and across Org.

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

Inserting a User-Defined Template

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

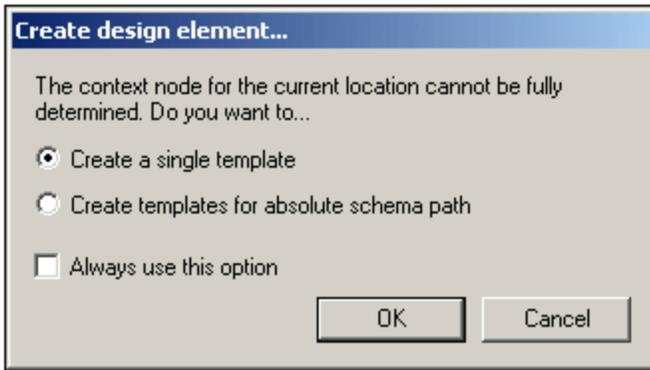
1. Click the **Insert User-Defined Template** icon in the Insert Design Elements toolbar and then click the design location where you wish to insert the template. Alternatively, right-click the design location where you wish to insert the template and, from the context menu that appears, select the **Insert User-Defined Template** command.
2. In the [Edit XPath Expression](#)³⁹⁷ 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](#)²²³.

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

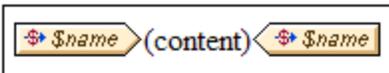
If a node from the schema tree is added to a User-Defined Template, the context for the new node will not be known if the User-Defined Template has been created for a node or sequence that cannot be placed in the context of the schema source of the SPS. You will therefore be prompted (*screenshot below*) 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](#)⁵¹³.

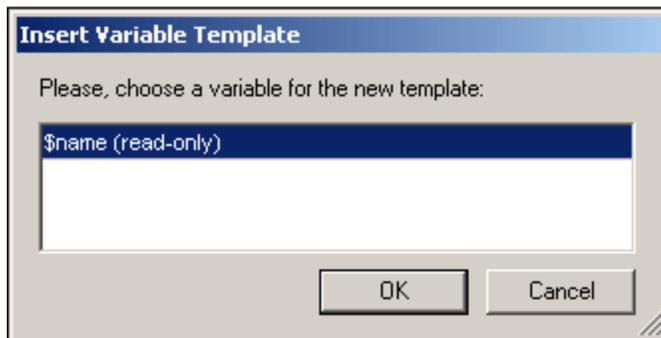
6.4.4 Variable Templates

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:

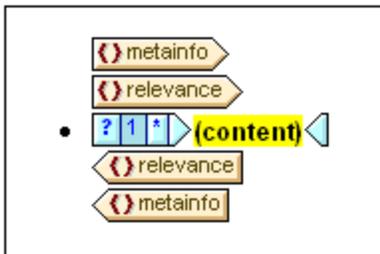
1. Place the cursor in the design at the location where the template is to be inserted.
2. Click the **Insert | Variable Template** command. This pops up the Insert Variable Template dialog (*screenshot below*).



3. The dialog contains a list of all the [user-declared parameters and variables](#)²⁶³ 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](#)³⁹³.



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).

The commands in this context menu are described below:

- [Global templates](#)²²³
- [Template match](#)²²⁴
- [XPath filters](#)²²⁴
- [Group by, Sort by, Define variables, Template serves as level](#)²²⁵
- [Create Design Fragment](#)²²⁵
- [Remove Tag Only](#)²²⁵
- [Edit, Enclose with, Change to](#)²²⁵

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](#) ²¹⁵.

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](#) ³⁹⁷, 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](#) ¹¹².

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](#) ²¹⁵.

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](#)²⁵⁰.
- 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](#)²⁵⁸.
- 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](#)²⁶⁸.
- 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](#)²⁷¹.

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](#)¹⁰³, [special paragraph](#)¹⁰⁵, [Bullets and Numbering](#)¹³⁸, [Hyperlink](#)²⁹⁸, [Condition](#)²⁴⁵, [TOC Bookmark and Level](#)²⁷¹.
- *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](#)¹⁶⁹.

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:

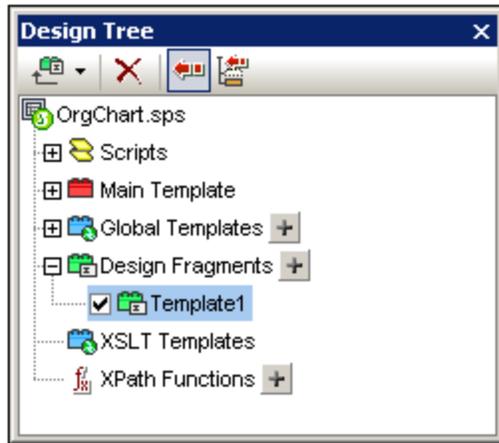
1. [Create the Design Fragment in the design](#)²²⁶
2. [Fill out the contents of the Design Fragment](#)²²⁷

3. [Insert the Design Fragment at a location in a template](#) ²²⁸.

Creating a Design Fragment

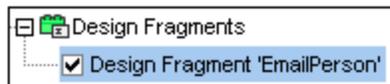
To create a Design Fragment do the following:

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](#) ⁴²⁰ icons in StyleVision's [Design Filter](#) ⁴²⁰ 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.



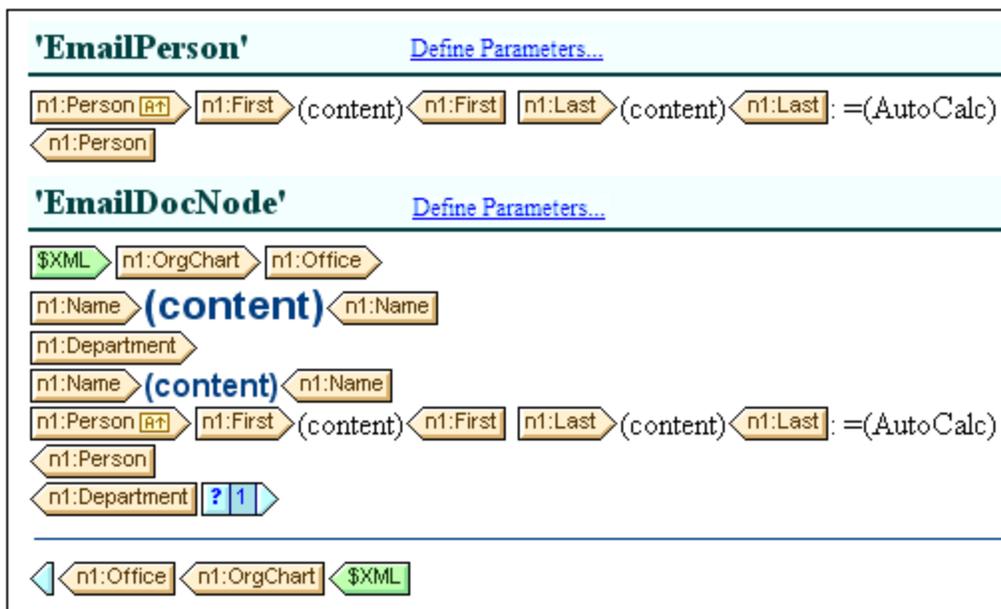
3. In the design, create the contents of the Design Fragment template. How to do this is described in the next section.

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](#) ²²⁸.

Creating the contents of a Design Fragment

The contents of the Design Fragment template are created [as for any other template](#)¹⁰². 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](#)²³, `C:\Documents and Settings\\My Documents\Altova\StyleVision2023\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](#)²⁶⁴ 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](#)²²⁷ 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](#)⁴⁴³). 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](#)²⁰¹, 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](#)²⁰¹.

Example file

For an example SPS, go to the [\(My\) Documents folder](#)²³, C:\Documents and Settings\\My Documents\Altova\StyleVision2023\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.

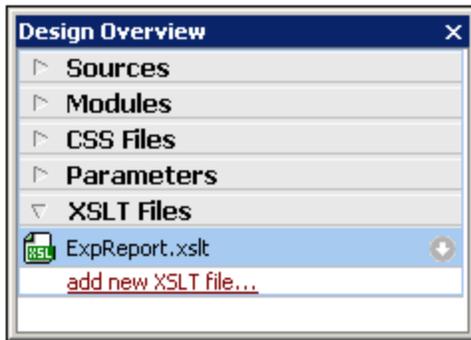
Note the following points:

- Imported XSLT templates cannot be modified in StyleVision.

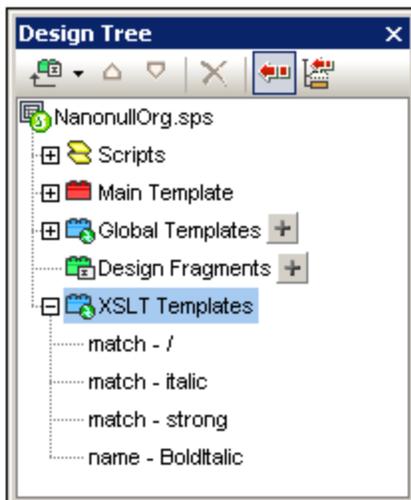
Importing the XSLT file

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.



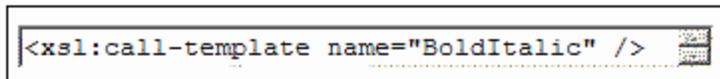
There are two types of imported XSLT templates: (i) match templates (indicated by *Match*), and (ii) named templates (indicated by *Name*). In the Design Tree, these two types are listed with (i) the value of the `select` attribute of match templates, and (ii) by the value of the `name` attribute of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

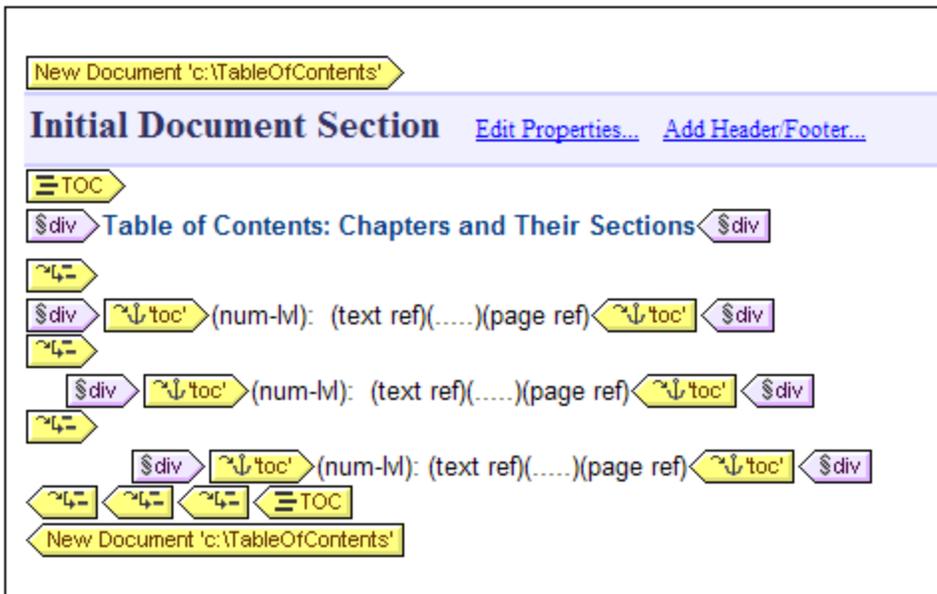
A named template can be dragged from the Design Tree to any location in the design. At this location, it will be created as an `xsl:call-template` element (*screenshot below*) that calls the named template.

A screenshot of a code editor showing an XSLT call-template element. The code is: `<xsl:call-template name="BoldItalic" />`. The text is in a monospaced font, and the element is enclosed in a rectangular box. There is a small icon on the right side of the box, possibly representing a code editor or a help icon.

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](#)²³³.

This description of multiple output-documents is organized into the following sub-sections:

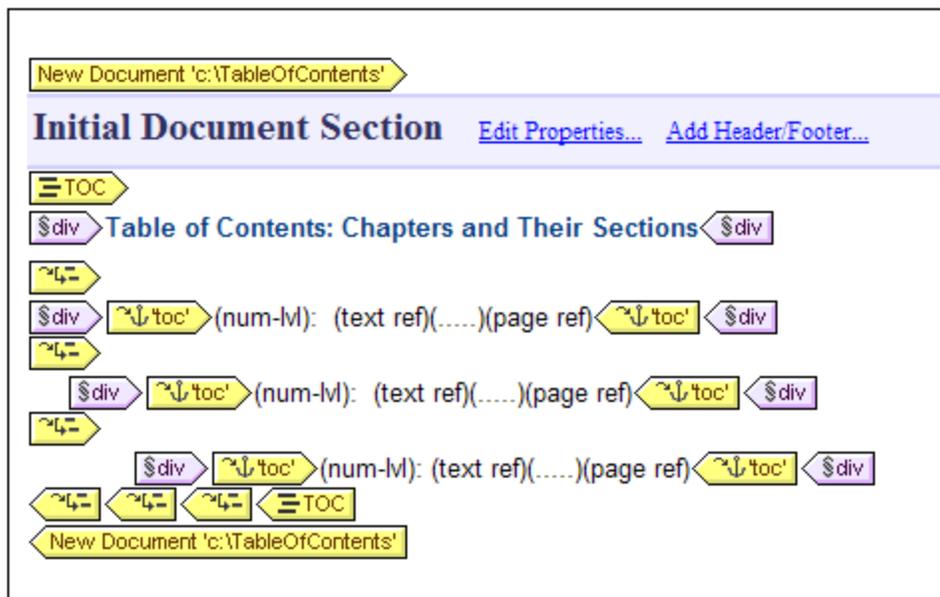
- [Inserting a New Document Template](#)²³²
- [New Document Templates and Design Structure](#)²³³
- [URLs of New Document Templates](#)²³³
- [Preview and Output Document Files](#)²³⁵
- [Document Properties and Styles](#)²³⁸

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](#)²³³.
2. The New Document Template contains one Initial Document Section.

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](#)²³³ as well as the [input \(source XML document\) structure](#)²³³ 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.

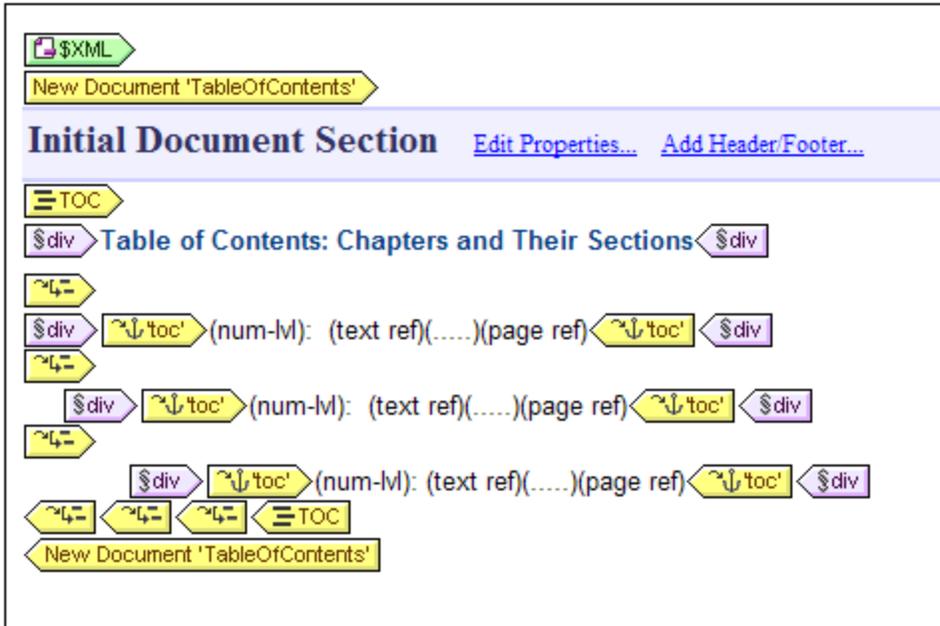
6.6.3 URLs of New Document Templates

In this section we describe how the [URLs of New Document templates relate to design structure](#)²³³, how [URLs are edited](#)²³⁵, and [how multiple output-documents can be linked](#)²³⁵ among each other.

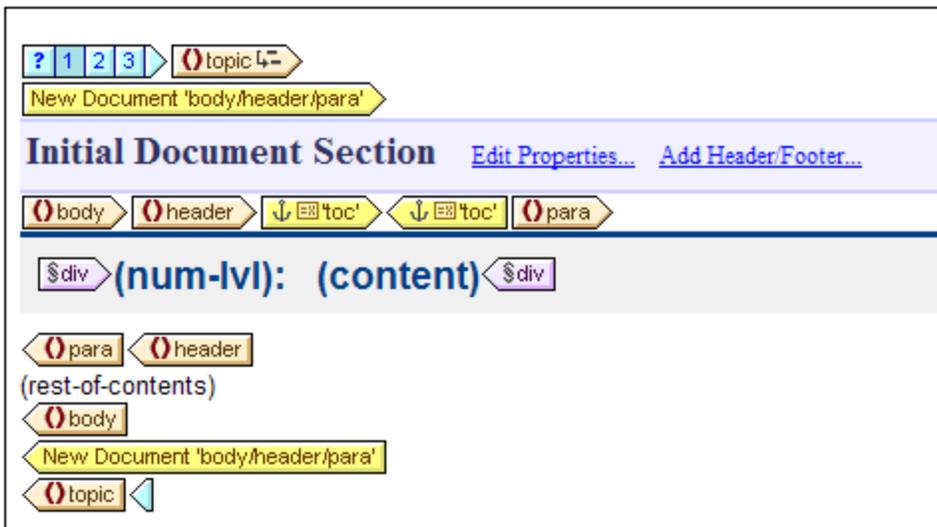
URLs of New Document templates

If the New Document template is processed only once (see [preceding section](#)²³³), 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](#) ²³⁵ for details). Alternatively, if the URL contained a path, the output-document will be saved to the location specified in the path.

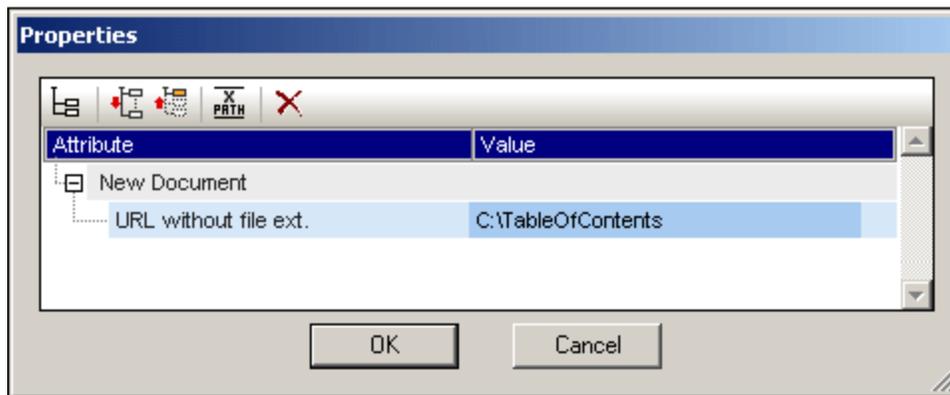


If, on the other hand, a New Document template will be processed multiple times to generate multiple output-documents (see [preceding section](#) ²³³), then the template's *URL* property must be a dynamic URL that is selected with an XPath expression. In the screenshot below, the URL of the New Document 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.

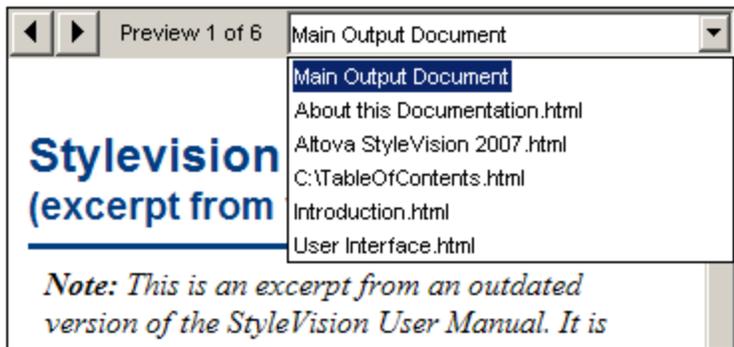
Linking the documents

Multiple output-documents can be linked to one another using [bookmarks and hyperlinks](#)²⁹⁸. A [bookmark](#)²⁹⁸ can be placed at the head of a New Document Template or anywhere within the New Document Template. [Hyperlinks](#)³⁰⁰ 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 \(TOC\)](#)²⁷¹ 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

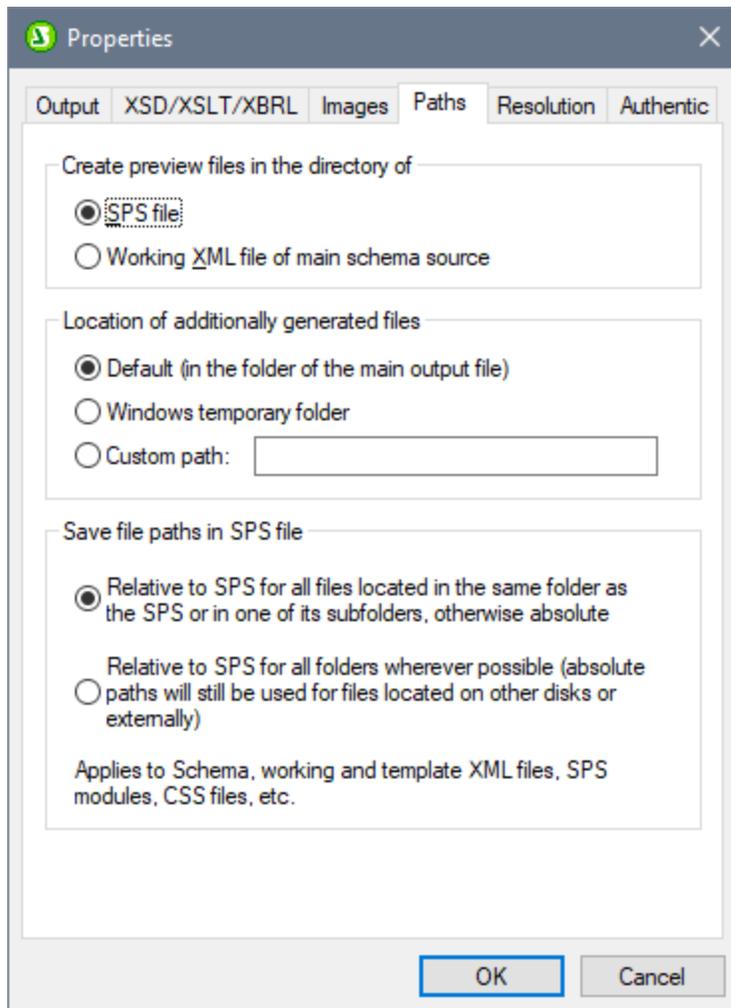
The output previews of a design document show each of the multiple output-documents that have been specified in the design as separate documents (*see screenshot below*).



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**.

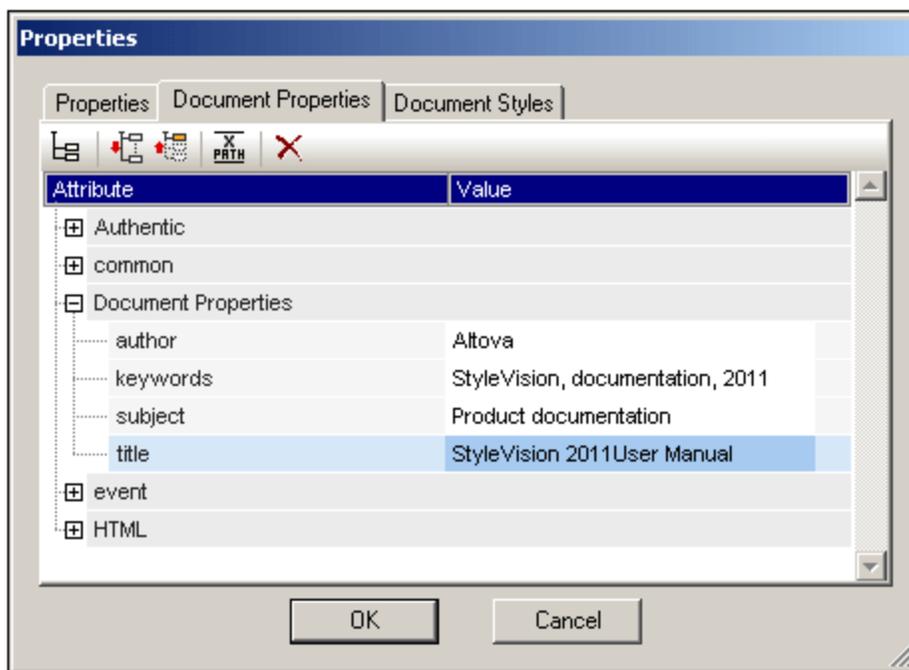
The location of all output-document files as well as other additionally generated files, such as image files and chart-image files, will be displayed in a pop-up window for your information.

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](#)³²⁷.

7 Advanced Features

How to create the basic content and structure of the SPS design is described in the sections, [SPS File Content](#)¹⁰² and [SPS File Structure](#)¹⁷². 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](#)²⁴⁰. 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](#)²⁴⁵. 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](#)²⁵⁰. Processing can be defined for a group of elements that are selected with an XPath expression.
- [Sorting](#)²⁵⁸. A set of XML elements can be sorted on multiple sort-keys.
- [Parameters and Variables](#)²⁶³. 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](#)²⁷¹. 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](#)²⁹³ (repeating nodes in the document can be numbered automatically and the numbers formatted); (ii) [Text References](#)²⁹⁷ (text in the document can be marked for referencing and then referenced from elsewhere in the document); and (iii) [Bookmarks and Hyperlinks](#)²⁹⁸ (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:

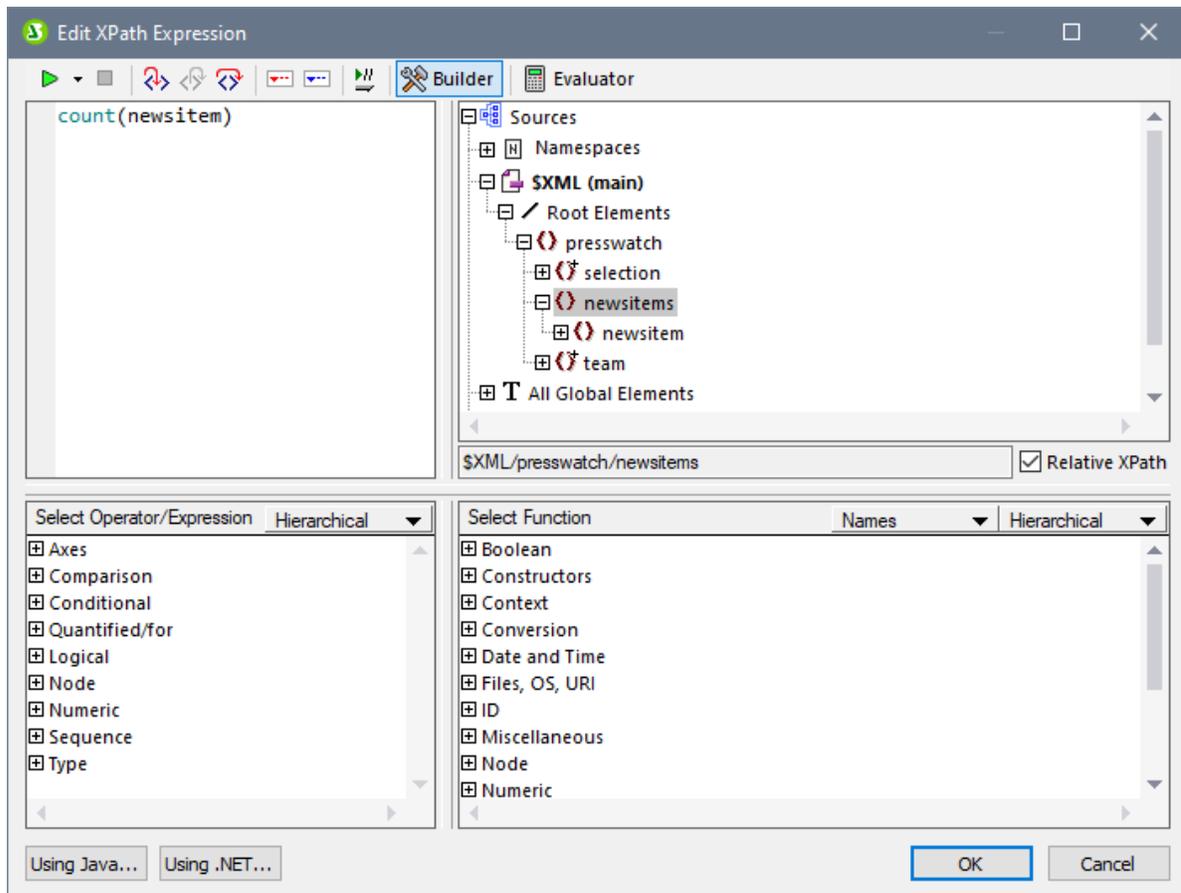
- Inserting calculations involving operations on dynamic data values. For example, you can count the number of `Employee` elements in an `Office` element (with `count(Employee)`), or sum the values of all `Price` elements in each `Invoice` element (with `sum(Price)`), or join the `FirstName` and `LastName`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

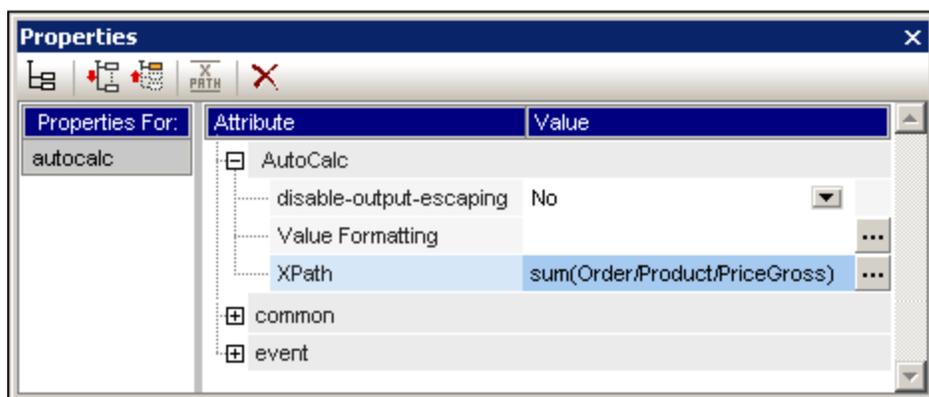


2. In the *Expression* pane, enter the XPath expression for the Auto-Calculation via the keyboard. Alternatively, enter the expression by double-clicking nodes, operators, and/or functions in the respective panes of the dialog. It is important to be aware of the context node at the insertion point; the context node is highlighted in the schema source tree (in the screenshot above the context node, *newsitems*, is highlighted). If you have selected XSLT 1.0 as the version of the XSLT language for your SPS, then you must use XPath 1.0 expressions; if you have selected XSLT 2.0 or XSLT 3.0, then you must use, respectively, XPath 2.0 or XPath 3.0 expressions. For a detailed description of the Edit XPath Expression dialog, see the section [Edit XPath Expression](#)³⁹⁷.

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 HTML 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](#)³⁹⁷ (*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](#)³¹⁰ 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. 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.

7.1.2 Example: An Invoice

The `SimpleInvoice.sps` example in the [\(My\) Documents folder](#)²³, `C:\Documents and Settings\\My Documents\Altova\StyleVision2023\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
- Creating complex calculations

In the example file, the Auto-Calculations have been highlighted with a yellow background color (see *screenshot below*).

Counting nodes

In the SimpleInvoice 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 <input type="button" value="v"/>
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 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 SimpleInvoice example, the product category (Book, CD, DVD, or Electronics) is contained in the `//Product/Category` node and is displayed in 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. Notice that the VAT value of different products are different (Book=10%; CD=15%; DVD=15%; Electronics=20%); they have been calculated by the Auto-Calculation.

Creating a complex Auto-Calculation

The VAT percentage, obtained by the Auto-Calculation described above, is required to calculate the gross price (net price + VAT amount) of each product. The formula to use would be derived as follows:

Gross Price = Net Price + VAT-amount

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

Gross Price = Net Price + (Net Price * VAT-percentage div 100)

The net price of a product is obtained from the `PriceNet` node. The VAT percentage is calculated by an Auto-Calculation as described above; it is not contained in any node. Since this value cannot be obtained directly from a node, it must be re-calculated in the gross price Auto-Calculation. The XPath expression to do this would be:

```
for $i in Category return PriceNet + (PriceNet * (/Invoice/Categories/Category[. =
    $i]/@rate) div 100)
```

The XPath expression can be [viewed and edited in the Properties window](#)²⁴⁰. You can test the Auto-Calculation for the gross price by changing, in the XML file and then re-loading the SPS, 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:	<input type="text" value="DVD"/>
VAT:	15%
Price including VAT:	€ 23

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](#)²⁴⁵, which describes how to create a condition and its branches.
- [Editing Conditions](#)²⁴⁸, about how to edit the XPath expressions of condition branches after they have been created.
- [Conditions and Auto-Calculations](#)²⁴⁹, explains usage issues when conditions and Auto-Calculations are used in combination.

7.2.1 Setting Up the Conditions

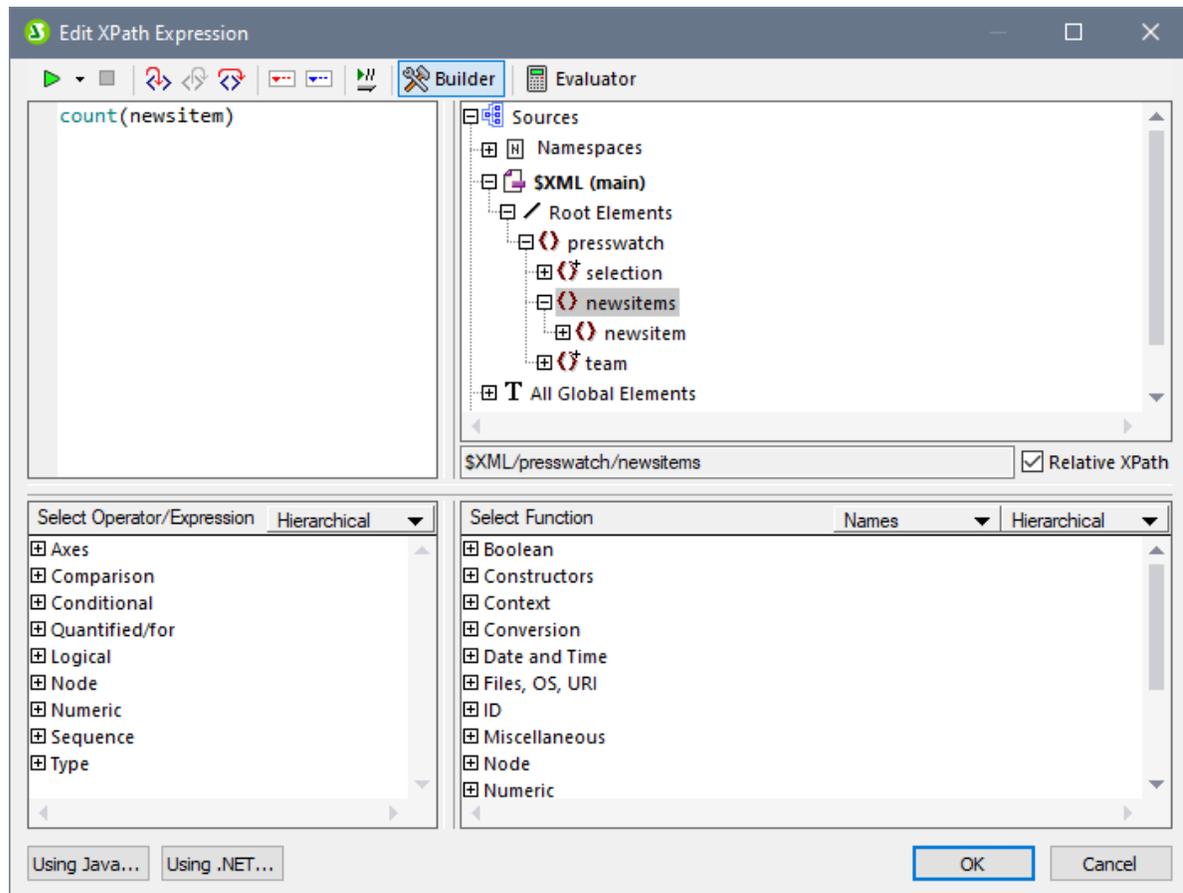
Setting up the condition consists of the following steps:

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.

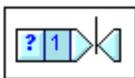
Creating the condition with its first branch

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](#)³⁹⁷ pops up (*screenshot below*).



- 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.
- 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.

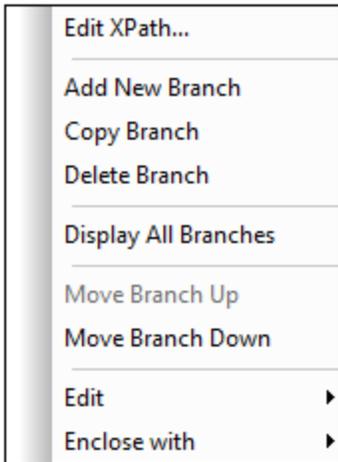


To select the entire condition, click the cell with the question mark. To select the first branch, click the cell with the number one.

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**).



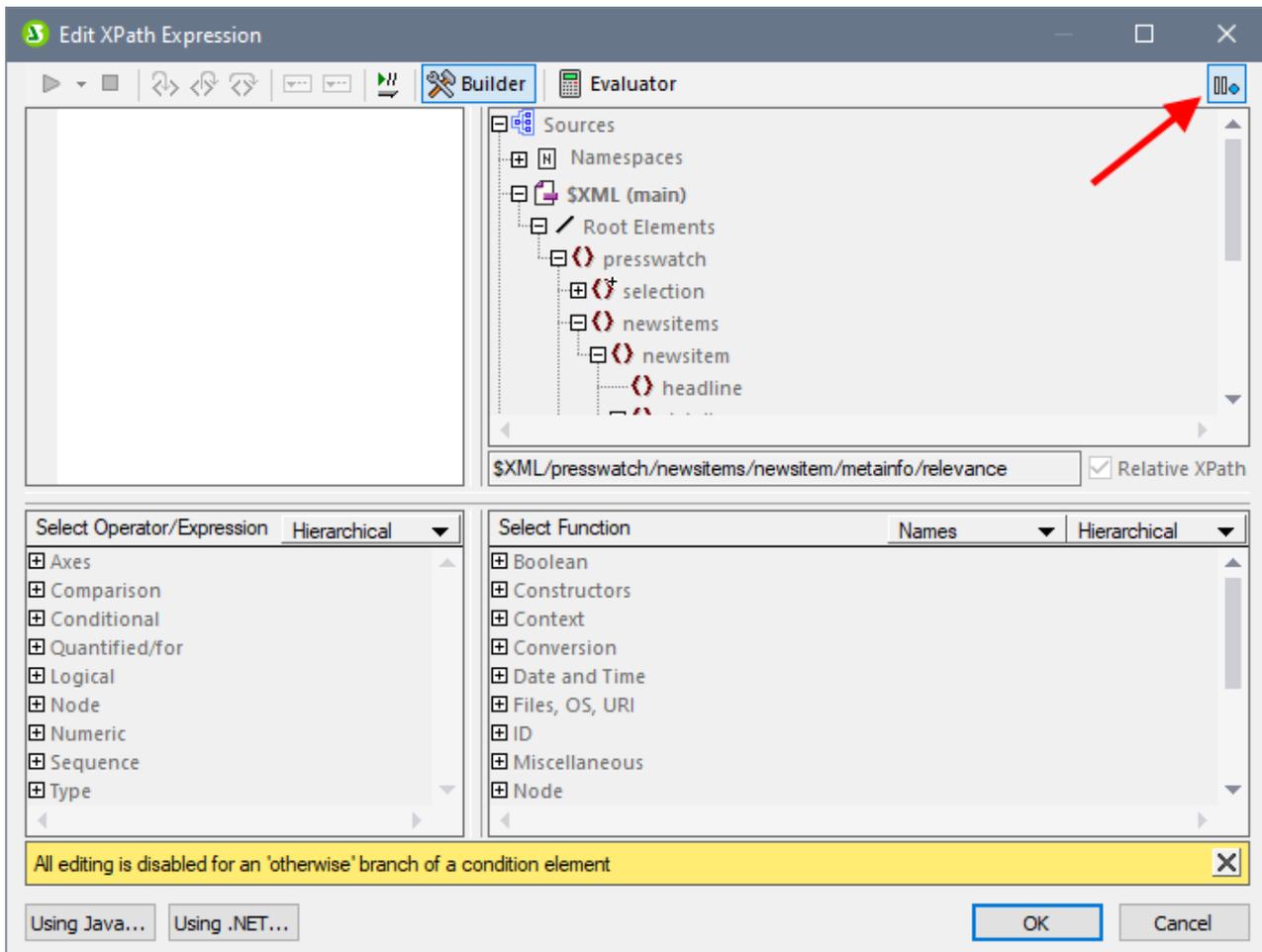
To create a new branch, right-click any branch of the condition and select **Add New Branch** from the context menu. The [Edit XPath Expression dialog](#)³⁹⁷ will pop up. After entering an XPath expression and clicking **OK**, a new empty branch is added to the condition. This is indicated in the design 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](#)³⁹⁷ 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](#)³⁹⁷ 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**.

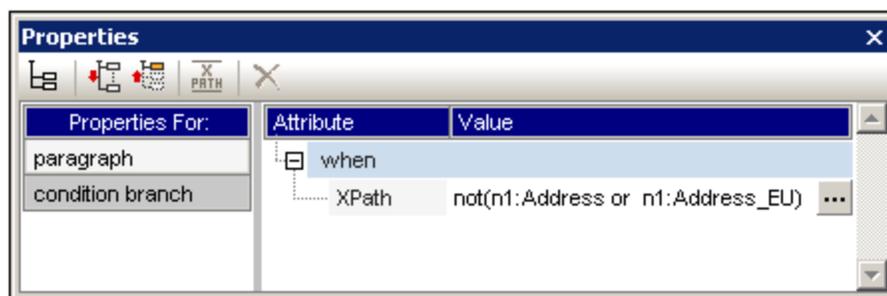
Deleting a branch

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](#)  ³⁹⁷, in which you can edit the XPath expression for that branch of the condition.

7.2.3 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. If the Auto-Calculation were inside a branch that is not visible, then it would not be triggered.
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 Grouping

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

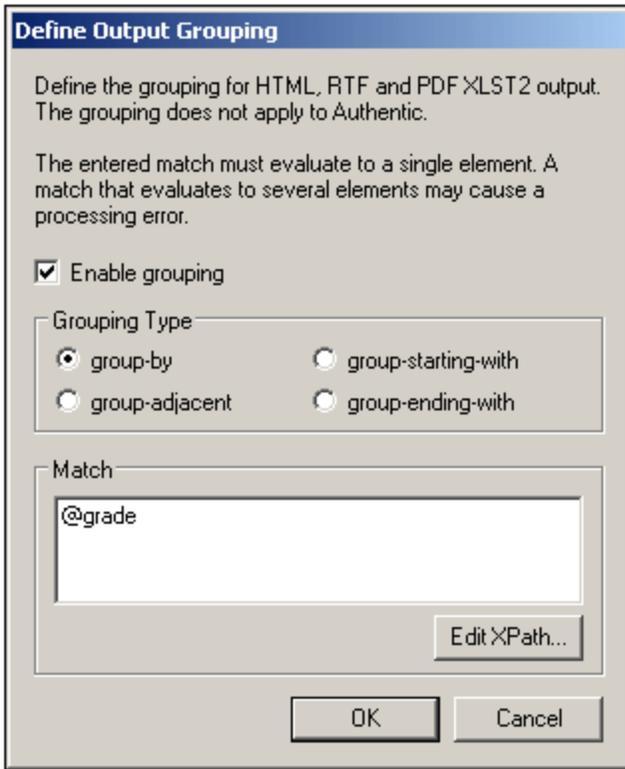
Grouping criteria

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:

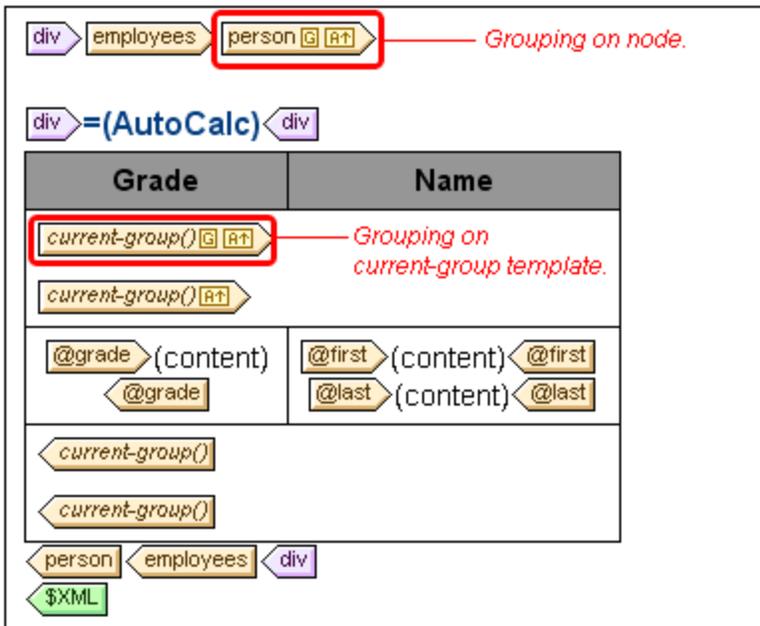
- **group-by**, which groups items on the basis of an XPath-defined key. For example, `car` elements can be grouped on the basis of their `brand` attributes. The grouping is set on the `car` element, and an XPath expression selects the `brand` attribute.
- **group-adjacent** uses a combination of grouping-key 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](#)²⁵⁸. How to use this dialog is described in the section [Sorting](#)²⁵⁸. 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](#)²¹⁹ 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.

7.3.1 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](#)²³, `C:\Documents and Settings\\My Documents\Altova\StyleVision2023\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

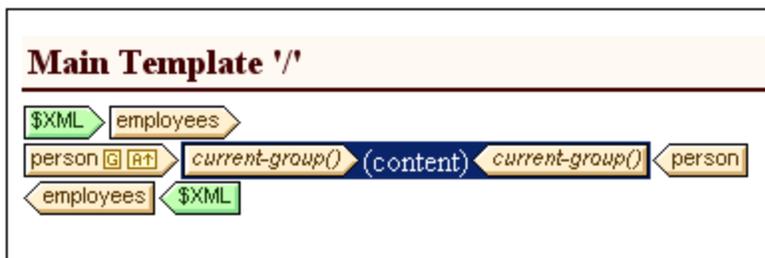
The strategy for creating the groups is as follows. The grouping is created on the `person` element with the `department` attribute being the grouping-key. This causes the `person` elements to be ordered in groups based on the value of the `department` attribute. (If sorting is specified, then the department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

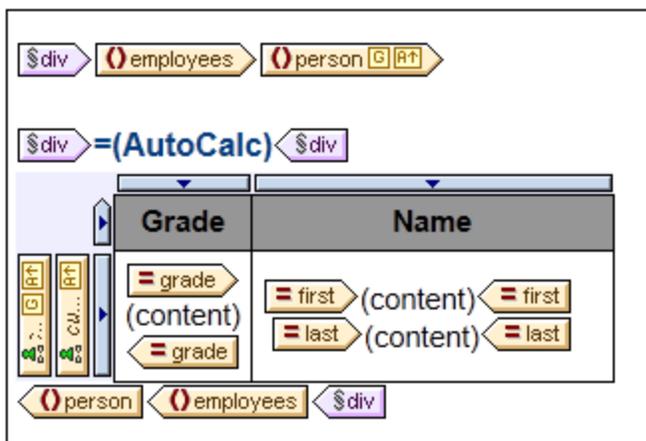
Creating the SPS

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 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.

7.3.2 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](#)²³, `C:\Documents and Settings\<username>\My Documents\Altova\StyleVision2023\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

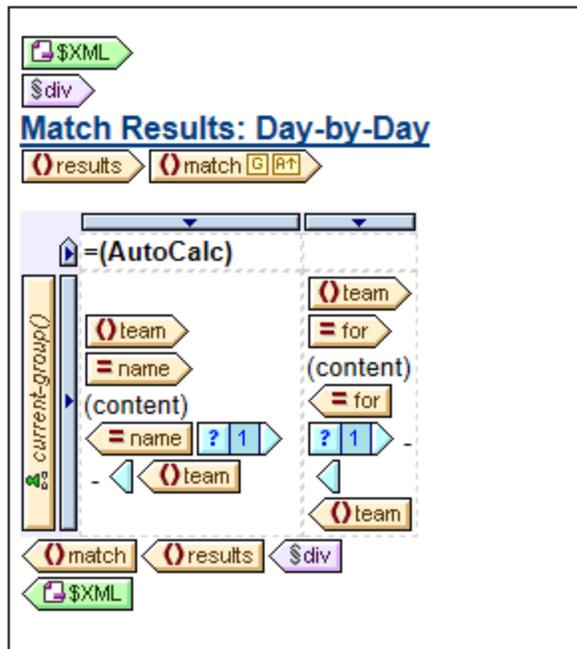
For the two sections containing the match results, we group matches by date and tournament-group. For members of each group (date and tournament group), we create borderless tables (for alignment purposes). So matches played on a single date will be in a separate table, and all the match results of a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

7.4 Sorting

The sorting functionality is available for HTML 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 output.

User-defined templates

[User-defined templates](#)²¹⁹ 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.

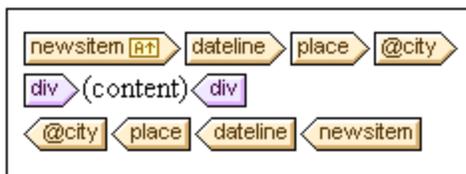
In this section

- The [sorting mechanism](#)²⁵⁸ is described.
- An [example](#)²⁶⁰ demonstrates how sorting is used.

7.4.1 The Sorting Mechanism

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

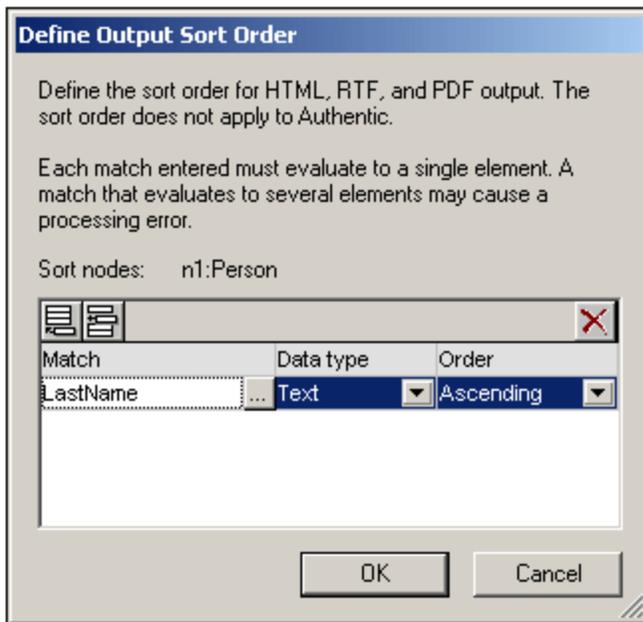
1. In Design View, select the schema element node that is to be sorted. Note that it is the instances of **this** element in the XML document that will be sorted. Often it might not immediately be apparent which element is to be sorted. For example, consider the structure shown in the screenshot below.



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`.

2. After selecting the element to sort, in the context menu (obtained by right-clicking the element selection), click the **Sort Output** command. This pops up the Define Output Sort Order dialog (*screenshot below*), in which you insert or append one or more sort instructions.



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](#)²⁶⁰.

User-defined templates

[User-defined templates](#)²¹⁹ 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.

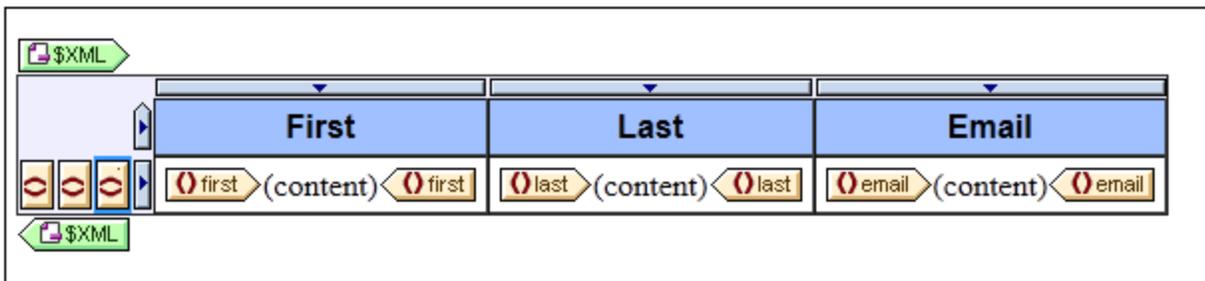
7.4.2 Example: Sorting on Multiple Sort-Keys

In the simple example below (available in the [\(My\) Documents folder](#)²³, `C:\Documents and Settings\\My Documents\Altova\StyleVision2023\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.

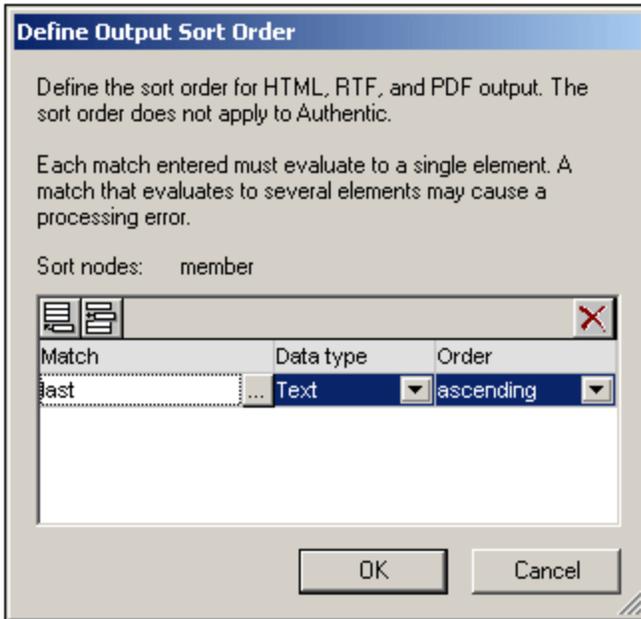
When the list is unsorted, the output order is the order in which the `member` elements are listed in the XML document (*screenshot below, which is the HTML output*).

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.



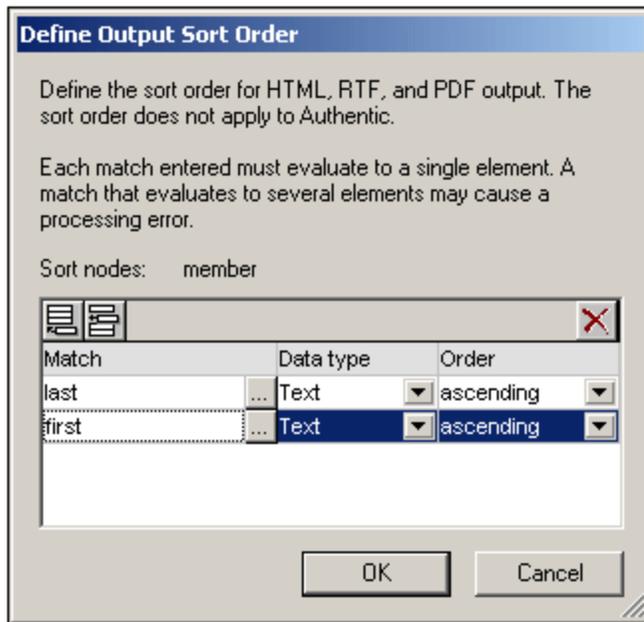
This pops up the Define Output Sort Order dialog (*screenshot below*). Notice that the element selected for sorting, `member`, is named at the Sort Nodes entry. This node is also the context node for XPath expressions to select the sort-key. Click the Add Row button (at left of pane toolbar) to add the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

7.5 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](#)²⁶³ explains how user-defined parameters can be used in an SPS.
- [Parameters for Design Fragments](#)²⁶⁴ describes how parameters can be used with design fragments.
- [SPS Parameters for Sources](#)²⁶⁷ 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](#)²⁶⁸ 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.

7.5.1 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](#).

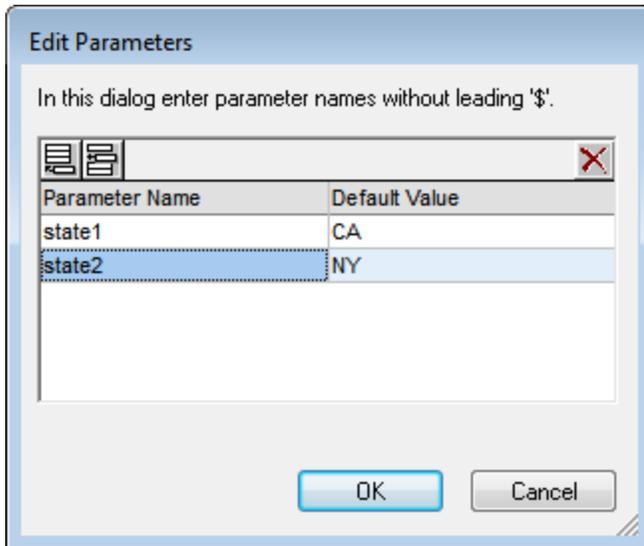
Usage mechanism

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

1. [Declaring the required parameters](#)²⁶³.
2. [Referencing the declared parameters](#)²⁶⁴.

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](#)⁴⁵² command.



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 the SPS.

Note:

- The Edit Parameters dialog contains all the user-defined parameters in an SPS.
- Parameters can also be declared in the [Design Overview sidebar](#)³².

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')`).

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

7.5.2 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 `EEmailAddresses` 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:

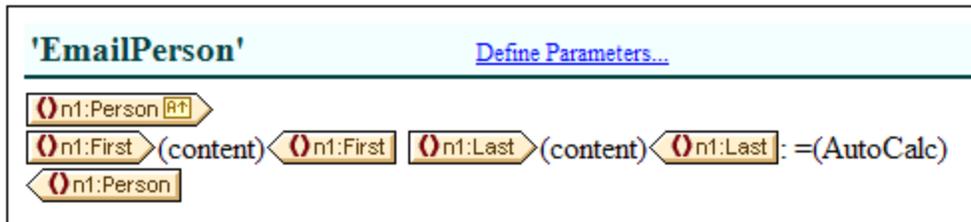
1. [Defining the parameter](#)²⁶⁵ with a default value on the design fragment where it is created.
2. [Editing the parameter value](#)²⁶⁶ where the design fragment is used.

These parts are explained in detail below.

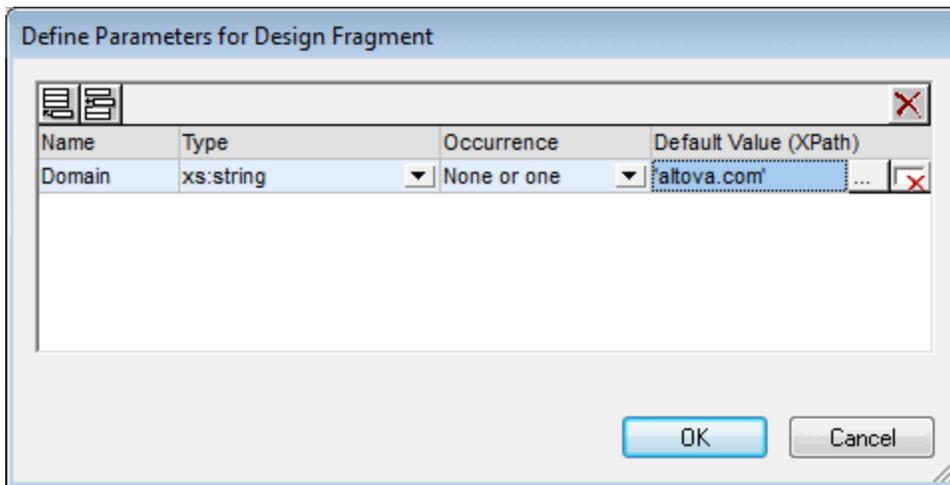
Note: Parameters for Design Fragments are supported in Authentic View only in the Enterprise Editions of Altova products.

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.

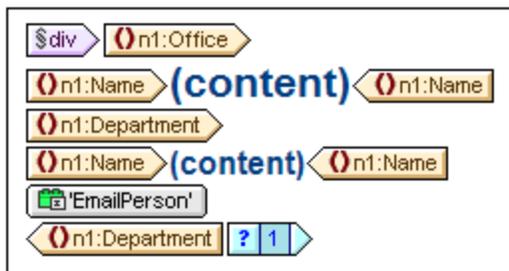


There are two types of **Delete** icon. The Delete icon to the right of each parameter entry deletes the default value of that parameter. The **Delete** icon at the top right of the pane deletes the currently highlighted parameter.

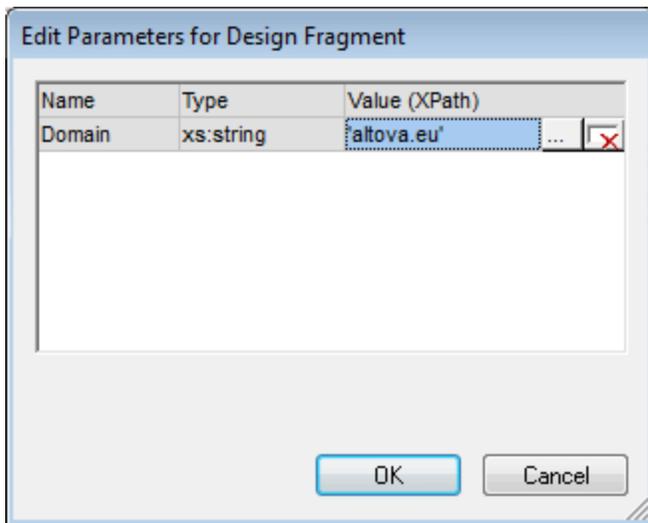
Note: If the SPS uses XSLT 1.0, then the XPath expression you enter must return a node-set. Otherwise an error is reported.

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.

7.5.3 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.

7.5.4 Variables

Using variables consists of two parts: (i) [declaring the variable](#)²⁶⁸, and (ii) [using the variable](#)²⁷⁰.

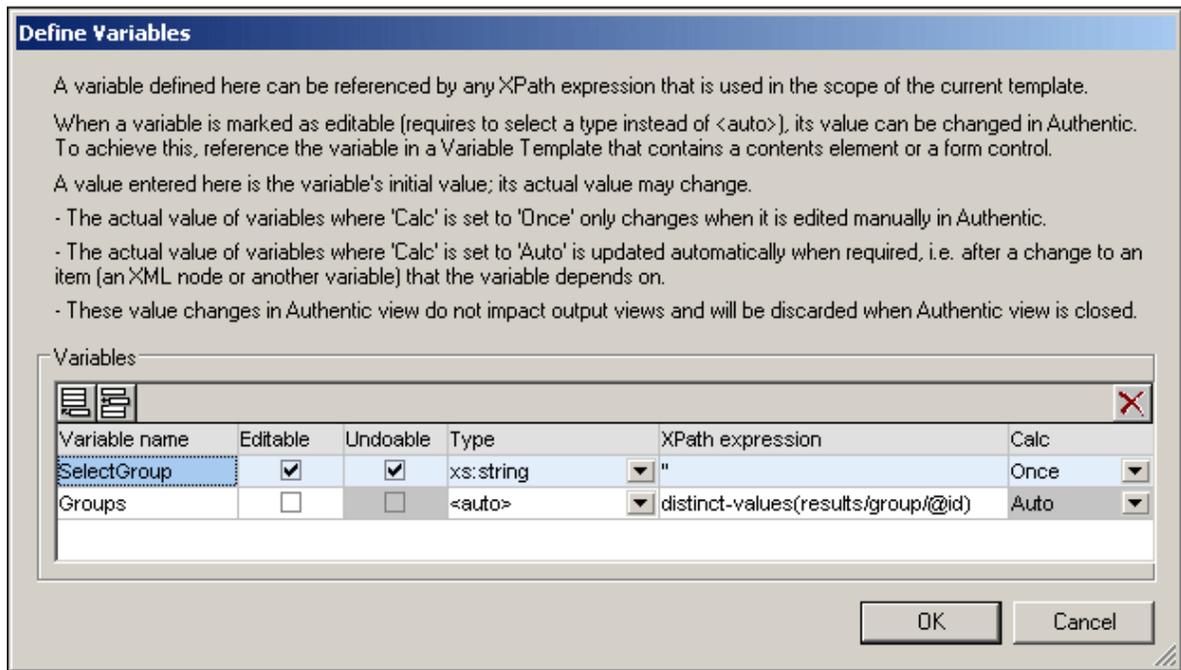
Note: Variables are supported in Authentic View only in the Enterprise Editions of Altova products.

Declaring a variable

A variable can be declared on any template included in the design. It is given a name, a datatype, and a value. Additionally, you can specify whether it is to be editable in the Enterprise editions of Authentic View. The variable will then be in scope on this template and can be used within it. To declare a variable so that it is in scope for the entire document, declare the variable on the root template. A major advantage of declaring a variable only on the template where it is needed is that XPath expressions to locate a descendant node will be simpler.

Declare a variable as follows:

1. Right-click the node template on which the variable is to be created and select the command **Define Variables**.
2. In the Define Variables dialog that appears (*screenshot below*), click the **Append Variable** icon in the top left of the Variables pane, then enter a variable name. The value of the variable is given via an XPath expression. If you wish to enter a string as the value of the variable (as in the first variable in the screenshot below), then enclose the string in quotation marks. In the screenshot below, the value of the `SelectGroup` variable is the empty string. Otherwise, the text will be read as a node name or a function-call.



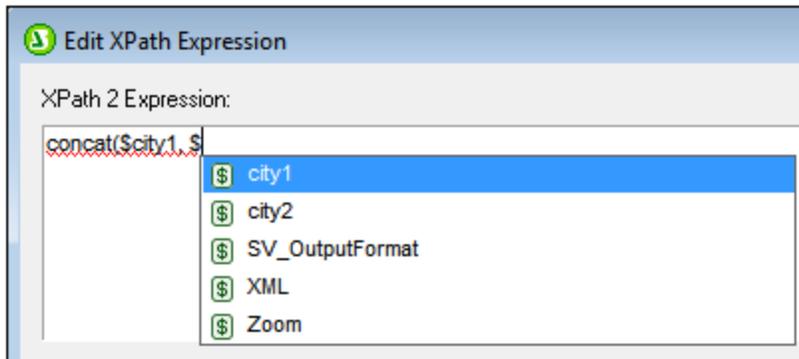
3. Setting a variable to Editable (by checking the *Editable* check box) enables the variable to be edited in Authentic View (which is available as a preview only in Enterprise and Professional editions of StyleVision). In this case, you must also set the datatype value to the correct type, such as `xs:string`. When a variable is editable, the original value set by the SPS designer can be edited when the Authentic View user makes changes to the document in Authentic View. Such changes can be the explicit editing of the variable (such as when the variable value is created as `editable (contents)` or an editable text box and this is edited by the Authentic View user), or when a node or value used in the variable's XPath expression is modified by the Authentic View user.
4. If the variable is set to Editable, then two more options relevant to Authentic View are enabled: *Undoable* and *Calc*. Checking the *Undoable* option generates an Undo step for every change made to the variable. The Authentic View user can therefore click through the Undo cycle to retrieve an earlier value of the variable. The *Calc* value can be either *Once* or *Auto*. If this option is set to *Once*, the variable value is calculated once, when the template containing the variable is evaluated. The value can only be changed when the user explicitly edits the variable (for example, if the variable is created as `editable (contents)` or an editable text box). On the other hand, if this option is set to *Auto*, the variable will be re-calculated also each time a node or value used in the variable's XPath expression is modified.
5. You can add as many variables as you like, but the name of a variable must not be the name of an already declared in-scope variable. To delete a variable click the **Delete** icon in the top right of the pane.
6. Click **OK** when done. The template tag will now have a \$ icon to indicate that one or more variables have been declared on it.

In this way, variables can be created for each node template that is present in the design. Each of these variables will have a name and a value, and will be in scope within the template on which it was declared. To edit a variable subsequently, right-click the node template on which the variable was created and select the command **Define Variables** to access the Define Variables dialog.

Using a variable

For a variable to be used at any location, it must be in scope at that location. This means that a variable can only be used within the template on which it was defined. Variables can also be edited in Authentic View so that users can control the display. The edited value is discarded when the SPS is closed.

A variable can be used in any XPath expression, and is referenced in the XPath expression by prefixing its name with a `$` symbol. For example, the XPath expression `$VarName/Name` selects the `Name` child element of the node selected by the variable named `VarName`.



When you enter an XPath expression in the [Edit XPath Expression dialog](#)³⁹⁷, 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 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](#)²⁹⁸ 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](#)²⁷¹. 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](#)⁹² (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](#)²⁷⁴ 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](#)²⁸¹ (*highlighted in screenshot below*). The TOC template contains the design of the TOC; it can be located anywhere in the design. In the example shown in the screenshot below, the TOC template is located near the top of the document.

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

Stylevision User Manual (excerpt from v2007r3)

Note: This is an excerpt from an outdated version of the StyleVision User Manual used here solely to demonstrate the use of various StyleVision features. The XML document is structured into sections and sub-sections that go down three levels. It contains headlines, paragraphs, lists, tables, images, etc. It does not include all the content, presentation, and usability features of the actual user manual and should not be used as a substitute for the actual user manual.

TOC

Table of Contents: Chapters and Their Sections

MyTOC (num-lvl): (text ref)(.....)(page ref) **MyTOC**

MyTOC (num-lvl): (text ref)(.....)(page ref) **MyTOC**

MyTOC (num-lvl): (text ref)(.....)(page ref) **MyTOC**

----- page break -----

helpproject **topics** **topic** **body** **header** **MyTOC** **MyTOC** **para**

(content) **para** **header** **body** **topic** **topics** **helpproject**

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 the HTML 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](#)⁹² is the selected XSLT version.

2. [Structure the document in TOC levels](#)²⁷⁵. 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](#)²⁷⁶ 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 \(levelrefs\)](#)²⁸¹. 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](#)²⁸⁴. 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](#)²⁸⁴. 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](#)²⁹³, 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).

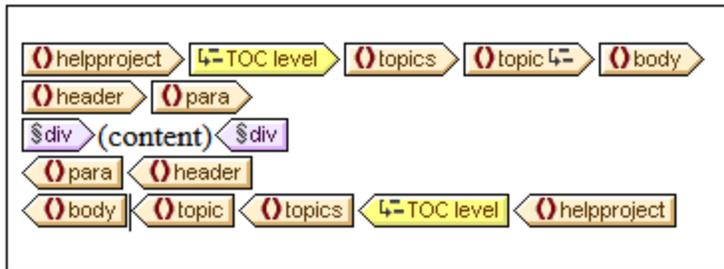
- The **TOC components in the document body** mark out items that will be used in the TOC template.
- The **TOC components in the TOC template** reference the marked items in the document body. Components in the TOC template have the word *'reference'* in their names.

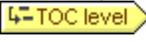
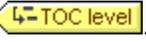
Document body	TOC template
TOC level: The TOC levels structure the document in a nested hierarchy.	Level references (levelrefs): Correspond to the TOC-level structure defined in the document body. Enables TOCrefs in a given level to target TOC bookmarks at the corresponding level.
TOC bookmark: Has a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

7.6.1 Bookmarking Items for TOC Inclusion

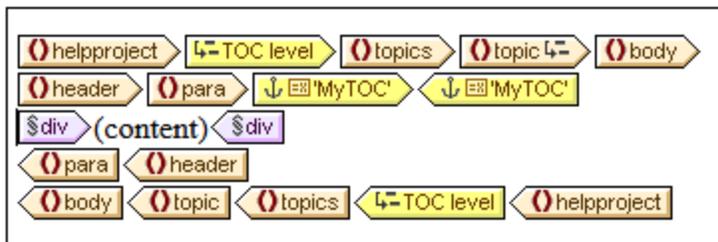
Bookmarking an item in the design for inclusion in a TOC consists of two steps, which can be done in any order:

1. [Structuring the design document in a hierarchy of nested TOC levels](#)²⁷⁵. 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](#)²⁷⁵. When the [TOC template is created](#)²⁸¹, 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](#)²⁷⁸ 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.

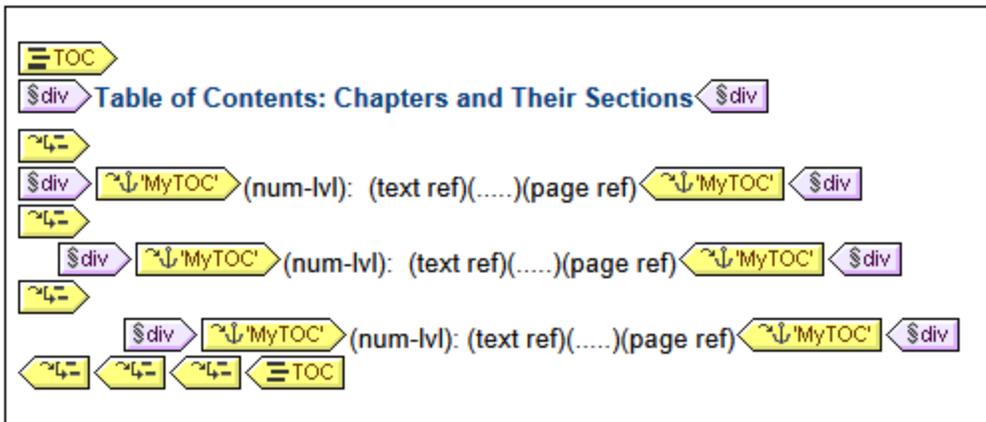


The TOC bookmark serves as an anchor in the document. In the screenshot above, the TOC bookmark (and anchor) is located at the start of `para` element instances. The TOC bookmark has two attributes: (i) a name that will be used to reference the TOC bookmark when creating the TOC item in the TOC template, 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](#)²⁷⁸.

How bookmarked items are referenced in the TOC template

The [TOC template](#)²⁸¹ 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](#) ²⁷⁵, and (ii) how [bookmarks are created](#) ²⁷⁸. How the [TOC template is created](#) ²⁸¹ is described in the section, [Creating the TOC Template](#) ²⁸¹.

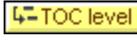
7.6.1.1 Structuring the Design in TOC Levels

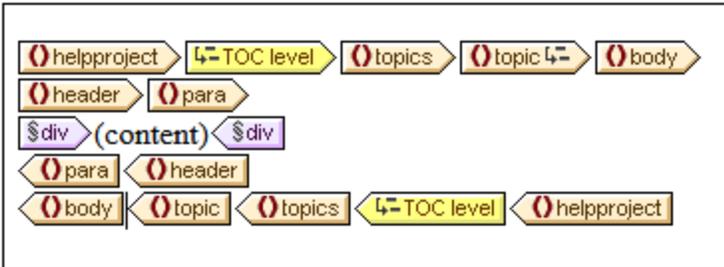
The hierarchical structure you wish to design for the TOC is specified as a set of **nested levels**. As such it is a hierarchical structure which, although related to the XML document structure, is separate from it. This structure is specified in the SPS document design. The TOC template that you construct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

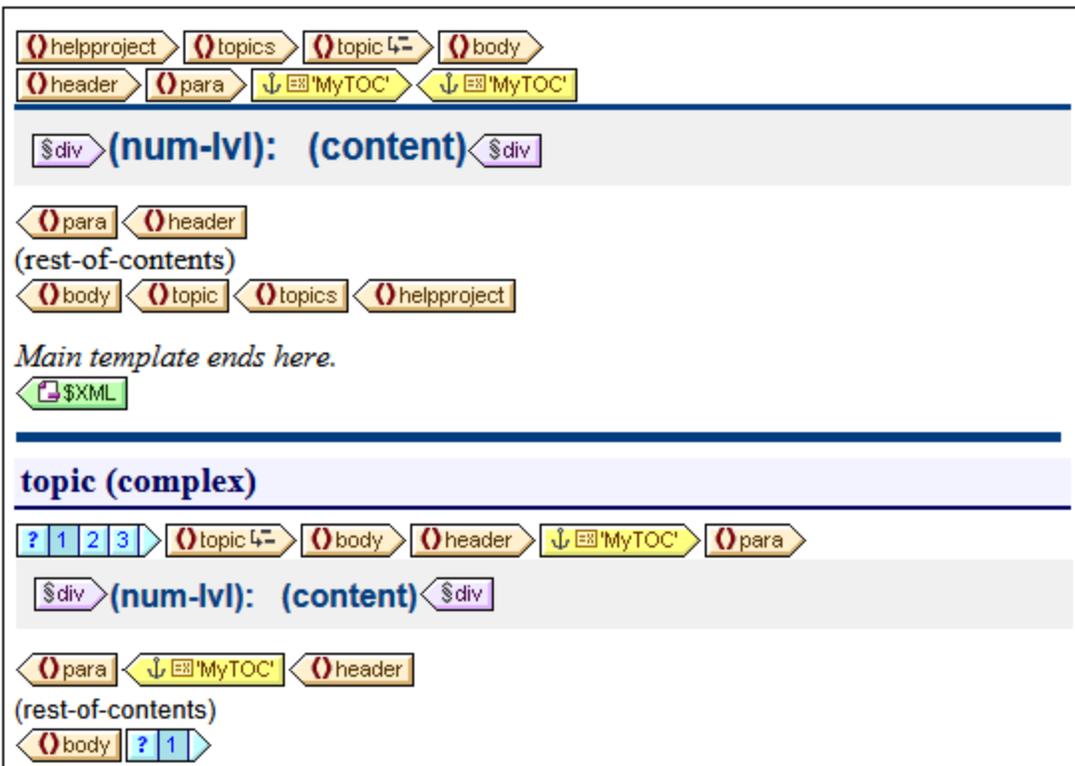


To create a level, do the following:

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 4=`. 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 4=`). 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

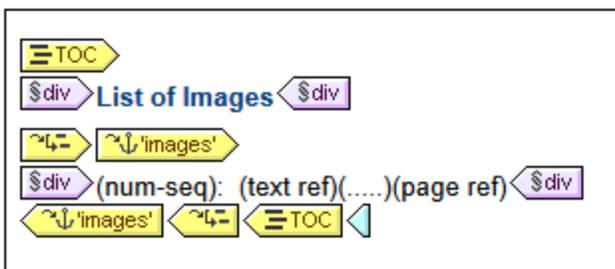
In a flat TOC hierarchy, TOC items will be output at a single level: for example, a simple list of the images in a document.

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.

7.6.1.2 Creating TOC Bookmarks

TOC bookmarks are created within a [TOC level](#)²⁷⁵ 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.

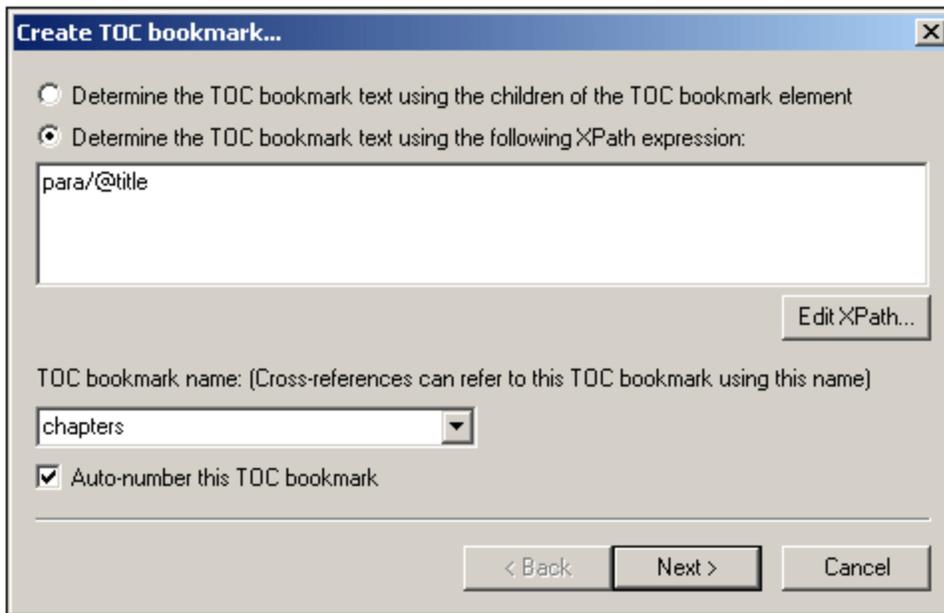
You can create a TOC bookmark in two ways:

- By using the [Create TOC Bookmark Wizard](#)²⁷⁸, 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](#)²⁸⁰, the properties of which will be defined subsequently.

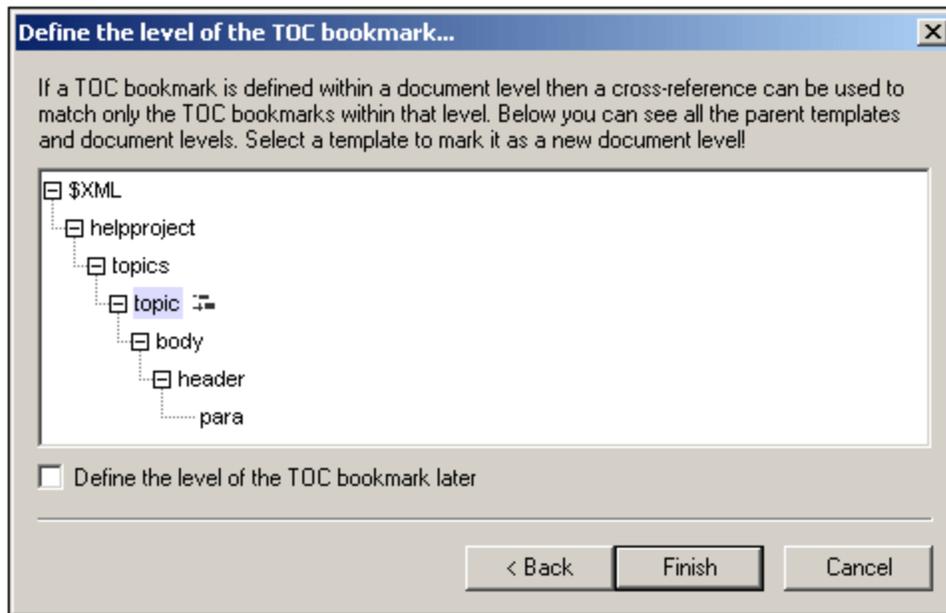
Creating the TOC bookmark with the Create TOC Bookmark Wizard

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](#)²⁸¹ (see *below*).

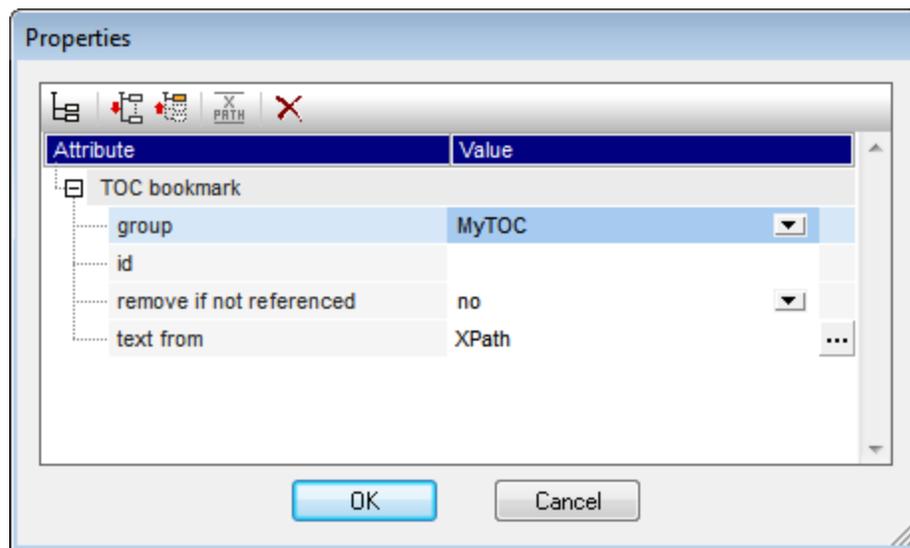
Inserting hierarchical or sequential numbering for a component

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](#)⁴⁴ or right-click the TOC bookmark and select the property you wish to edit.



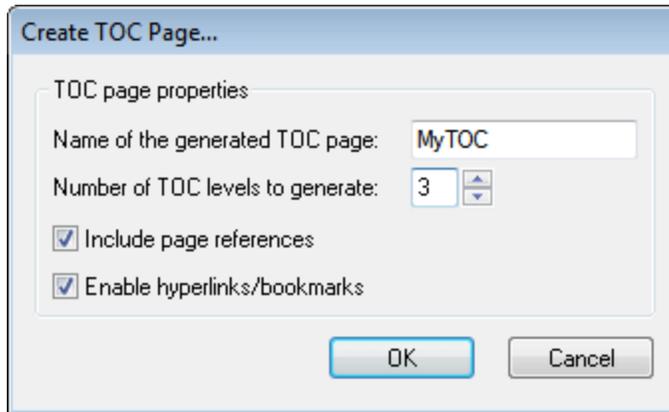
The TOC bookmark has the following properties: (i) the name of the TOC bookmark group (*Group*); (ii) a unique ID; (iii) an option to remove the bookmark if it is not referenced; and (iv) an option (*Text From*) to specify the text entry, which could come from the bookmark's content or from an XPath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

7.6.2 Creating the TOC Template

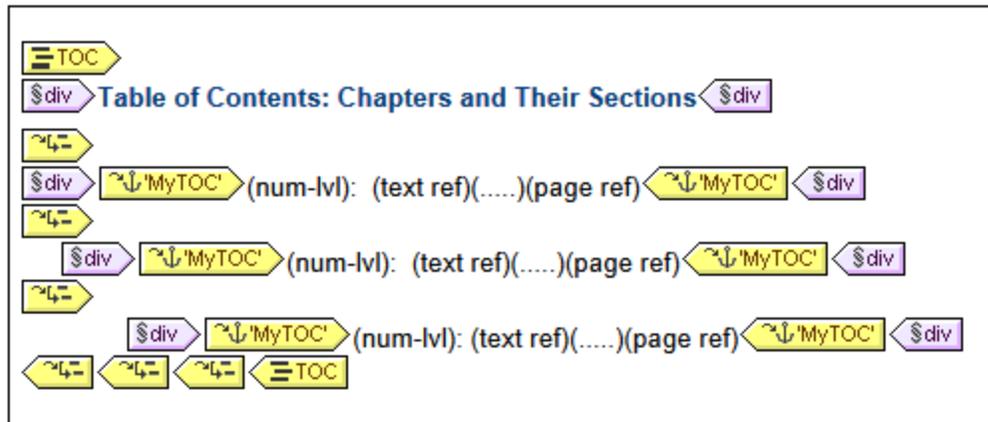
The TOC template is the template that produces the table of contents in the output. It can be created anywhere within the SPS design, and multiple TOC templates can be created in a single SPS design.

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.)



3. Enter the information requested in the dialog: (i) The name of the generated TOC page is the (TOCref) name that will be used to reference the [TOC bookmarks](#) ²⁷⁸ 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](#) ²⁸⁴). (ii) The number of [TOC level references \(levelrefs\)](#) ²⁷⁵ specifies how many level references the TOC is to have. (iii) For printed media, the option to output page references (i.e. page numbers) is available. (iv) The text entries in the TOC can be used as links to the TOC bookmarks.
4. Click **OK** to finish. The TOC template is created with the specified number of levelrefs (*screenshot below; the formatting of the TOC template has been modified from that which is created initially*).



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](#) ²⁸⁴.

Editing the TOC template

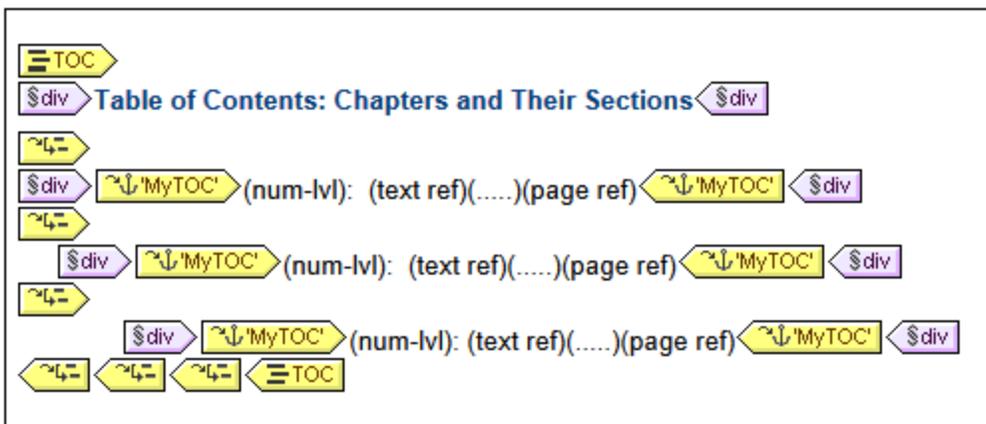
The following editing options are available:

- 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](#) ²⁸³ can be added to or deleted from the structure of the TOC template.
- The [properties of individual TOC references](#) ²⁸⁴ (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](#)²⁸⁴ can be added to or deleted from any levelref in the TOC template.
- The [TOC item](#)²⁸⁴ within a TOCref can be formatted with CSS properties using the standard [StyleVision mechanisms](#)³¹⁹.
- Standard SPS features (such as images, Auto-Calculations, and block-formatting components) can be inserted anywhere in the TOC template.

7.6.2.1 Levelrefs in the TOC Template

The [TOC template](#)²⁸¹ 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](#)²⁸¹.



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](#)²⁷⁸ within a [specified scope](#)²⁸⁴ 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.

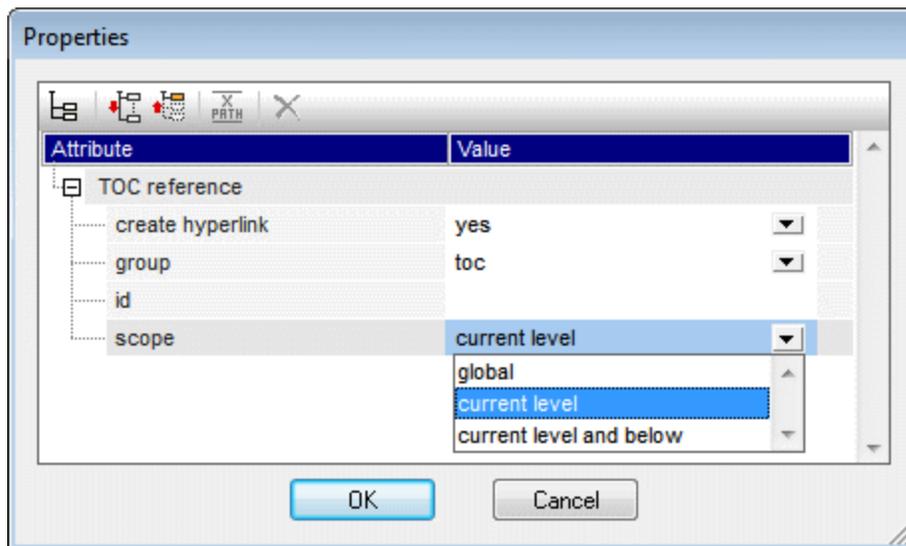
7.6.2.2 TOC References: Name, Scope, Hyperlink

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*).



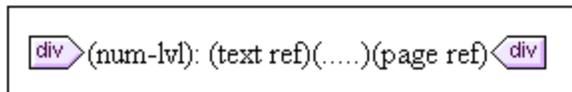
Alternatively, with the TOCref tag selected, go directly to the required property in the Properties window (*TOC reference* group of properties).

7.6.2.3 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

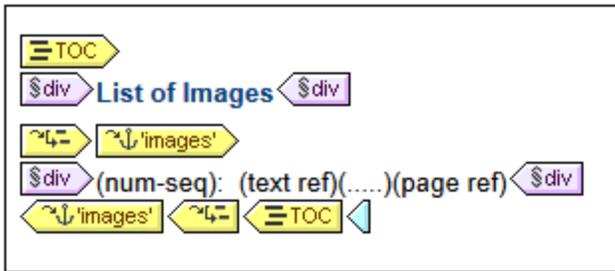
In a flat TOC hierarchy, TOC items will be output at a single level: for example, a simple list of the images in a document.

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.



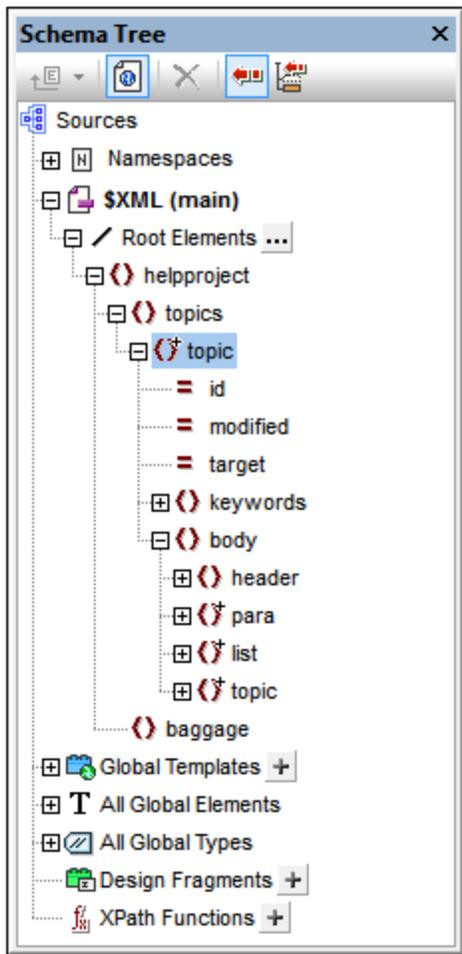
If the design contains more than one level, and a flat TOC is required, say, for items in the second level, then the TOC template could have two levelrefs with a TOCref only within the second level (no TOCref within the first level). Alternatively, the scope property of TOCrefs can be used to specify 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](#)³¹⁹ via the [Styles sidebar](#)³²⁵. Individual TOC item components can be separately formatted by selecting the component and assigning it [style properties](#)³²⁵ in the Styles sidebar.

7.6.3 Example: Simple TOC

An example SPS file to demonstrate the basic use of TOCs, called `ChaptersSimple.sps`, is in the [\(My\) Documents folder](#)²³, `C:\Documents and Settings\\My Documents\Altova\StyleVision2023\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](#)²⁹⁰.)



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](#)²⁷⁵: 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](#)²⁷⁶: 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](#)²⁸¹: 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.



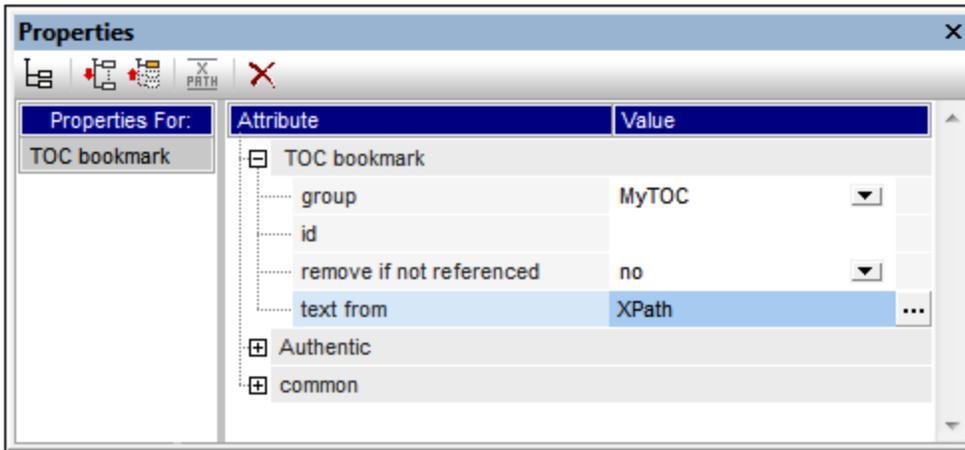
The screenshot above shows the contents of the first conditional branch, for first-level, chapter-type `topic` elements. Note that a TOC level has been created on the template start-tag of this `topic` element. In the other two conditional branches no TOC level has been created on the topic template.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



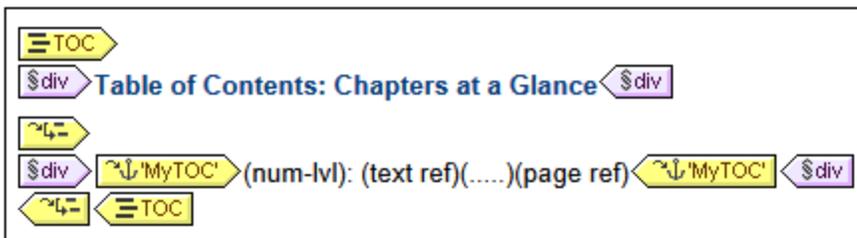
The properties of the TOC bookmark can be edited in the Properties sidebar (*screenshot below*).

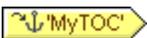


The *Group* property sets the TOC bookmark group (and is the name of the TOC bookmark). In our example, we have specified the value `MyTOC` for this property. The bookmark group will be referenced in the TOC when it is created, and it enables different TOC groups to be specified within the same level.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

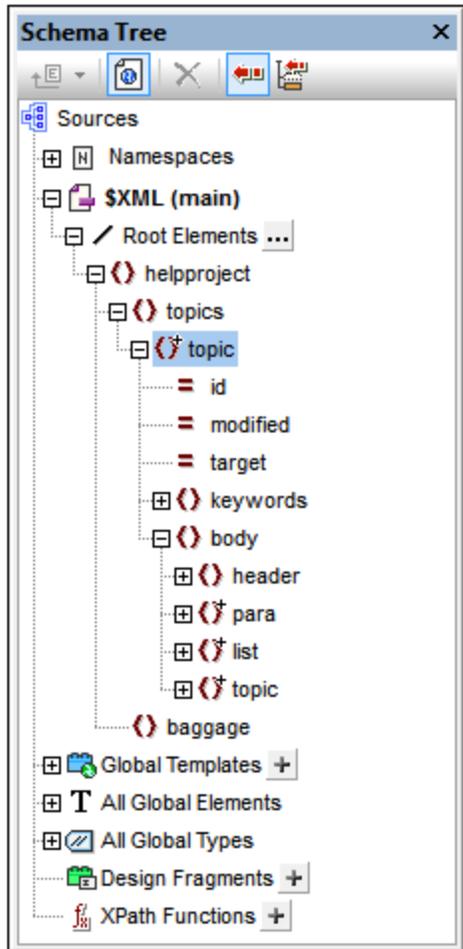


Within this levelref, a TOC reference (TOCref)  has been inserted. This TOCref has been set to select bookmarks (i) that are in the bookmark group named `MyTOC` (see 'Creating the TOC bookmark' above), and (ii) that are within the scope of the current level only. These settings can be made in the Properties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

7.6.4 Example: Hierarchical and Sequential TOCs

An example SPS file to demonstrate the use of TOCs, called `Chapters.sps`, is in the [\(My\) Documents folder](#)²³, `C:\Documents and Settings\\My Documents\Altova\StyleVision2023\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](#)²⁹¹, which lists the names of each chapter (the first-level topics).
2. [Chapters and their sections](#)²⁹², 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](#)²⁹², 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).



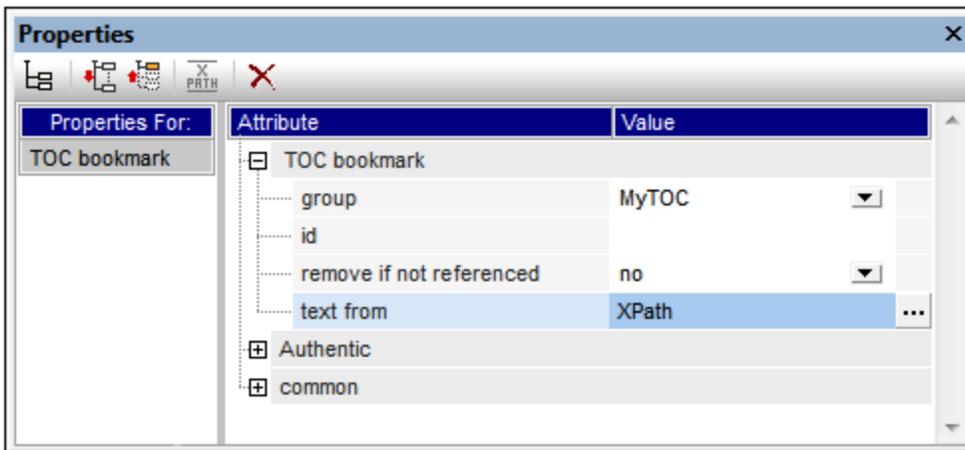
Since the `topic` element is recursive, the TOC level and the TOC bookmark will also recurse. This means that, at the first recursion, a new hierarchically subordinate TOC level and a new TOC bookmark is created. This process continues for each descendant topic, thus creating a hierarchy of descendant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

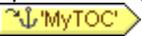
Notice that hierarchical numbering (`num-lvl`) has been inserted within the level. This is done by right-clicking at the required location and selecting **Insert Table of Contents | Hierarchical Numbering**. The effect is to insert the correct hierarchical number before each topic title in the **document's 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.

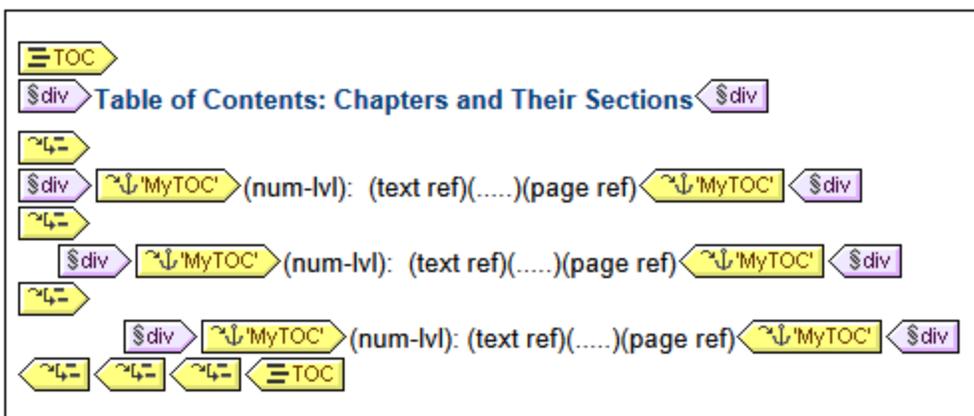


The TOC template itself (*screenshot below*) contains one level reference (levelref) , and the TOCref within that levelref  has been set to select TOC bookmarks named `MyTOC` within the scope of the current level only—which is the first level. As a result, TOC items will be created only for first-level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

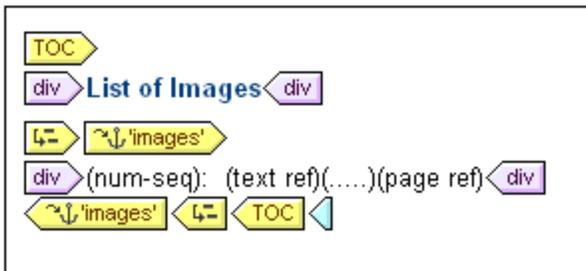
topic elements, `image` elements will occur within the first, second, and/or third levels of the document. There is therefore no need to create any level for the `image` element.

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.

7.6.5 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.

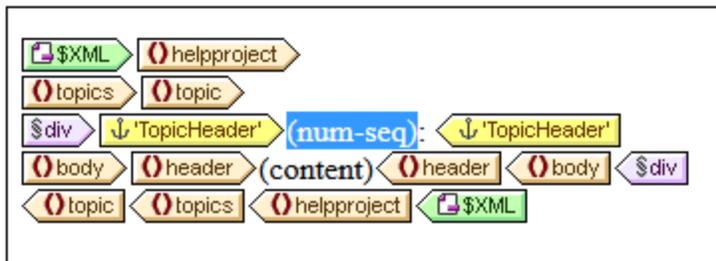
Auto-Numbering can be either sequential (flat) or hierarchical. Sequential numbering provides ordinary numbering on a single level. Hierarchical numbering is based on the TOC-level hierarchy created in the document and creates numbering according to the element's position in the TOC-level hierarchy.

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](#)²⁷⁸ 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*).



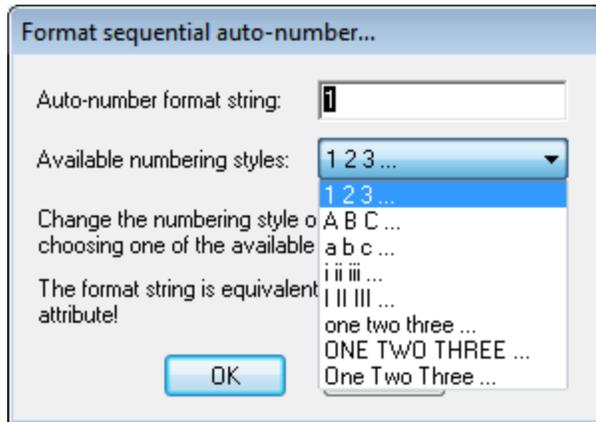
3. If the TOC bookmark is going to be referenced from within a TOC template, then you can enter TOC bookmark properties as required. However, if the TOC bookmark is going to be used only for sequential numbering, there is no need to name it. If you wish to name it, right-click it and select the **Edit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

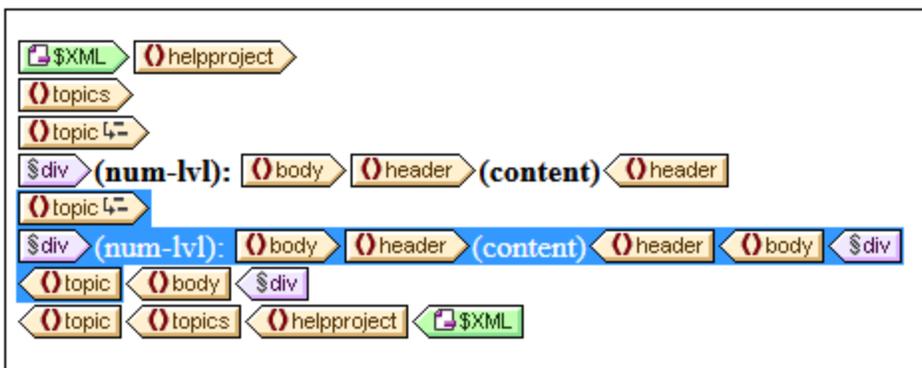


Select the format you want from the dropdown box of the *Available numbering styles* combo box (see *screenshot above*) and click **OK** to apply the selected format.

Hierarchical numbering (num-lvl)

Hierarchical numbering can be inserted within a [TOC level in the design](#)²⁷⁵. 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](#)²⁷⁵. 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*).



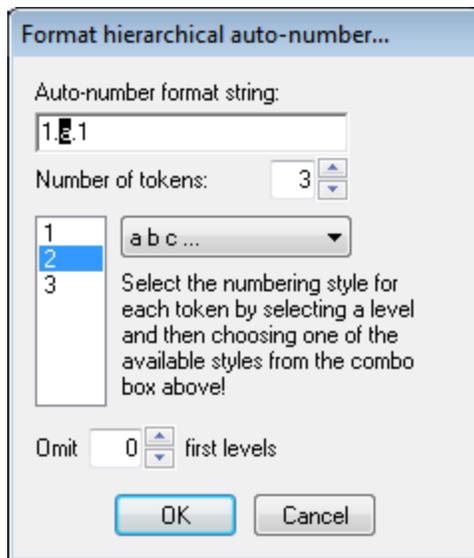
In the screenshot above, there are two levels. The `topic` element is recursive, and a level has been created on two `topic` elements (by right-clicking the node tag and selecting **Template Serves as Level**). One `topic` element (*highlighted in the screenshot above*) is nested within the other. As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

7.6.6 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:

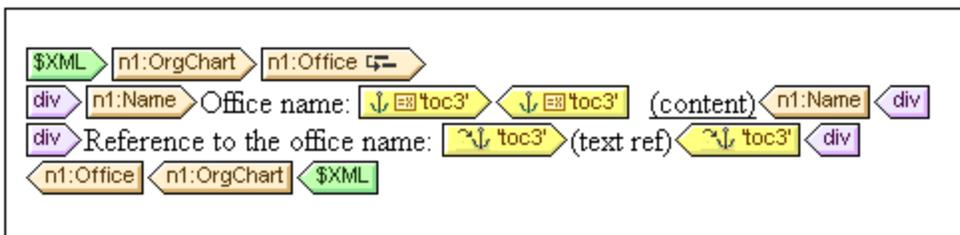
Step 1: Levels

The document is structured into TOC levels as described in the section [Structuring the Design in Levels](#)²⁷⁵. 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.

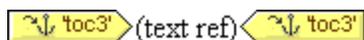
In the screenshot below, the TOC bookmark within the `n1:Name` element  has a name of `toc3` and an XPath expression that locates the current node. This means that the output text will be the contents of the `n1:Name` node.



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.

7.6.7 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](#)²⁹⁸ can be inserted in the SPS.
- How [hyperlinks](#)³⁰⁰ 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.

7.6.7.1 Inserting Bookmarks

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

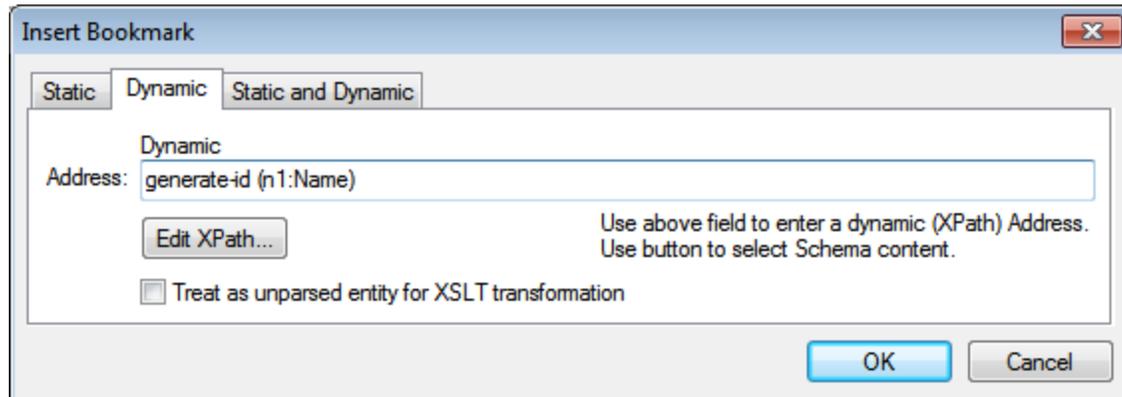
Bookmarks are created in the SPS via the Insert Bookmark dialog (*screenshot below*). In this dialog you define the name of the bookmark. The name can be a static name, or it can be a dynamic name that is (i) derived from XML document content, or (ii) generated arbitrarily with an XPath expression.

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](#)⁴⁶⁶, 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.

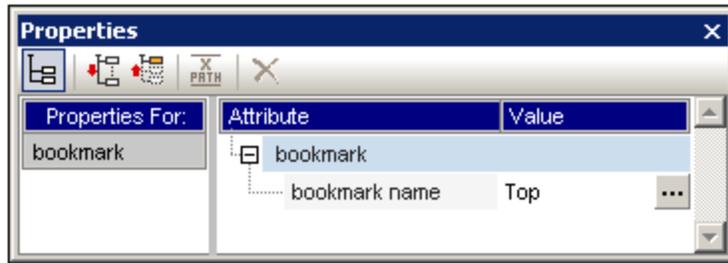
After a bookmark has been created, it can be [linked to by a hyperlink](#)³⁰⁰.

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](#)³⁰³. 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.

7.6.7.2 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.

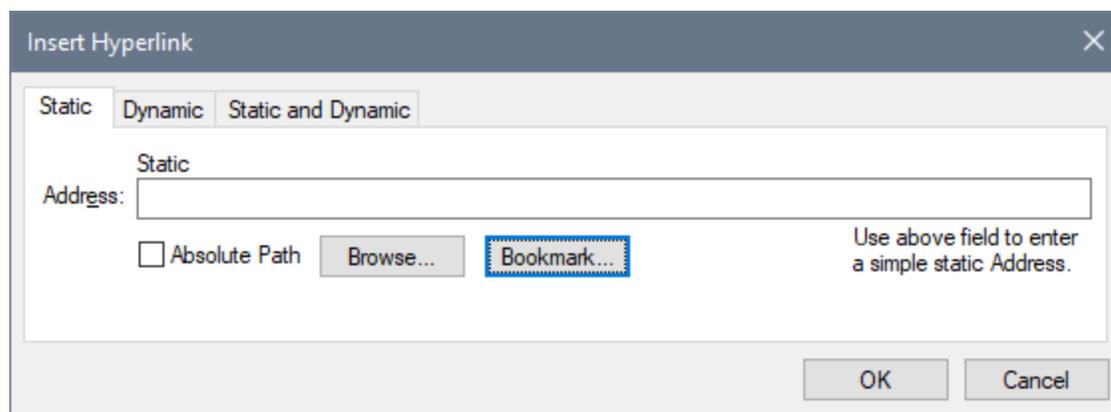
Creating hyperlinks

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](#)⁴⁷⁶ or [Insert | Hyperlink](#)⁴⁶⁷.



The target of a link can be either:

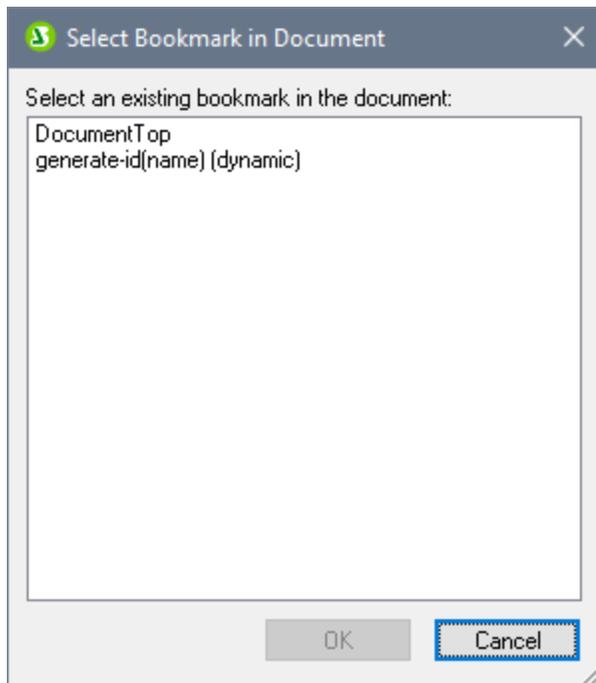
- A [bookmark](#)³⁰¹ in the same SPS design (in which case the target URI must be a fragment identifier),
- [Dynamically generated](#)³⁰³ to match bookmark anchors (these URIs are also fragment identifiers),
- An [external resource](#)³⁰³; the URI can be static (directly entered), dynamic (taken from 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.

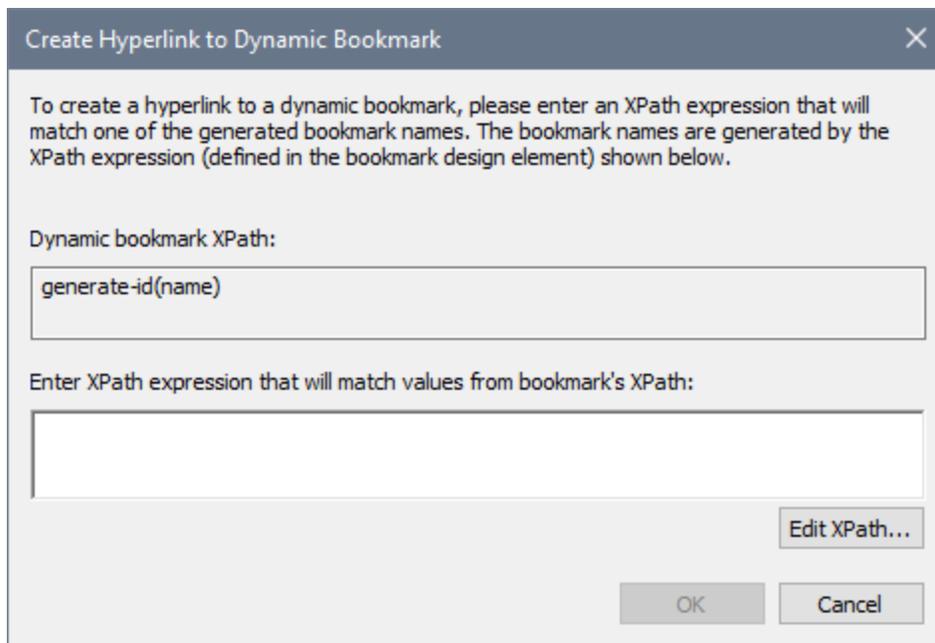
Linking to bookmarks

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](#)²⁹⁸ 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](#)²⁹⁸. 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](#)⁴⁶⁷ 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](#)⁴⁶⁷. 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.

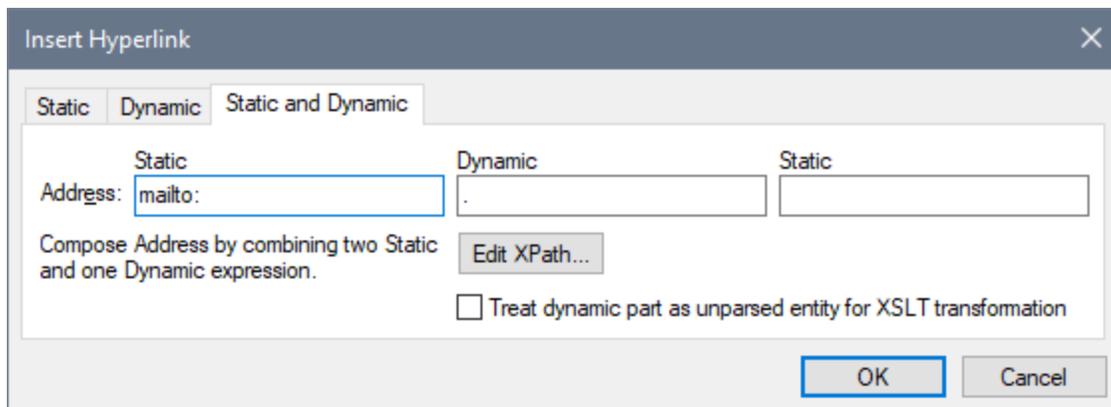
Two points should be borne in mind:

- 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](#)³³⁸ 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](#)⁴⁰⁹ to help you build the dynamic part of the hyperlink.



How to use unparsed entities is described in the section [Unparsed Entity URIs](#)³³⁸.

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.

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*).

Note: Printed-page based output formats, such as RTF, PDF, Word 2007+ and Text (as opposed to HTML), are not supported in the Basic Edition of StyleVision. As a result, all features relating to such output (such as adding page headers and footers) are disabled in the Basic Edition. To be able to use these features, you must obtain a license for the Professional Edition (includes RTF and Text output) or Enterprise Edition (includes RTF, PDF, Word 2007+ and Text). Please see the StyleVision edition comparison page on the [Altova website](#) for more information.

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](#)³¹⁹ 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](#)³²⁹. 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](#)³⁰⁶ 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](#)³⁰⁶ sub-section of this section.

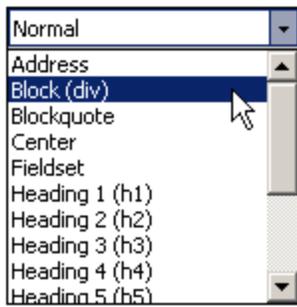
8.1 Predefined Formats

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](#)³²⁵ 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](#)³²⁵.

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

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
<code>&amp;</code>	<code>no</code>	<code>&amp;</code>

&	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](#)³¹⁰) 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 the HTML 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](#)³¹⁰) for which Value Formatting is available.

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

- how the [Value Formatting mechanism works](#)³¹⁰, and
- the [syntax](#)³¹³ 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 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`

Note: Not all formats are available in Basic Edition since Authentic View is not supported in Basic Edition.

8.3.1 The Value Formatting Mechanism

Value Formatting can be applied to:

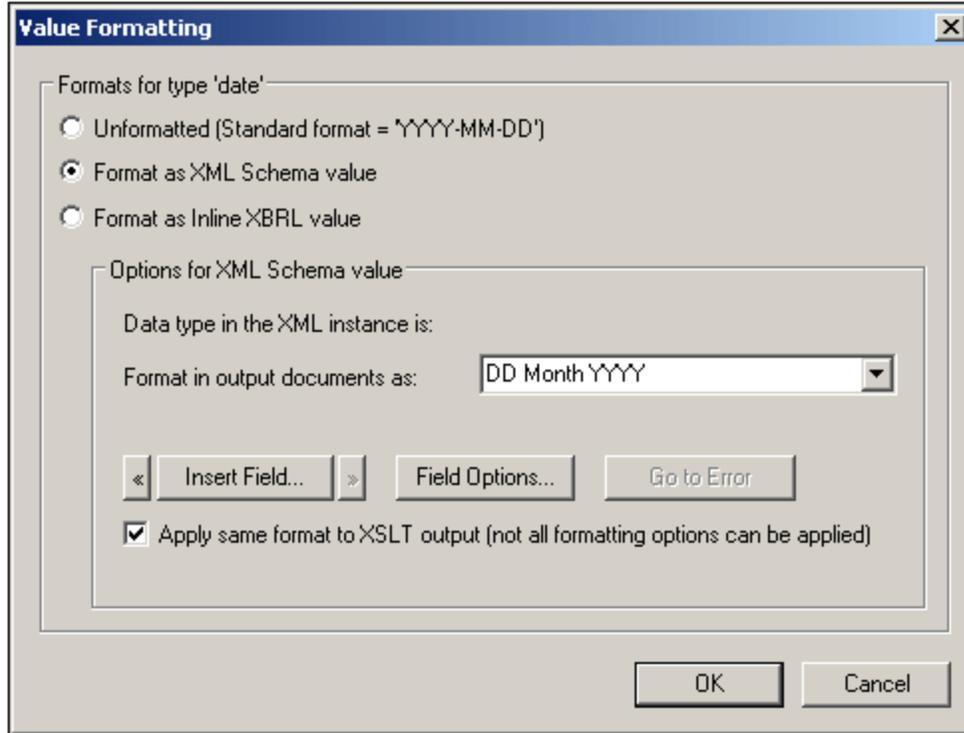
- A [numeric datatype node](#)³¹⁰, such as `xs:decimal` or `xs:date` that is present in the SPS as **contents or an input field**.
- An Auto-Calculation that evaluates to a value which has the lexical format of a [numeric datatype](#)³¹⁰.

Defining Value Formatting

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

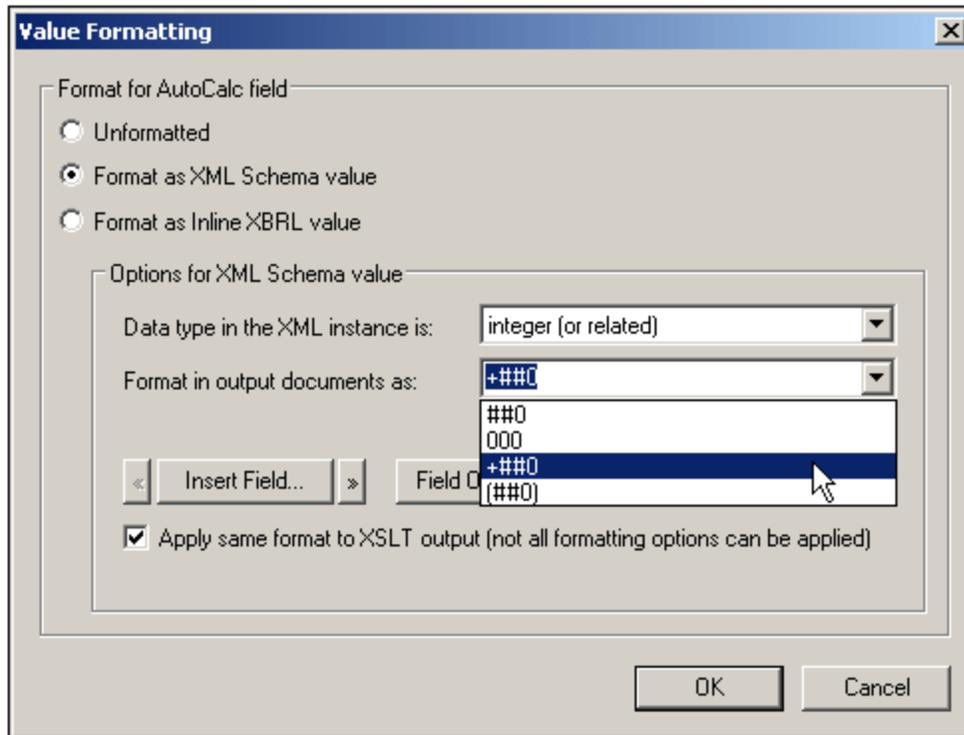
1. Select the `contents` placeholder or input field of the node, or the Auto-Calculation.
2. In the Properties sidebar, select the item, and then the *Content* group (or *AutoCalc* group) of properties. Now click the Edit button  of the `Value Formatting` property. Alternatively, right-click

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](#)³¹³ 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 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.

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**).

8.4 Working with CSS Styles

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

- In [external CSS stylesheets](#)³²⁰. External CSS stylesheets can be added via the [Design Overview](#)³² sidebar and via the [Style Repository](#)⁴¹ sidebar.
- In [global styles](#)³²³ 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](#)⁴¹ sidebar.
- [Locally](#)³²⁵, 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](#)⁴³ sidebar.

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

Terminology

A CSS stylesheet consists of one or more style rules. A rule looks like this:

```
H1 { color: blue }
```

or

```
H1 { color: blue;
      margin-top: 16px; }
```

Each rule has a selector (in the examples above, `H1`) and a declaration (`color: blue`). The declaration is a list of properties (for example, `color`) with values (`blue`). We will refer to each property-value pair as a style definition. In StyleVision, CSS styles can be defined in the [Styles](#)⁴³ sidebar (local styles) and [Style Repository](#)⁴¹ 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](#)²⁰¹.

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](#)⁴⁴³ whether the HTML output documents should be styled according to [Internet Explorer's older or newer interpretation](#)⁹³. You should then set CSS styles according to the selected interpretation. For more details, see [Properties: CSS Support](#)⁴⁴³.

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:

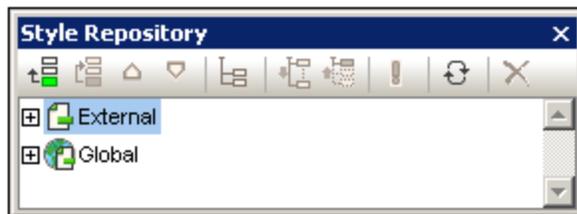
- [Adding an external CSS stylesheet to the SPS](#)³²⁰
- [Viewing the contents of an external CSS stylesheet and modifying the media applicability](#)³²¹
- [Changing the precedence](#)³²²
- [Switching between the full CSS stylesheet set and a single CSS stylesheet](#)³²²

External CSS stylesheets can be managed from two sidebars: the [Style Repository sidebar](#)⁴¹ and the [Design Overview sidebar](#)³². 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](#)³² sidebar.

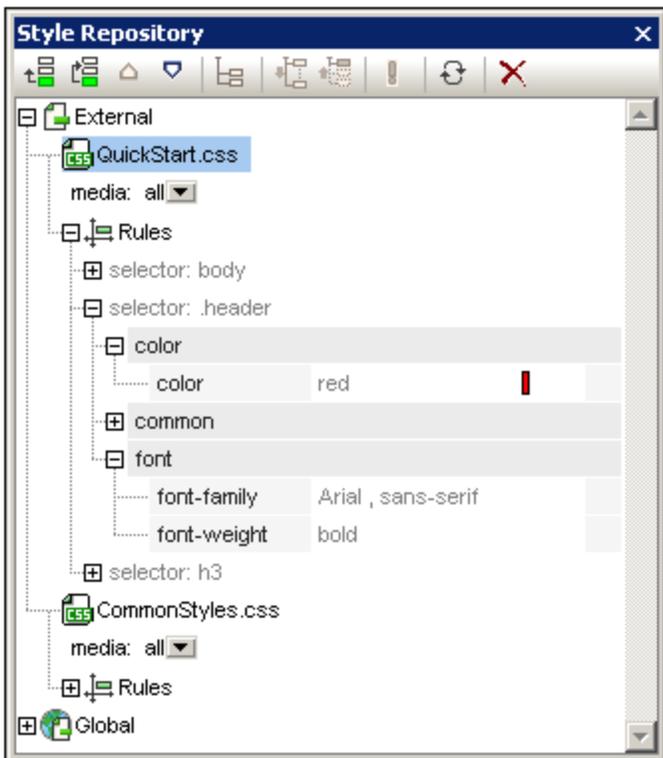
Viewing and modifying the tree of external CSS stylesheets

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:

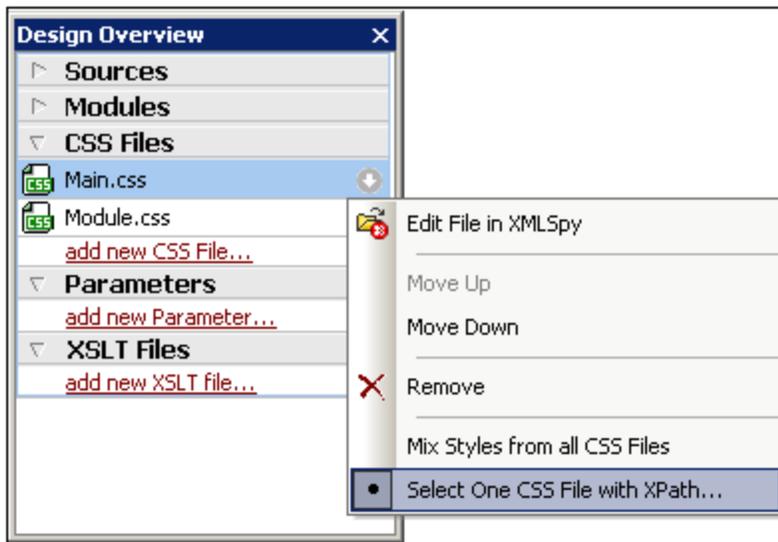
```
<html>
  <head>
    <style>
      <!--
        @import url("ExternalCSS-1.css");
        @import url("ExternalCSS-2.css") screen;
        @import url("ExternalCSS-3.css") print;
      -->
    </style>
  </head>
  <body/>
</html>
```

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.

- *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, 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 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

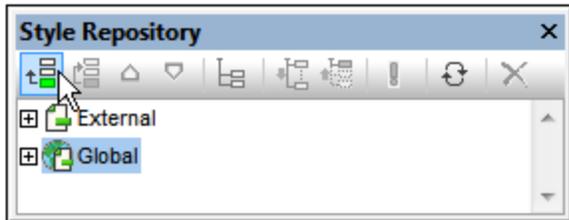
The following [selectors](#) are supported:

- *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`

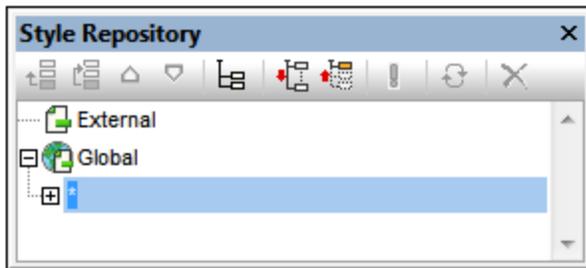
Adding a global style

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](#)³²⁷.
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](#)³²⁷. (The style properties can be displayed in three possible views. These views and how to switch between them are described in [Views of Definitions](#)⁴⁴.)

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>
</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](#)⁴¹⁶ or in the [Styles](#)⁴¹³ sidebar. (This is as opposed to global styles, which are defined in the [Style Repository](#)⁴¹ sidebar.)

Local styles via the Format toolbar

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

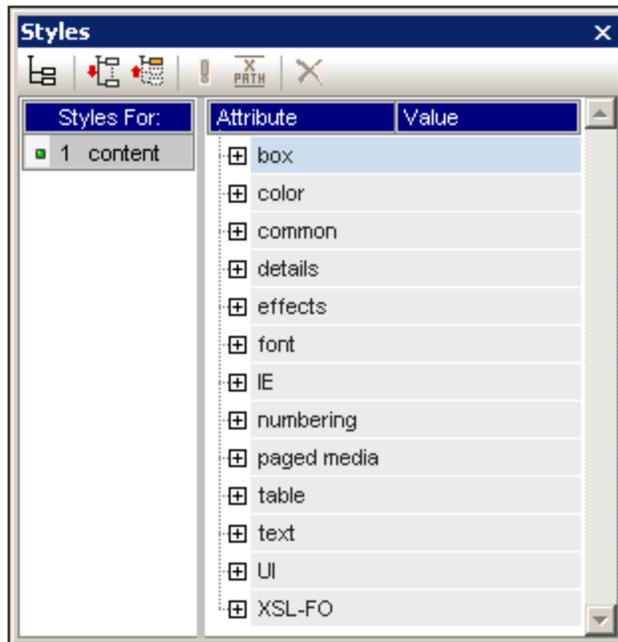


You can apply predefined HTML formatting (such as `div`, `h1`, `pre`, etc), text styling, background color, text alignment, lists, and hyperlinks. See the section, [Format toolbar](#)⁴¹⁶, for details.

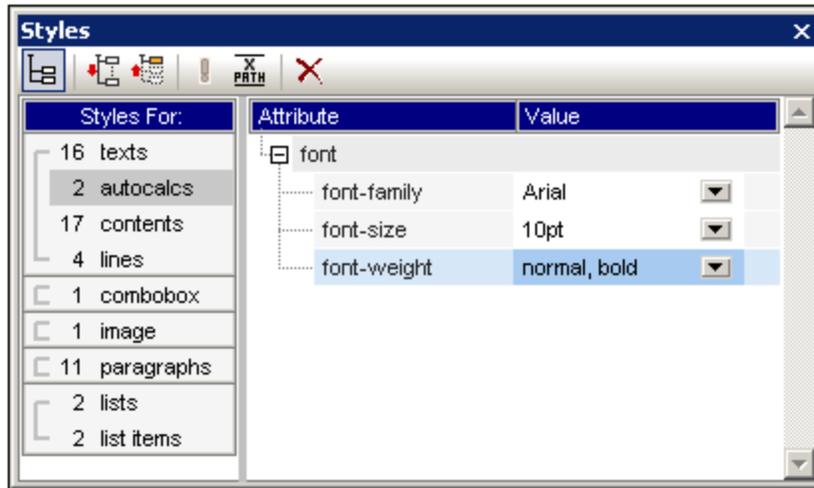
Local styles via the Styles sidebar

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

1. The component to be styled is selected in Design View. Any component in the design except node tags can be styled. The component selected in Design View then appears in the *Styles-For* column of the Styles sidebar (*see screenshot below*). In the screenshot below, a `content` component was selected in Design View and consequently appears in the *Styles-For* column.



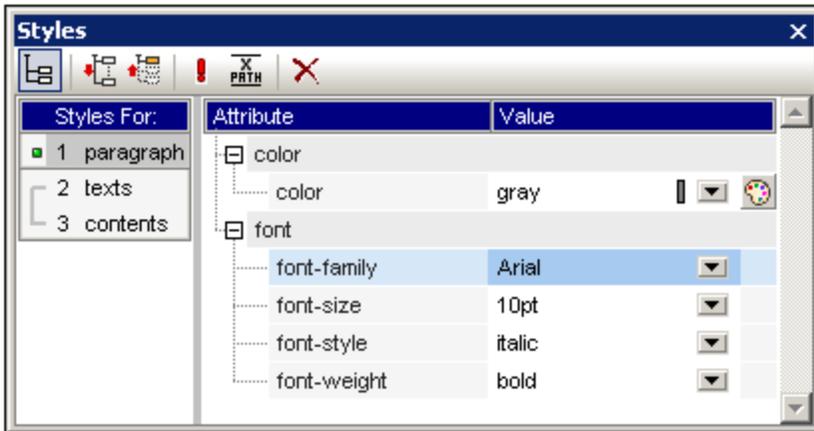
Very often, the component selected in Design View might contain other components. In this case all the components in the selection are displayed, organized by component-type, in the *Styles-For* column of the Styles sidebar. The screenshot below shows the different component-types contained in the Design View selection. To the left of each component-type is the number of instances of that component-type in the selection. For example, in the screenshot below, there are 16 text components and two Auto-Calculation components (among others) in the Design View selection. You can also select a range of components by keeping the **Shift** key pressed while selecting the second, end-of-range component.



2. After making the selection in Design View, you select, in the *Styles-For* column, the **component-type** you wish to style. If there is more than one instance of the component-type you select, then the styles you define will be applied to all these instances. So, for example, if you select the *16-texts* item of the screenshot above, then the styles you define (*see Step 3 below*) will be applied to all 16 `text` components. If you wish to style, say, four of these text components differently, then you must select and style each of the four components separately. If two components of the same component-type have been styled differently and both are selected in Design View, then the styles of both instances are displayed in the *Style Definitions* pane. In the screenshot above, for example, while one Auto-Calculation has a normal font-weight, the other has a bold font-weight. When the *2-Autocalcs* item is selected in the *Styles-For* pane, both font-weights are displayed.
3. After selecting, in the *Styles-For* column, the component-type to style, styles are defined in the [Style Definitions pane](#)⁴³. How to do this is described in the section [Setting Style Values](#)³²⁷.

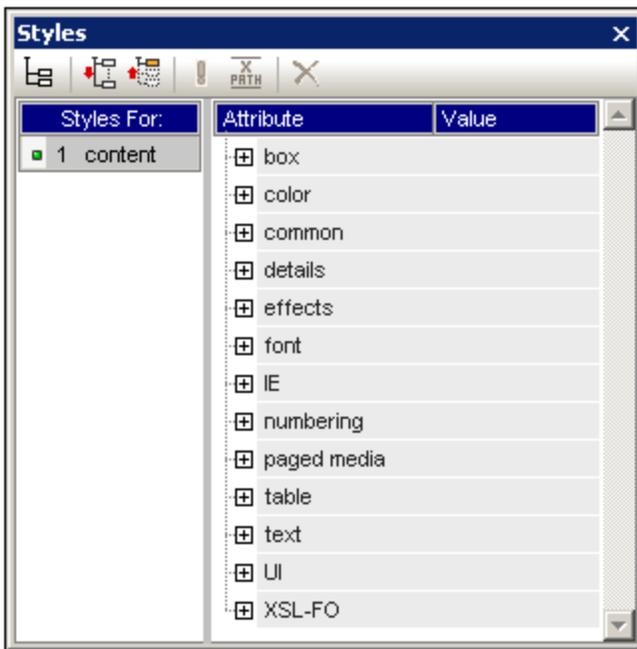
8.4.4 Setting Style Values

For the component-type selected in the *Styles-For* column, style properties are defined in the [Style Definitions pane of the Styles sidebar](#)⁴³ (*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.



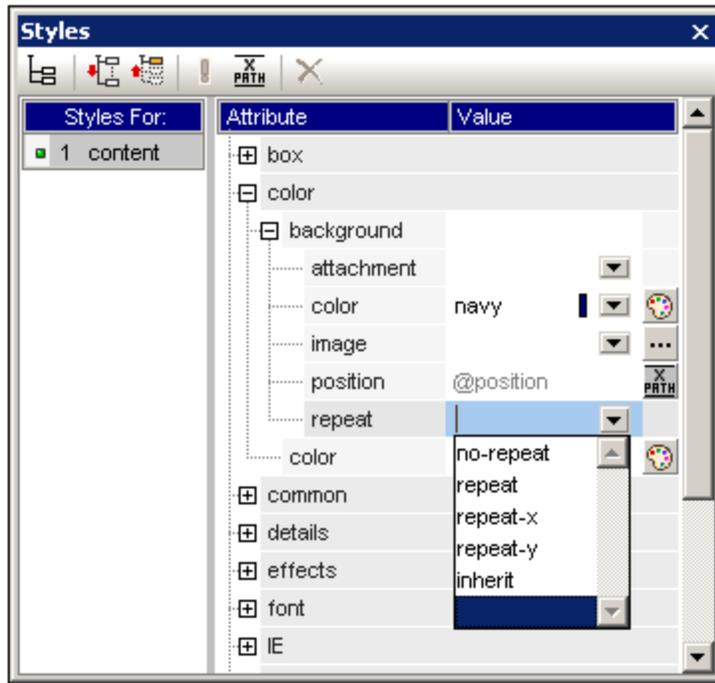
The display of properties can be modified using the [List Non-Empty](#) ⁴³ [Expand All](#) ⁴³ and [Collapse All](#) ⁴³ toolbar buttons ⁴³. Each group of style properties can be expanded to access style properties or sub-groups of style properties (see screenshot below).

Entering style values

Style 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](#) ³²⁹.

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](#) ³²⁸.

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

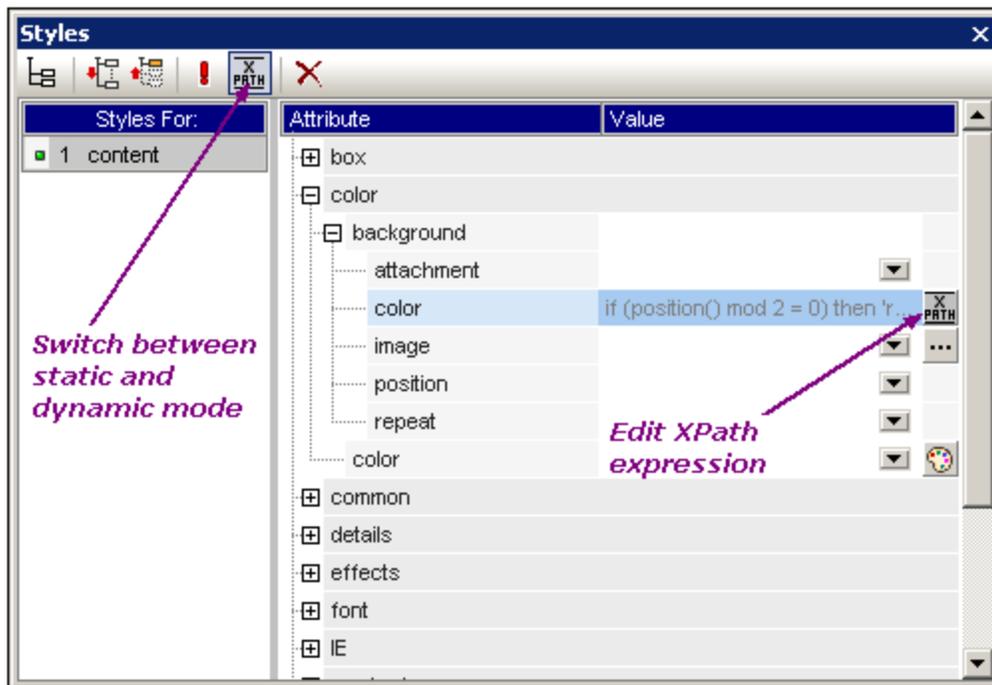
Static mode and dynamic (XPath) mode for property values

For those properties where [XPath expressions are enabled](#)³²⁹, 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](#)³⁹⁷ 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](#)⁴⁴⁷.) 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](#)³⁹⁷. 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](#)³⁹⁷ 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](#)³⁹⁷ 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](#) ²⁴⁰
- [The \(contents\) placeholder](#) ¹⁰³
- [Paragraph \(block\) design elements](#) ¹⁰⁵
- [Table cells](#) ¹¹⁸

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.
- 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:

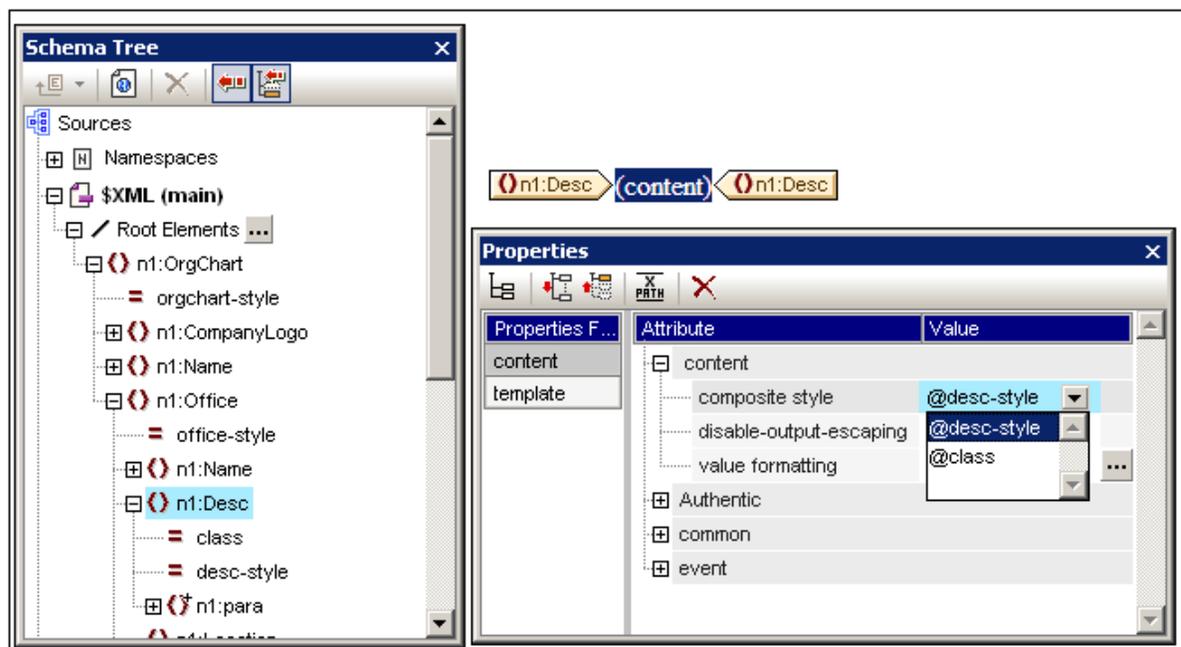
font-family	font-size	font-weight	font-style
color	background-color	text-align	text-decoration

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.

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.

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">
```

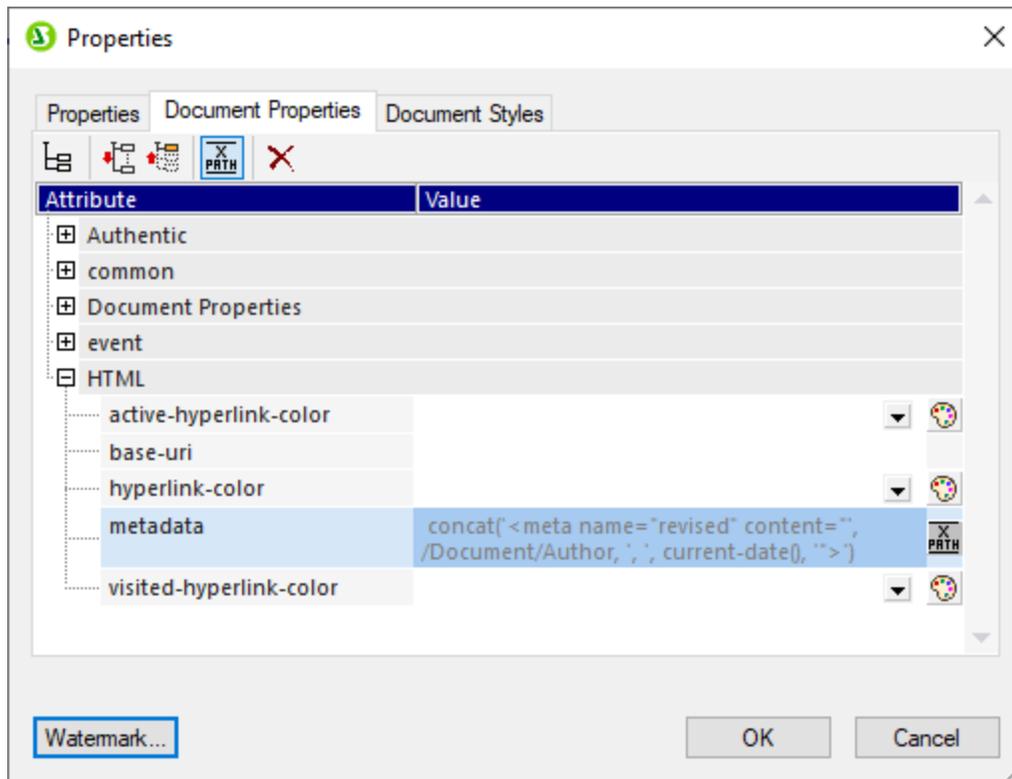
Note: Attribute value templates are XSLT constructs that allow the value of an attribute to be read as an XPath expression. They are delimited by curly braces and allow the value of the attribute to be assigned dynamically.

8.5 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 or in Properties View when the [Main Template in the Design Tree window](#)³⁸ is selected.

Via properties of the Initial Document Section

Click the *Edit Properties* hyperlink in the Initial Document Section title bar and then select the *Document Properties* tab.

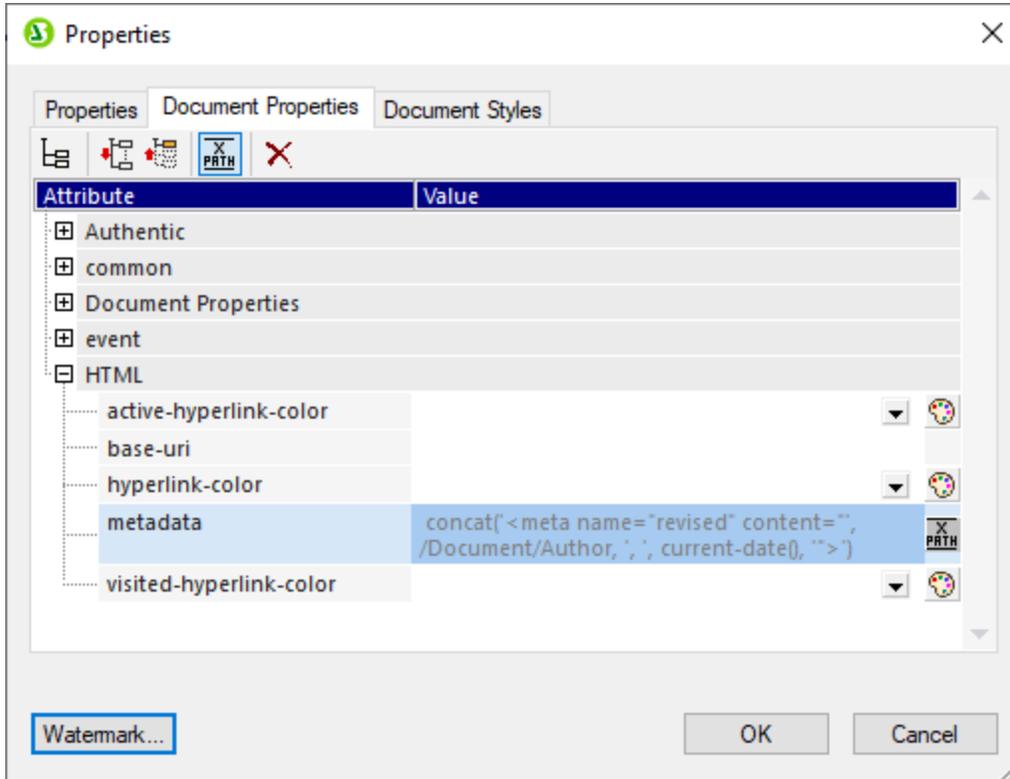


Here you can set various properties of the HTML output document that will be generated. Note the following points:

- 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, which are from the *Enterprise Edition of StyleVision and contain features specific to that edition*). 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](#) ³⁸ 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.

9 Additional Functionality

Additional to the [content editing](#)¹⁰², [structure](#)¹⁷², [advanced](#)²³⁹, and [presentation](#)³⁰⁵ 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.

- [Working with Dates](#)³⁵⁹. Dates can be manipulated and formatted as required.
- [Unparsed Entity URIs](#)³³⁸. 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](#)³⁶². 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](#)³⁶⁷. An HTML file can be imported into StyleVision and an XML, XSD, and SPS files can be created from it.
- [New from XSLT](#)³⁴⁰. 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](#)⁴⁴

9.1 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

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](#)³³⁸, 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](#)³³⁸. This is done by giving the names of the required unparsed entities.
3. In the SPS, unparsed entities can be used to target [images](#)¹⁴³ and [hyperlinks](#)³⁰⁰ by [correctly accessing the relevant dynamic node values as unparsed entities](#)³³⁸.

Declaring and referencing unparsed entities

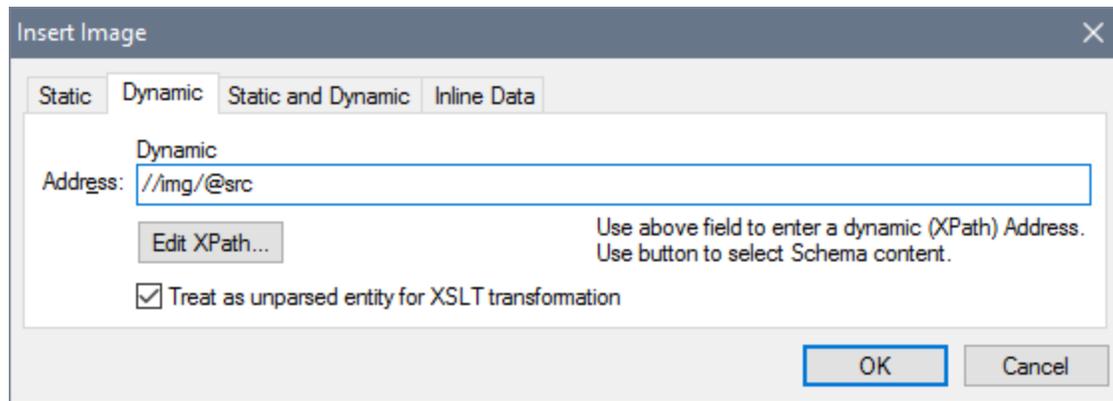
Given below is a cut-down listing of an XML document. It has an internal DTD subset which declares two unparsed entities, one with a `GIF` notation (indicating a GIF image) and the other with an `LNK` notation (indicating a hyperlink). The `img/@src` and `link/@href` nodes in the XML code reference the 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`.

9.2 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](#)²⁷.
5. You can now modify the SPS in the usual way. For example, you could drag in nodes from the [Schema Tree](#)³⁵, modify the styling and presentation or add additional styling, and use StyleVision functionality such as [Auto-Calculations](#)²⁴⁰ and [Conditional Templates](#)²⁴⁵.
6. You can save the SPS and use a [Working XML File](#)³² to preview [various output formats](#)²⁸. Subsequently you can [generate stylesheets and output files](#)⁴⁴⁰ using the [Save Generated Files](#)⁴⁴⁰ command.

Example

The example discussed below is located in the [\(My\) Documents folder](#)²³, C:\Documents and Settings\<<username>\My Documents\Altova\StyleVision2023\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>
</?xml>
```

```

        <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>

```

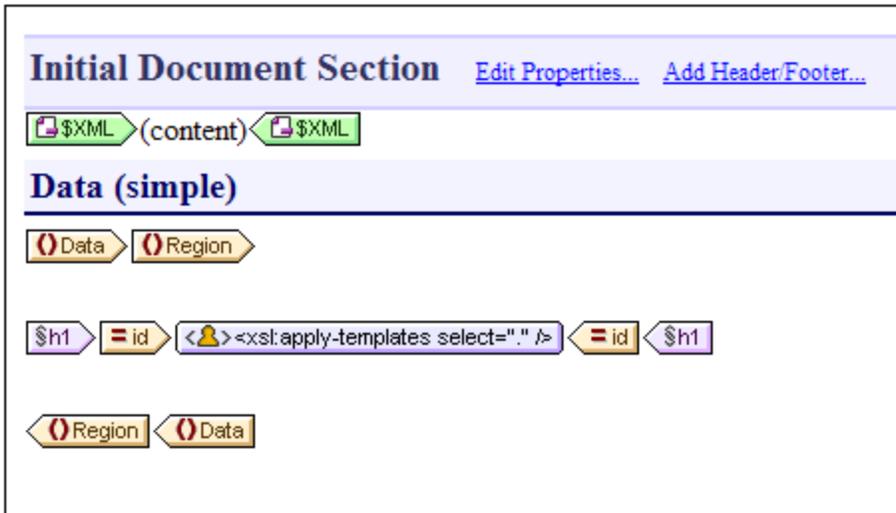
The XSLT file is as follows:

```

<?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/2005/2005/2005/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>

```

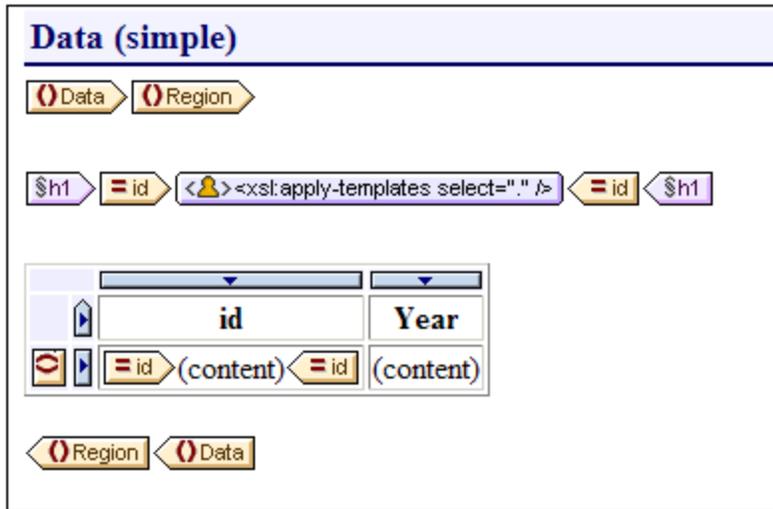
Follow the steps 1 to 4 listed above to obtain the SPS in Design View. The SPS will look something like this:



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.



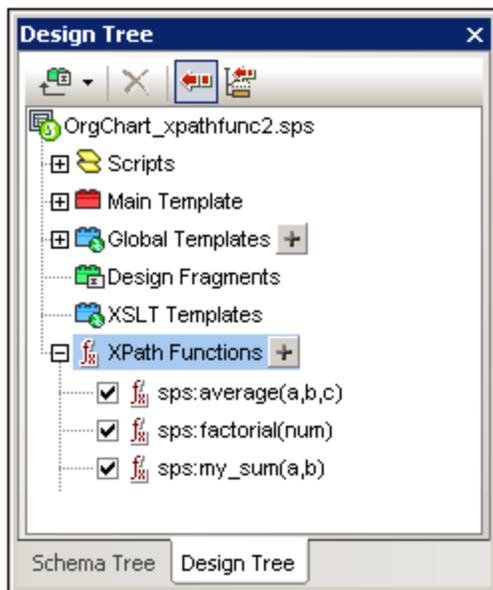
You can make additional changes in the content, structure, and presentation properties of the document, then preview the output and save files using the [Save Generated Files](#)⁴⁴⁰ command.

9.3 User-Defined XPath Functions

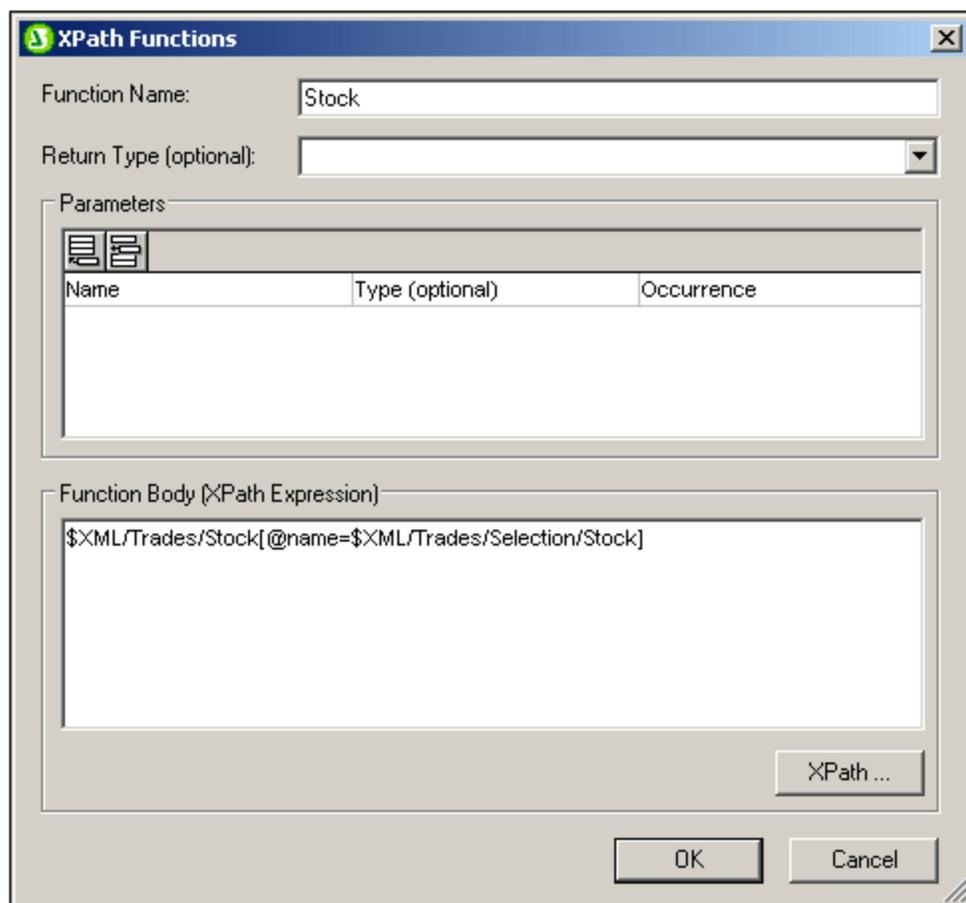
The SPS designer can define customized **XPath 2.0/3.0** functions. A user-defined XPath function can be re-used in any design component that accepts an XPath expression, for example, in Auto-Calculations, conditions, and combo boxes.

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.



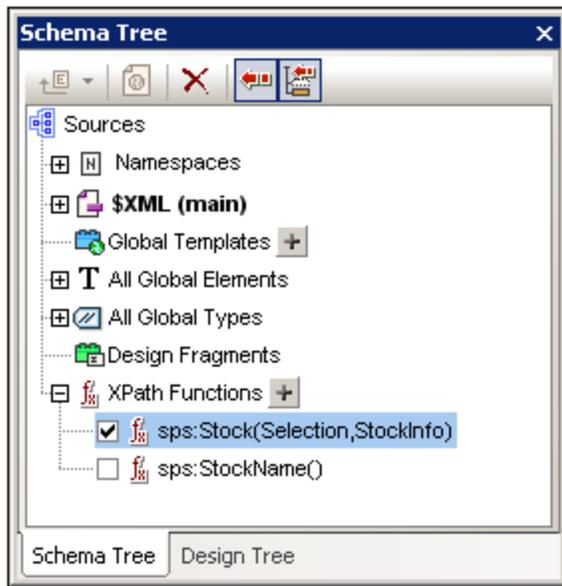
After a user-defined XPath function is created, it is available for use anywhere in the design.

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.

9.3.1 Defining an XPath Function

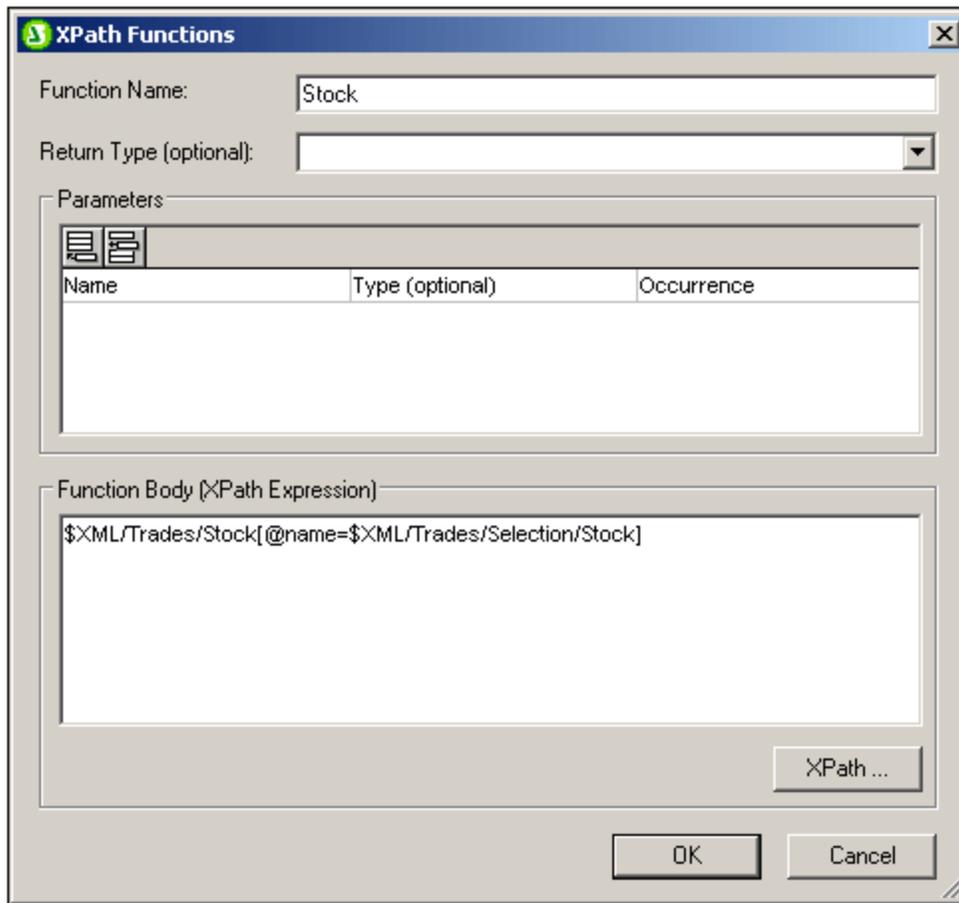
A user-defined XPath function requires: (i) a name (a text string), and (ii) a definition (an XPath expression).

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](#)³⁵² and [Parameters and Nodes](#)³⁵⁷, 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](#)³⁵⁷ below for an explanation.

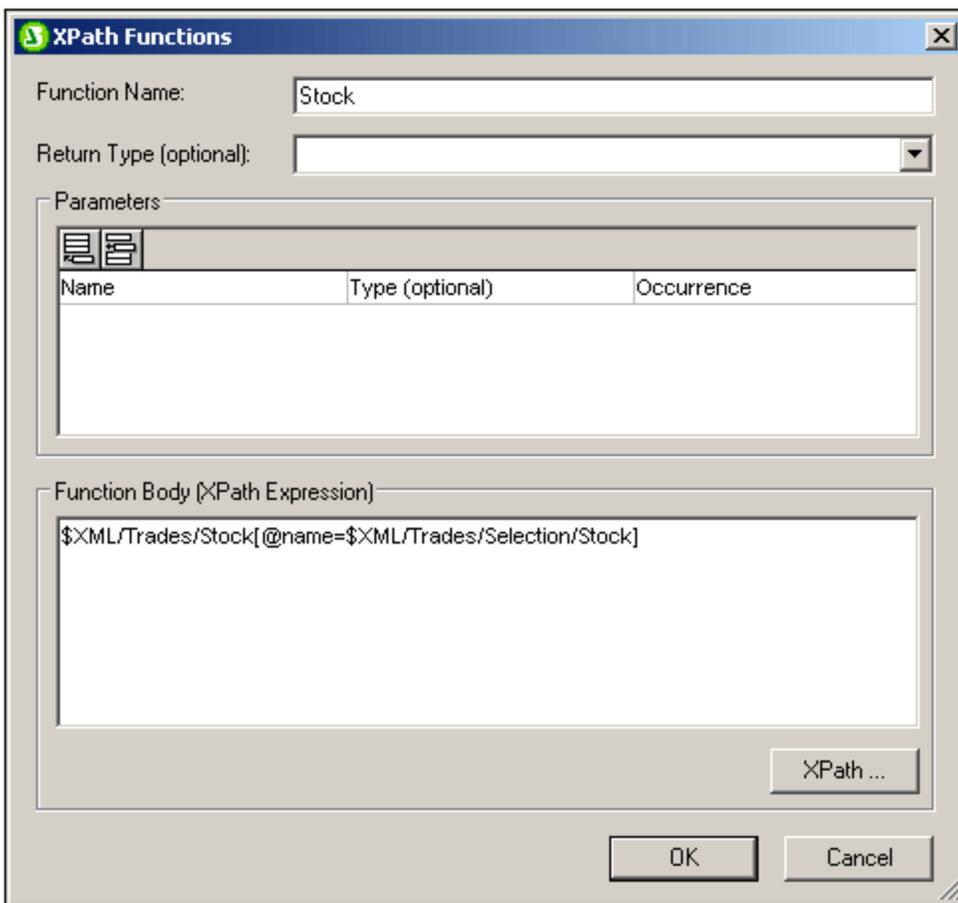
In the following cases, errors are returned:

- 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.

9.3.2 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.

9.3.3 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).

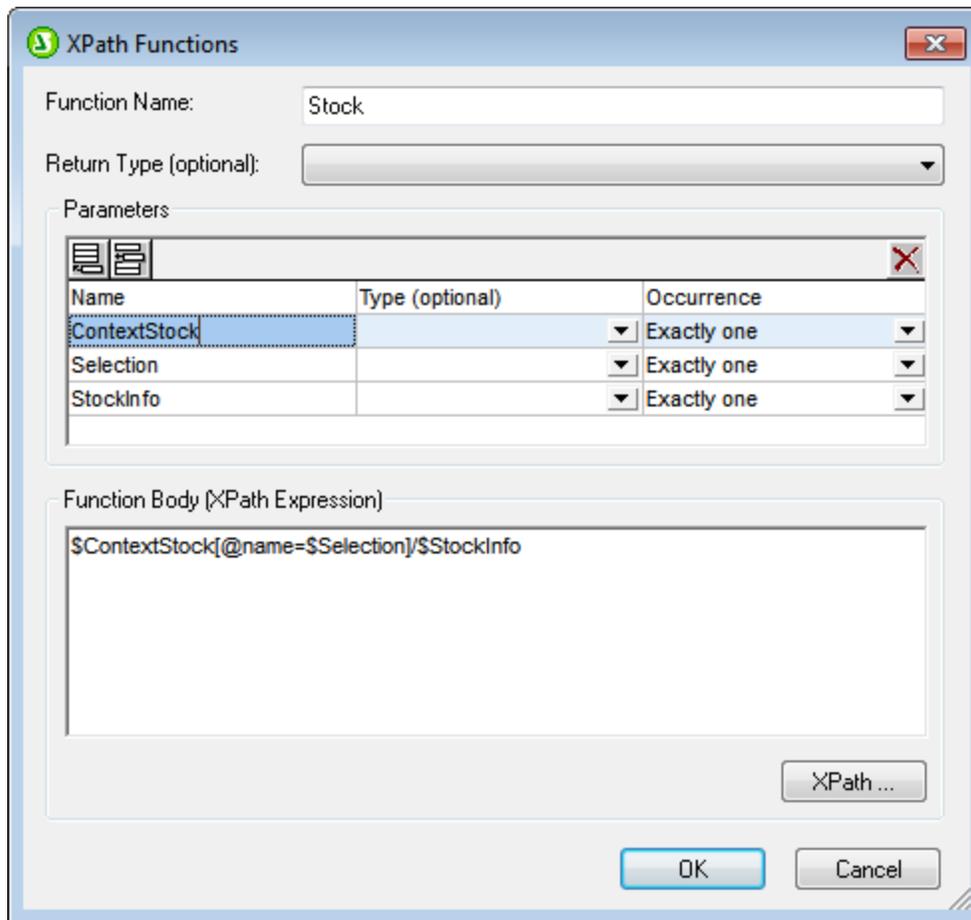
User-defined XPath function mechanism

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`.

9.3.3.1 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).*

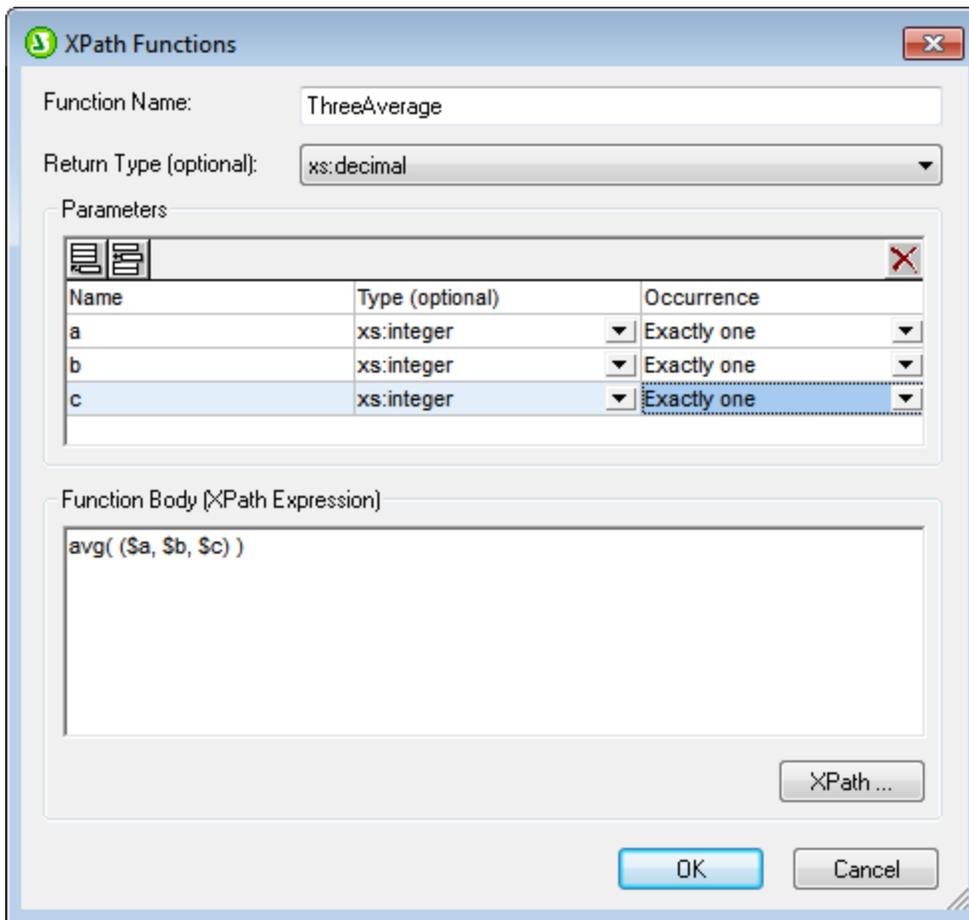
Parentheses are not necessary around a path (or locator) expression (example of a path expression: `//Person/@salary`), because a path expression can be read unambiguously as one parameter or one argument. It is in fact a one-sequence parameter/argument.

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 controls:

- Function Name:** Average
- Return Type (optional):** xs:decimal
- Parameters:** A table with three columns: Name, Type (optional), and Occurrence.

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:

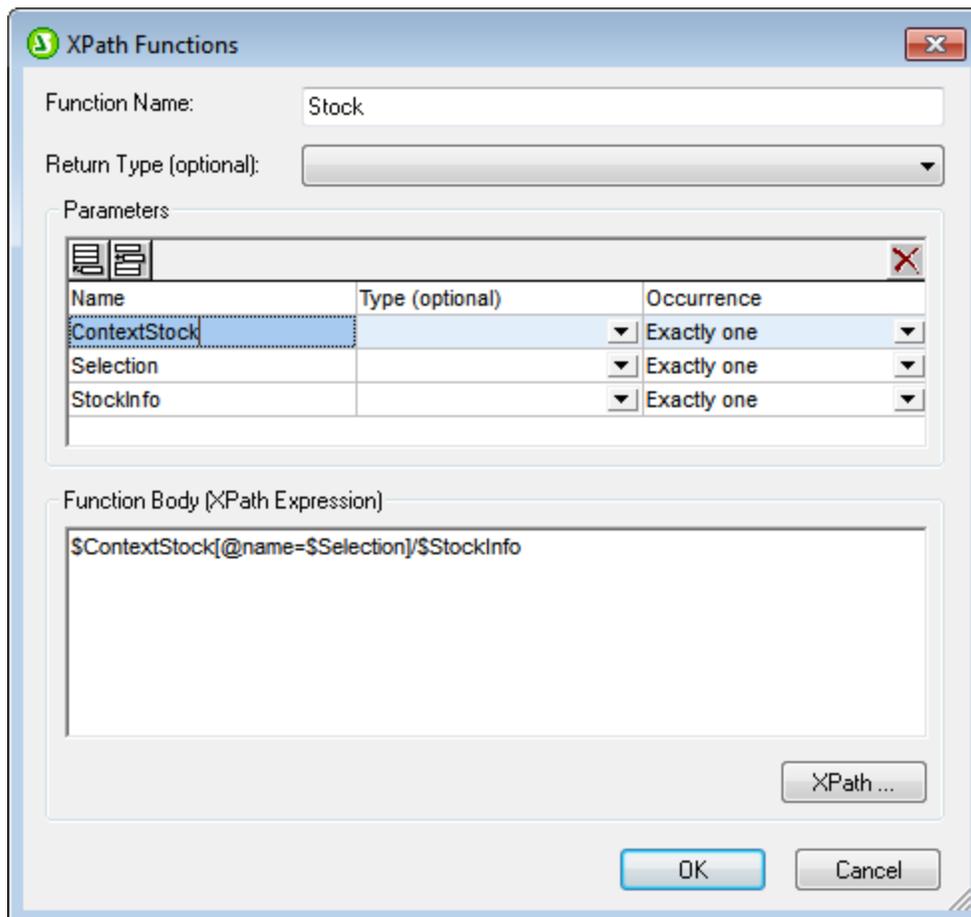
```
if ($num < 2) then 1 else $num * sps:Factorial($num - 1)
```

If this function were called `Factorial()`, then the factorial of, say 6, could be obtained by calling the function with: `sps:Factorial(6)`.

9.3.3.2 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.

9.4 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](#)³⁵⁹). StyleVision supports the `xs:date` or `xs:dateTime` datatypes by providing a wide range of [date formatting](#)³⁵⁹ possibilities via the [Input Formatting](#)³¹⁰ feature.

Note: Date and time data cannot be manipulated with XPath 1.0. However, with XPath 1.0 you can still use Input Formatting to provide [date formatting](#)³⁵⁹.

Date calculations with XPath 2.0

Data involving dates can be manipulated with XPath 2.0 expressions in [Auto-Calculations](#)²⁴⁰. 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.

9.4.1 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.

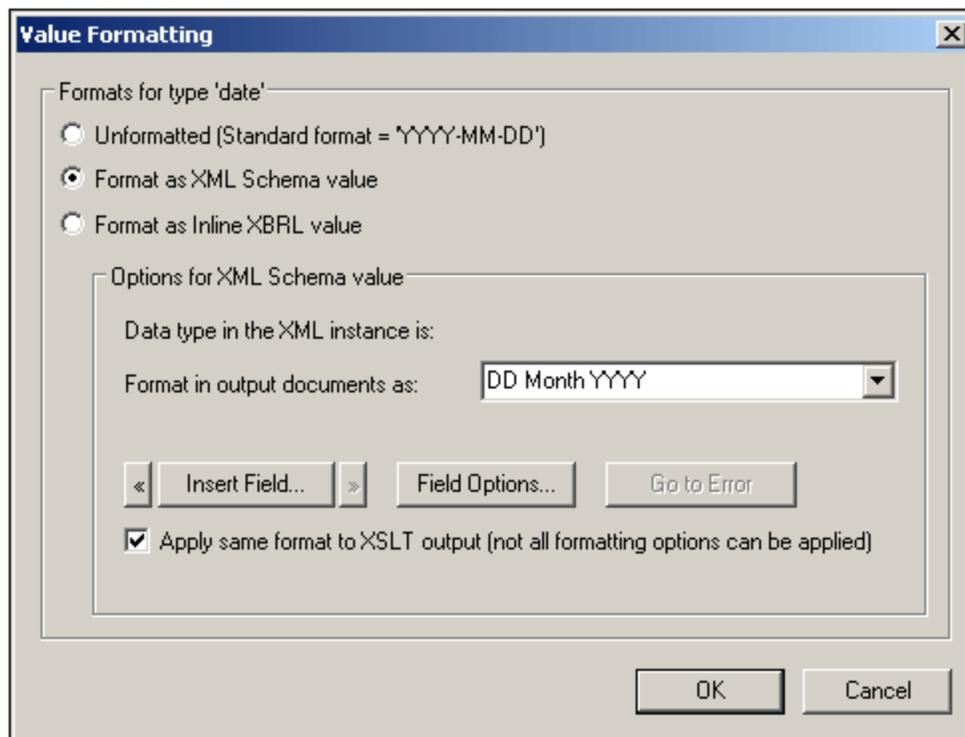
Value Formatting for dates can be used to define custom formats for nodes and Auto-Calculations of the following datatypes:

- 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*).



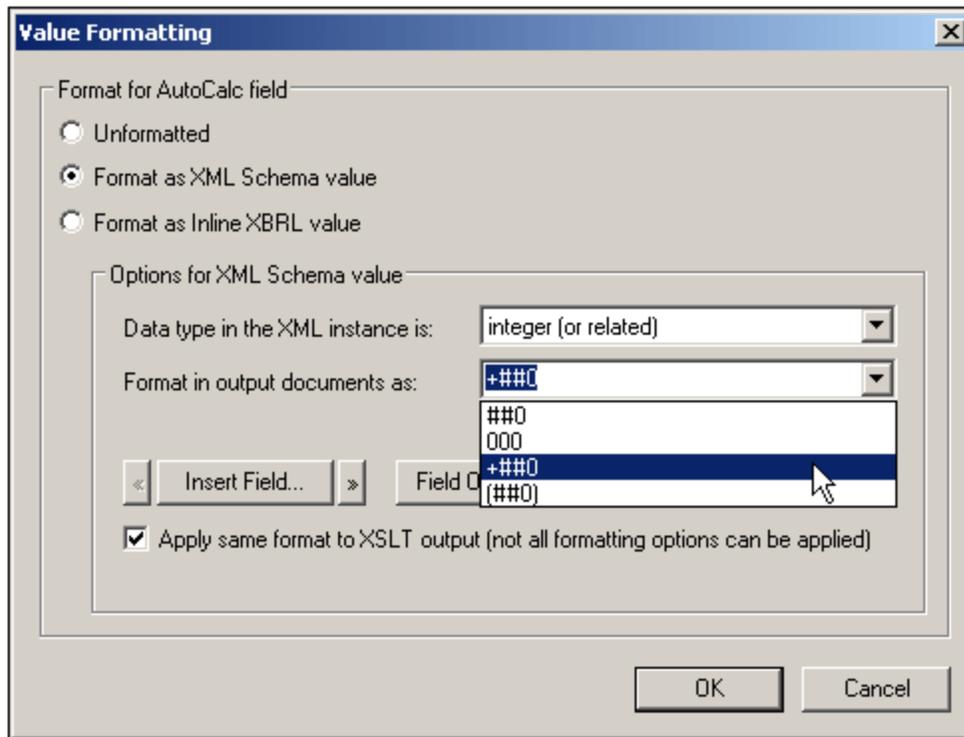
By default, the *Unformatted* radio button (the standard lexical format for the node's datatype) is selected.

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](#) ³¹³ 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](#)³¹³ 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 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.

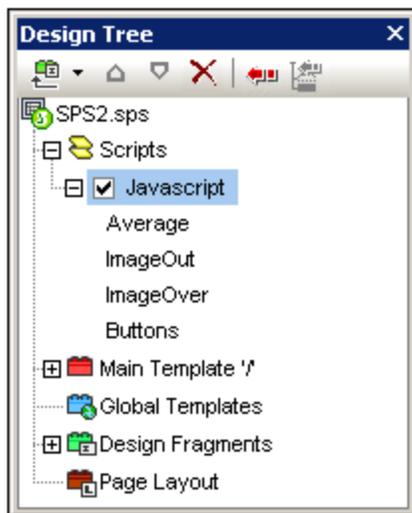
9.5 Using Scripts

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](#)³⁶³.

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. How to assign a JavaScript function to an event handler is described in the section [Assigning Function to Event Handlers](#)³⁶⁴.

Scripts in modular SPSs

When an [SPS module is added to another SPS module](#)²⁰¹, 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](#)²⁰¹.

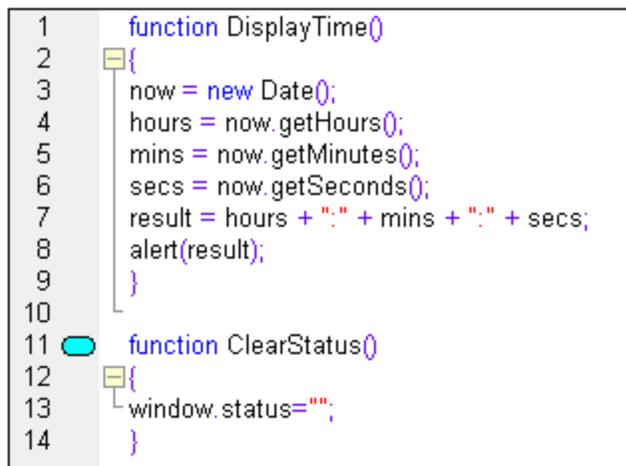
9.5.1 Defining JavaScript Functions

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*).



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)
}

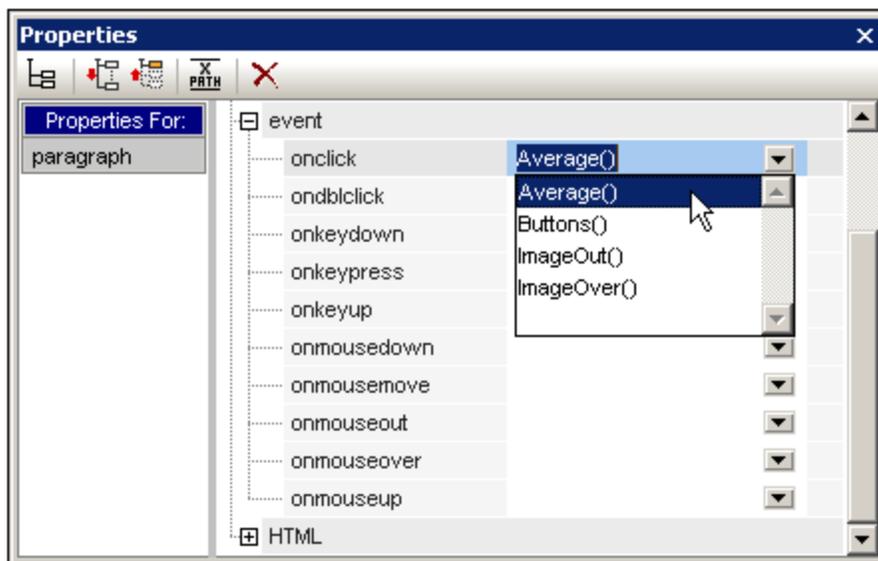
function ClearStatus()
{
    window.status="";
}
-->

</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](#)³⁶⁴ for details.

9.5.2 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.

9.5.3 External JavaScript Files

An SPS can access external JavaScript files in two ways:

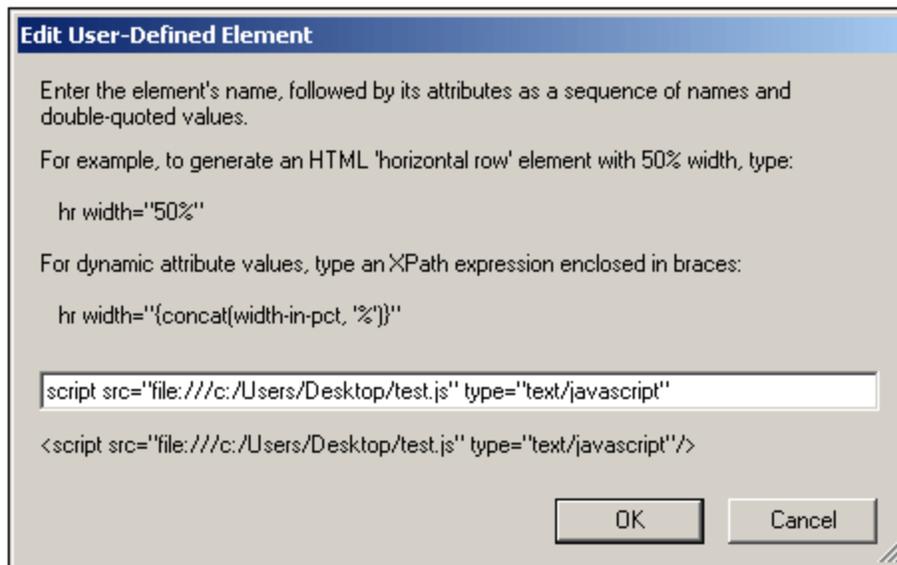
1. [By creating a User-Defined Element or User-Defined XML Block](#)³⁶⁵. These design objects can contain a `SCRIPT` element that accesses the external JavaScript file. Note that location of the User-Defined Element or User-Defined XML Block is within the `BODY` element of the design (and therefore within the `BODY` element of the HTML output, not within the `HEAD` element).
2. [By adding a script in the Javascript Editor](#)³⁶⁶ 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](#)¹¹⁵ and [User-Defined XML Blocks](#)¹¹⁶. 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](#)¹¹⁵ 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](#)¹¹⁵.



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](#)¹¹⁶ 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](#)¹¹⁶ is inserted instead of a [User-Defined Element](#)¹¹⁵, 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](#)³⁶³ 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.

9.6 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:

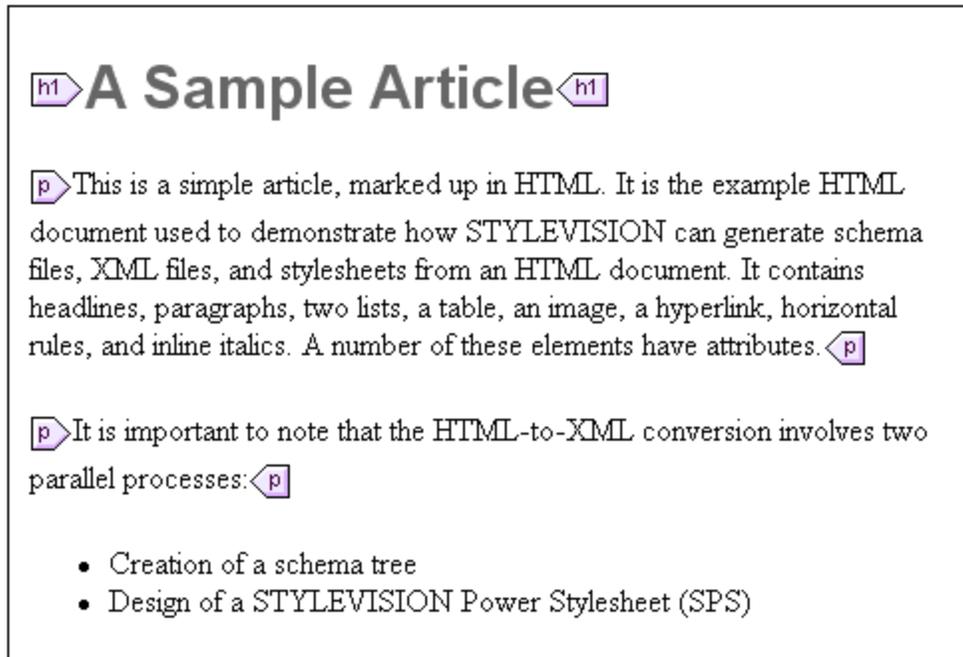
1. [Creating New SPS via HTML Import](#)³⁶⁷. When an HTML file is imported into StyleVision, a new SPS document is created. The HTML document is displayed in Design View with HTML markup tags. A user-defined XML Schema with a document element called `UserRoot` is created in the Schema Tree window. This 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](#)³⁶⁹. 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](#)³⁷³ 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.

9.6.1 Creating New SPS via HTML Import

To create a new SPS file from an HTML document, do the following:

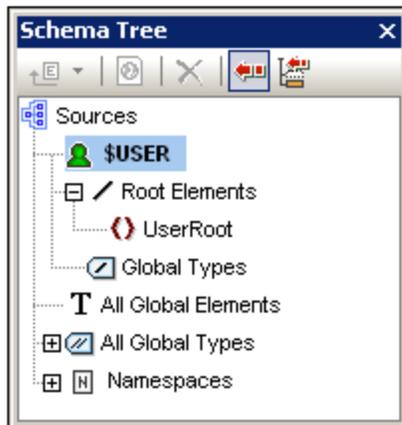
1. Select the menu command **File | New | New from HTML File**.
2. In the Open dialog that pops up, browse for the HTML file you wish to import. Select it and click **Open**.
3. You will be asked whether relative paths should be converted to absolute paths. Make your choice by clicking either **Yes** 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).

9.6.2 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](#) ³⁷¹.

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

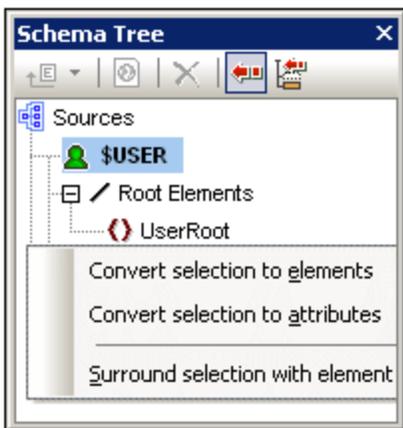
On dragging the selection over the desired schema tree node, the mouse pointer will be changed to one of the following symbols.

- 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](#) ³⁷¹.

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.

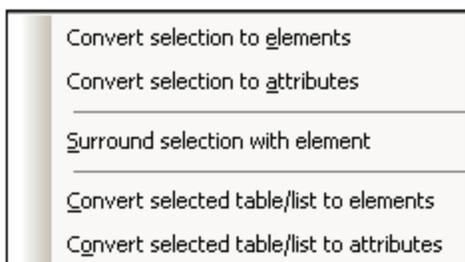
9.6.3 Creating Tables and Lists as Elements/Attributes

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.

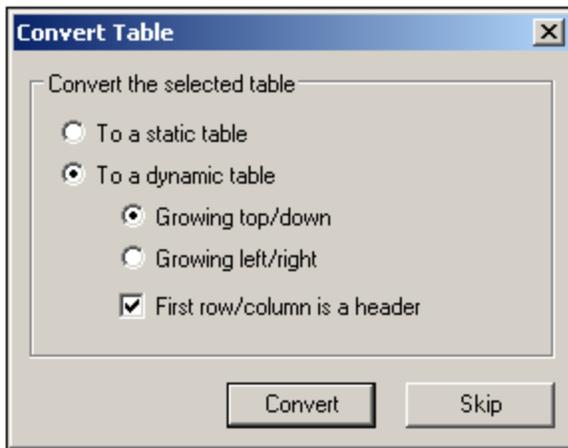
Converting a table to elements/attributes

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.



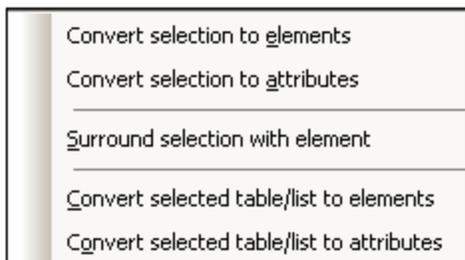
If the **static table** option is selected, then for each cell in the table, a schema node is created. In the design, each node is inserted with the `(content)` placeholder. The data in the table cells is copied to the temporary internal XML document (and to the generated XML document). The **dynamic 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.

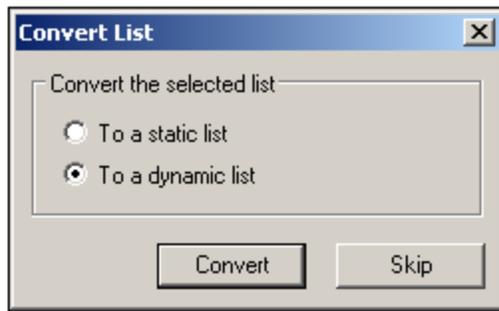
Converting a list to elements/attributes

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.

9.6.4 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 output.
- HTML output.

9.7 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.

9.7.1 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.

9.8 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.

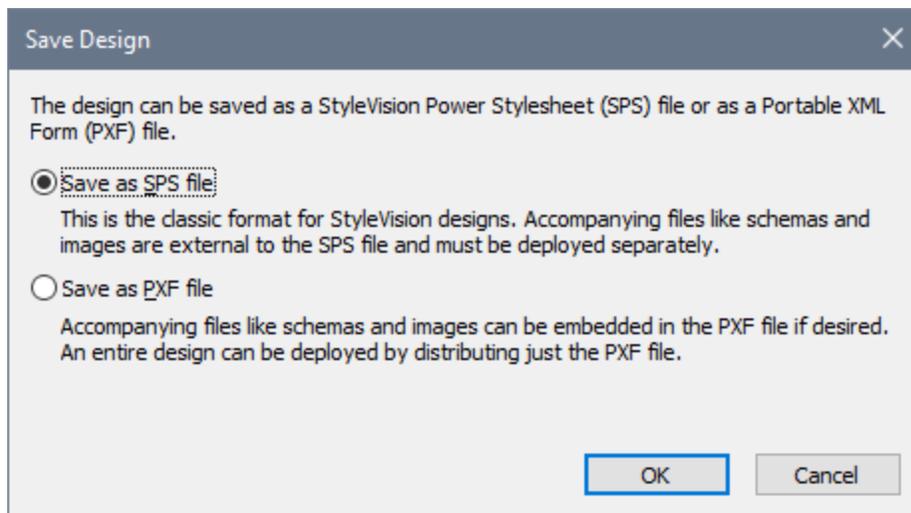
This section describing the use of PXF files is organized in two parts:

- [Creating a PXF file](#) ³⁷⁷
- [Editing a PXF File](#) ³⁸⁰
- [Deploying a PXF file](#) ³⁸¹

Note: PXF files can be created only from SPSs designed with XSLT 2.0 or 3.0.

9.8.1 Creating a PXF File

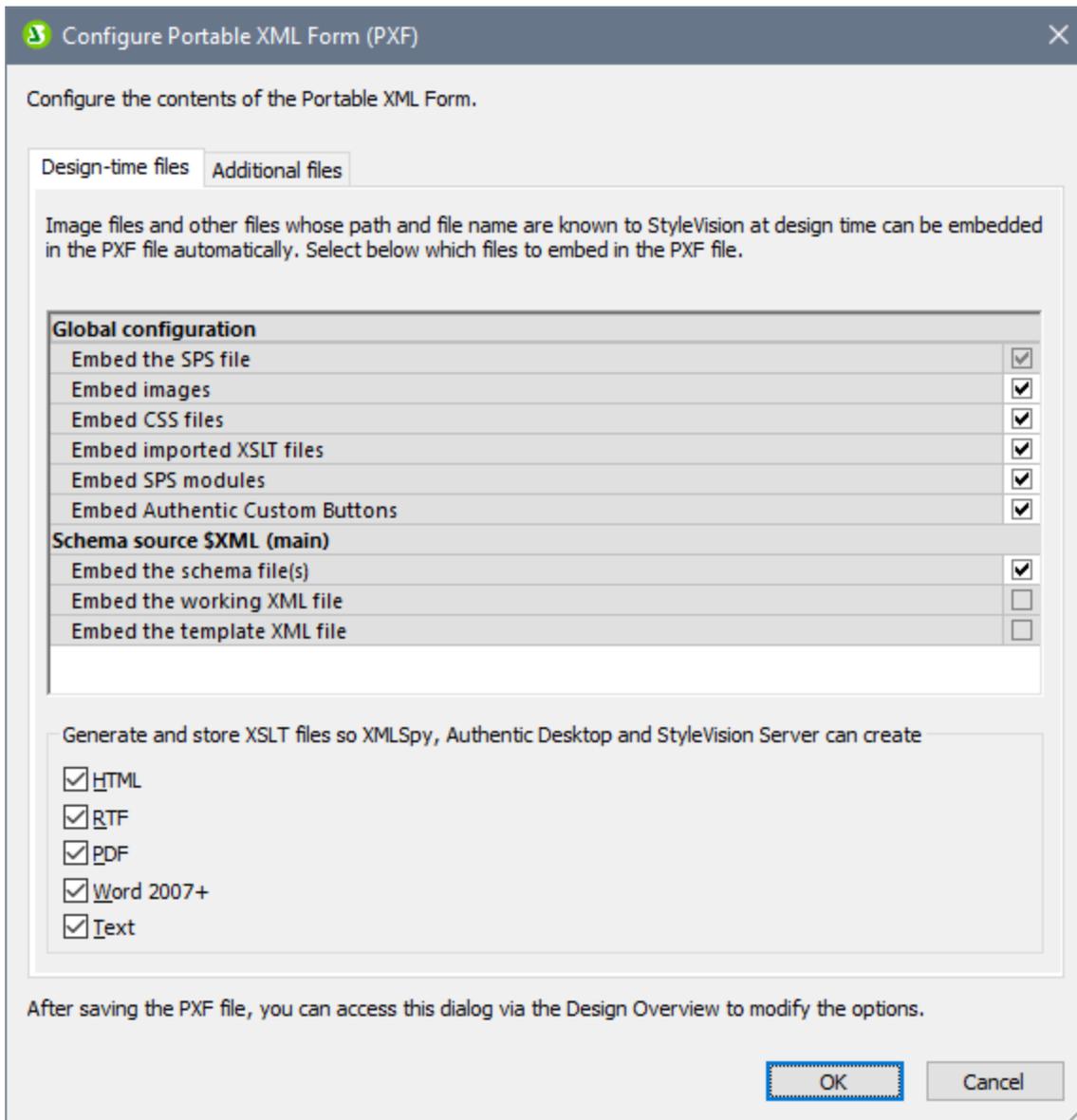
To create a PXF file that will contain an SPS design plus related files, open the SPS design in StyleVision and select the command **File | Save As**. This pops up the Save Design dialog (*screenshot below*).



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](#) ⁴³⁹.

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](#) ⁴³⁴ [command](#) ⁴³⁴—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.



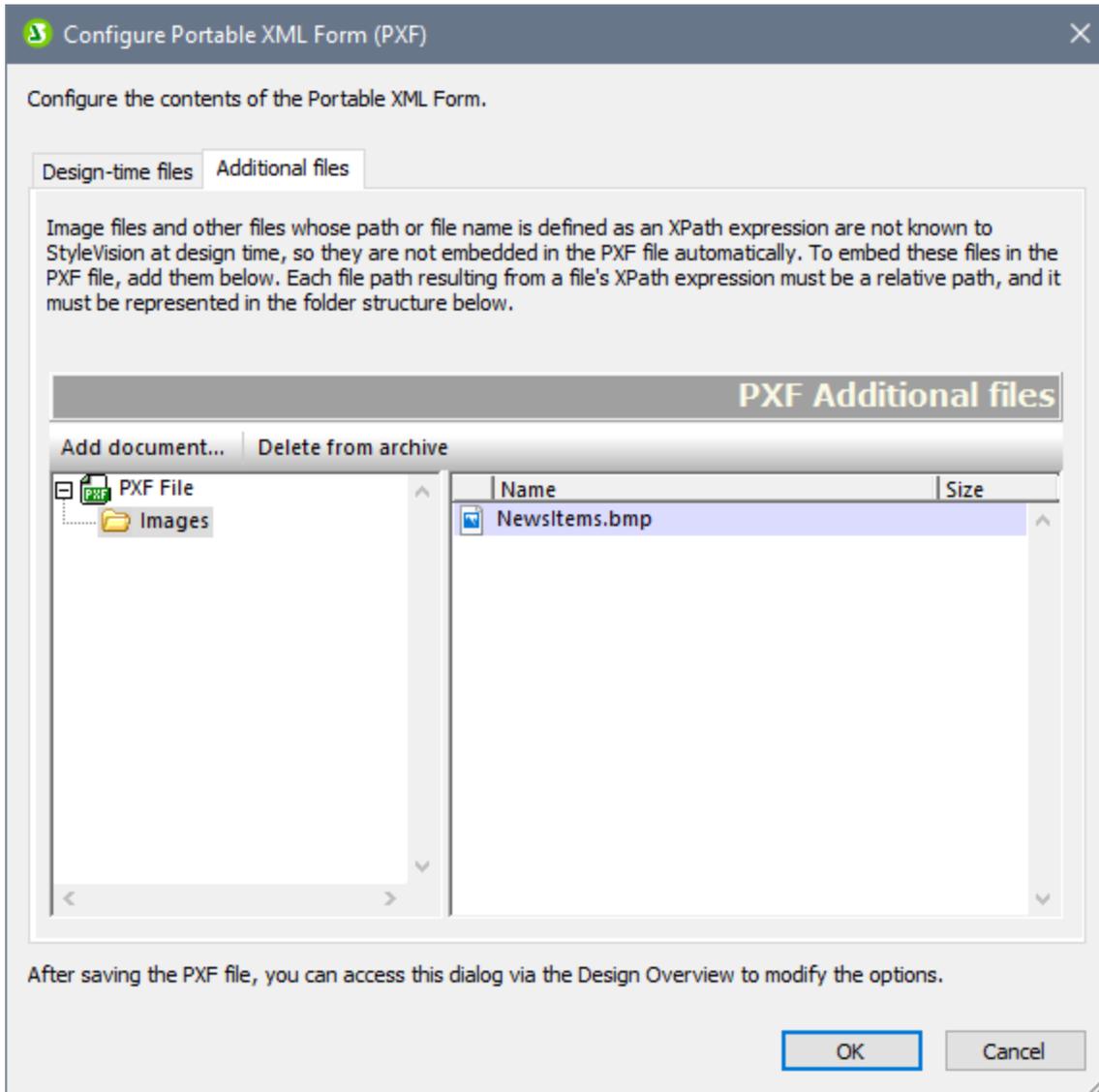
In the Global Configuration pane of the *Design-time Files* tab, you can select/deselect the design-related source files to be embedded/omitted. You can additionally choose to embed XSLT files generated from the design. In the XSLT files pane, select the output formats for which you wish to generate 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.

9.8.2 Editing a PXF File

A PXF file can be opened in StyleVision via the [File | Open](#) ⁴²⁹ 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*).

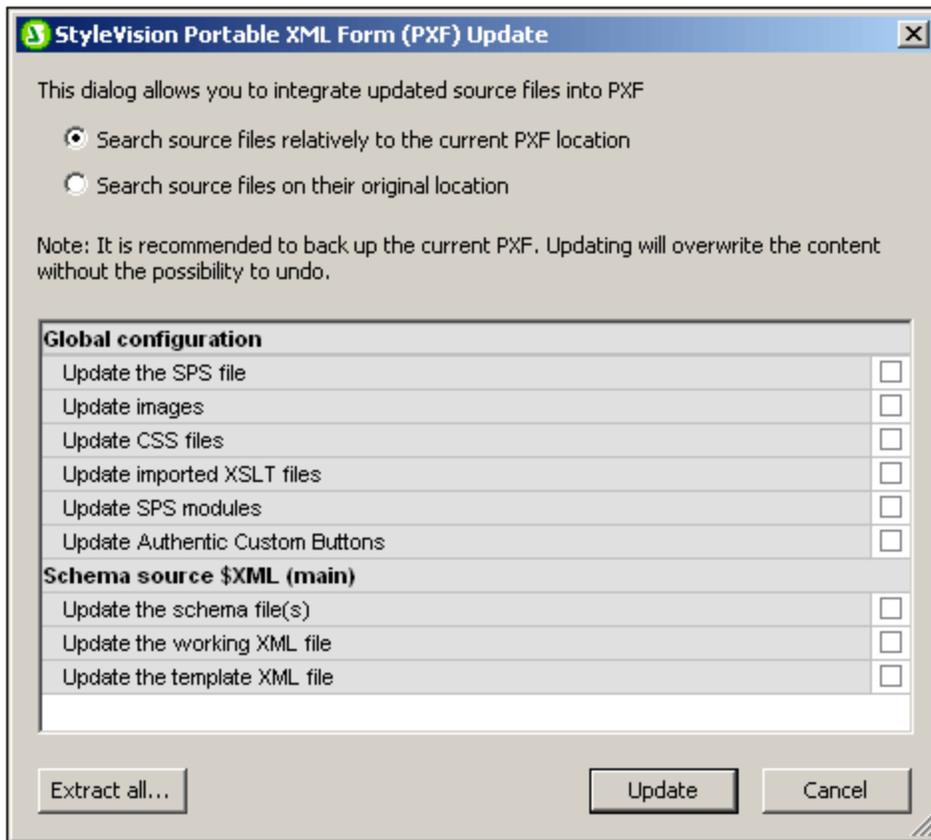


Configure embedded files

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](#) ³⁷⁷.

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.

9.8.3 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](#)¹⁸. 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 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](#)⁴⁴⁰ command:

- XSLT files for HTML output.
- HTML 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).

The processes to generate the final HTML 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](#)⁴⁴⁰ 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](#)³⁸⁷, 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](#)³⁸⁷.
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](#)³⁸⁹.

10.1 Command Line Interface

StyleVision functionality can be called from the command line in two ways:

- By calling the [StyleVision executable](#)³⁸⁴. This provides a access to StyleVision's XSLT-file-generation functionality. The XSLT files are generated from the SPS file.
- By using [StyleVision Server](#)³⁸⁵ 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](#)³⁸⁵, 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](#)³⁹¹.

10.1.1 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, RTF, PDF, and Word 2007+ output-related options are available in the Enterprise edition, or the Enterprise and Professional editions only; these options are indicated with the words *Enterprise edition* or *Enterprise and Professional editions* 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 (<i>Enterprise and Professional editions</i>)
<code>-OutXSLText=<file></code>	Writes XSLT-for-Text to the specified file (<i>Enterprise and Professional editions</i>)
<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>)

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.

10.1.2 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](#)³⁷⁷ as its two arguments, and the desired output formats as its parameters. The XSLT stylesheets for the transformation are obtained from the [PXF file](#)³⁷⁷ 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](#)³⁸⁹ 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

10.2 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**⁴⁴⁰ 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 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](#)³⁹¹ 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](#)³⁹¹.

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](#).

10.2.1 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](#)⁴⁴⁰ 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
```

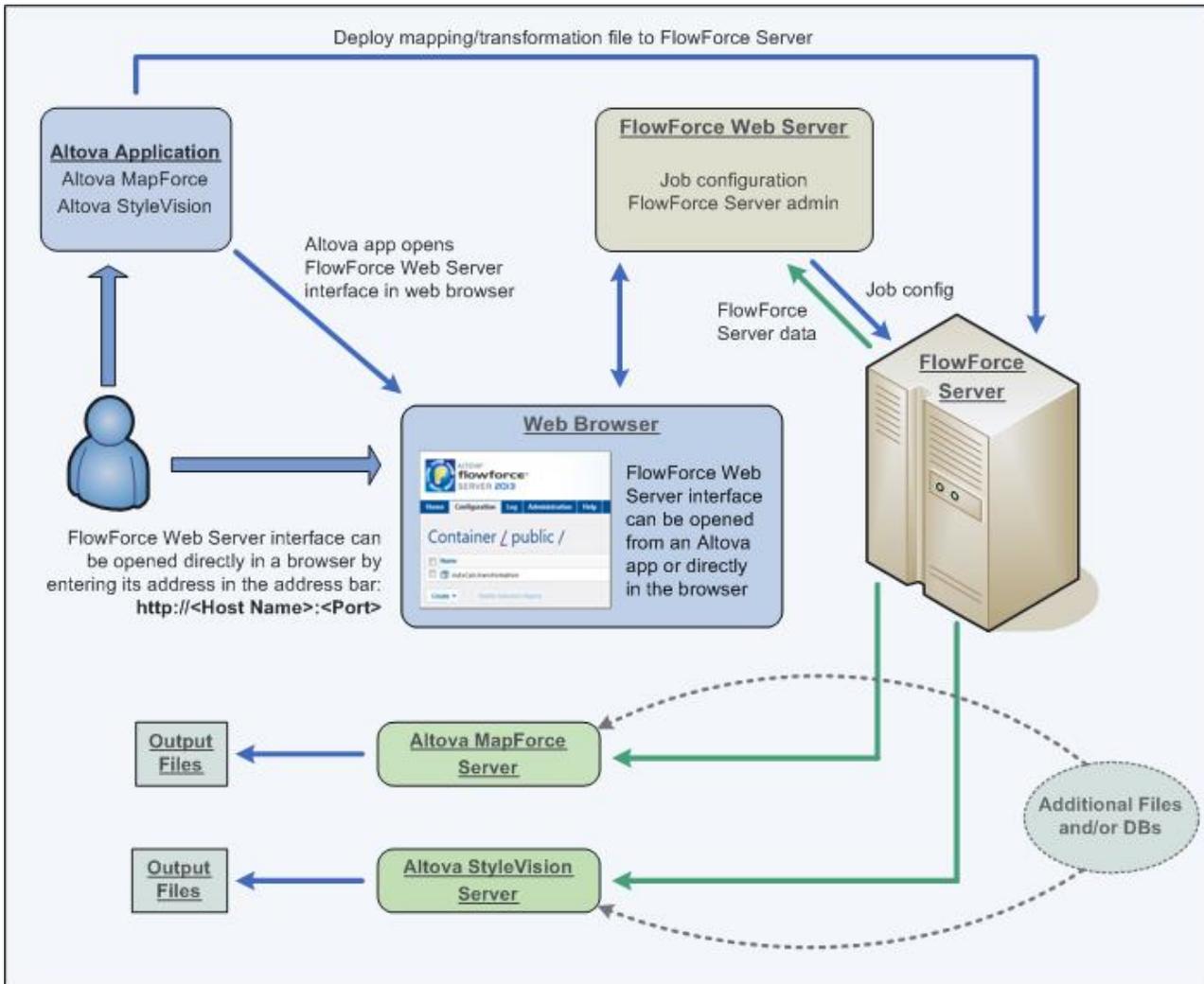
The first command calls RaptorXML and produces `test.fo` as output. The second command passes `test.fo` to the FOP processor, which generates the PDF file `test.pdf`. For more information about batch processing and how batch files can be used to automate processes, see the following section: [How to Automate Processing](#)³⁹¹.

10.3 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](#)³⁷⁷ is [deployed to FlowForce Server](#)⁴⁴¹ (with the [File | Deploy to FlowForce](#)⁴⁴¹ 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](#).

10.4 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.

11 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](#)³⁹³.
- A description of the [Edit XPath Expression dialog](#)³⁹⁷.
- A description of all the [toolbars with their icons](#)⁴¹⁴, 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](#)¹⁰²
- [SPS File Structure](#)¹⁷²
- [SPS File Advanced Features](#)²³⁹
- [SPS File Presentation](#)³⁰⁵
- [SPS File Additional Functionality](#)³³⁷

See also

- [User Interface](#)²⁵
- [Quick Start Tutorial](#)⁵⁰

11.1 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](#) ³⁹³
- XML document content
- Data-entry devices
- Predefined formats
- XPath objects
- URI objects

Each of these component types can:

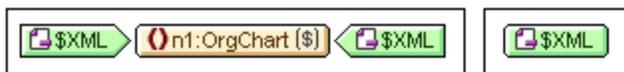
- be moved using drag and drop;
- be cut, copied, pasted, and deleted using (i) the commands in the [Edit menu](#) ⁴⁴⁷, or (ii) the standard Windows shortcuts for these commands;
- have formatting applied to it;
- have a context menu pop up when right-clicked.

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.

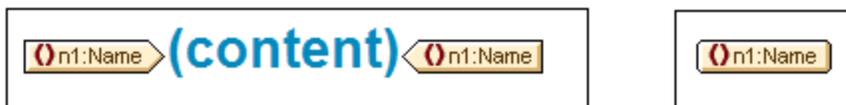
The following types of node are represented:

- **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](#)²¹⁵.

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**



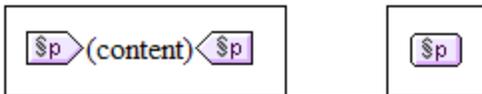
- **Radio buttons**



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](#)¹⁴⁸.

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](#)³⁰⁶.

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](#) ²⁴⁰.

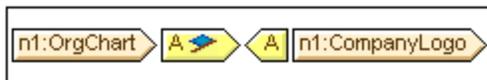
URI objects

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](#) ⁴¹⁴
- [Design sidebars](#) ³⁰
- [Content Editing Procedures](#) ¹⁰²

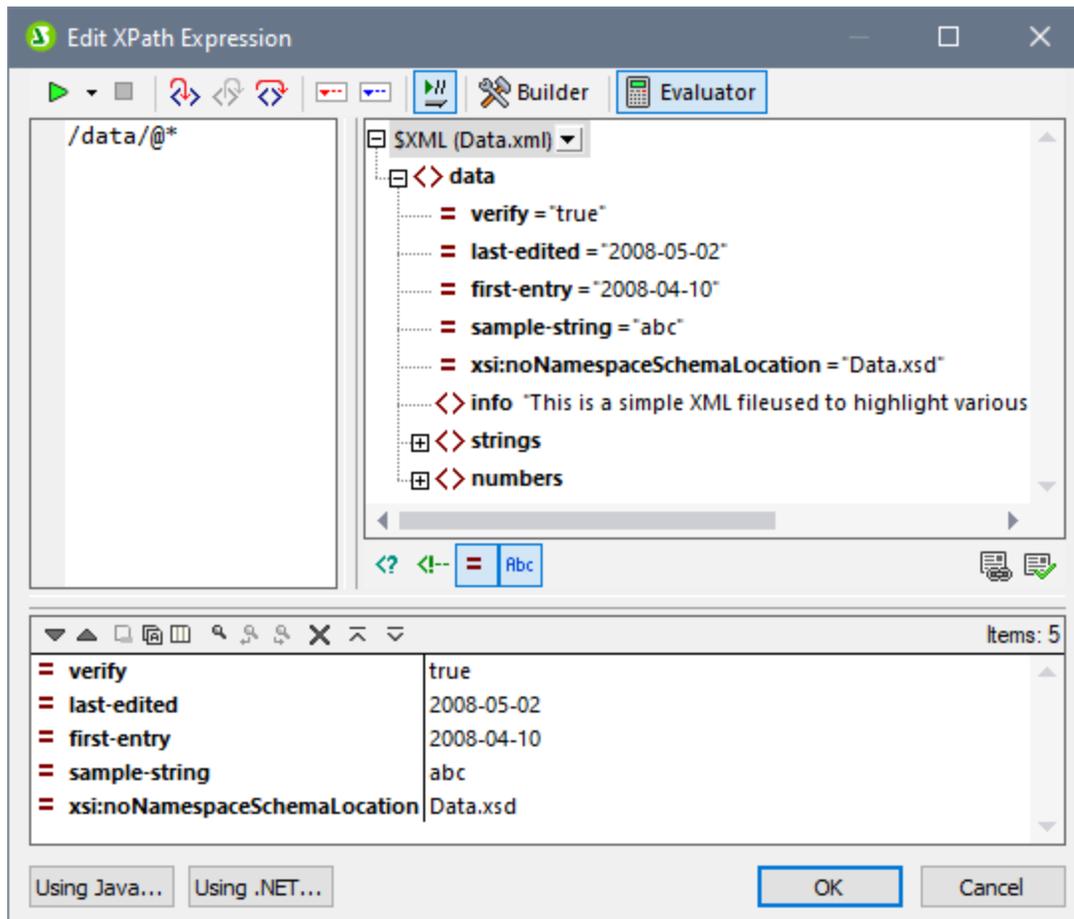
11.2 Edit XPath Expression Dialog

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](#)²⁴⁵ or the values of [Styles](#)⁴³ and [Properties](#)⁴⁴.

The dialog automatically supports the **XPath version** that corresponds to the [XSLT version](#)⁹² 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](#)⁹².

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.

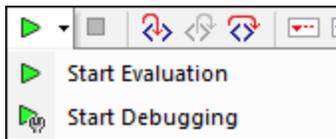


Evaluation Mode and Debug Mode

The Edit XPath Expression dialog has two modes:

- [Evaluation Mode](#)³⁹⁸, 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](#)⁴⁰¹, 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.

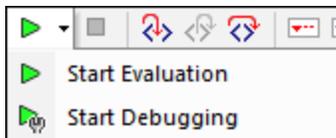
XPath/XQuery Expression Builder

In both modes, the [Expression Builder](#)⁴⁰⁹ can be used to help you construct syntactically correct expressions.

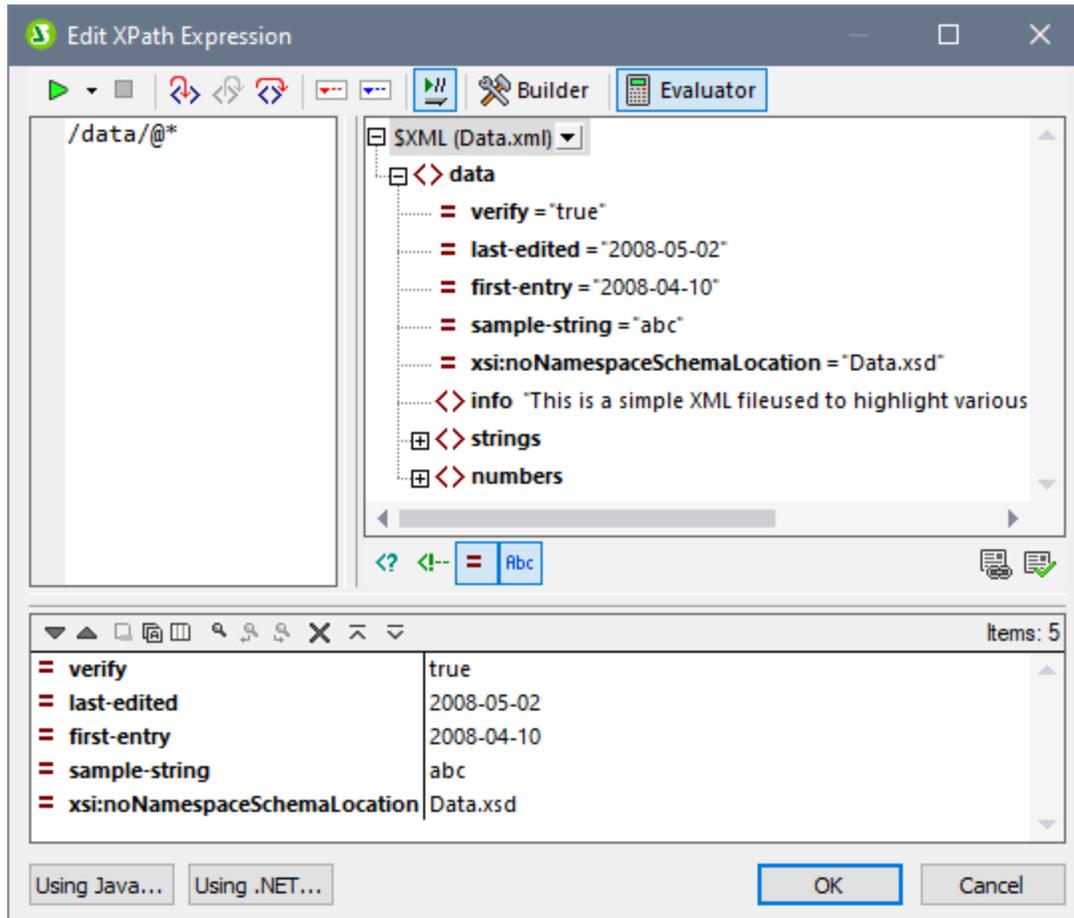
Switch [Expression Builder](#)⁴⁰⁹ on/off with the **Builder Mode** button of the main toolbar .

11.2.1 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:

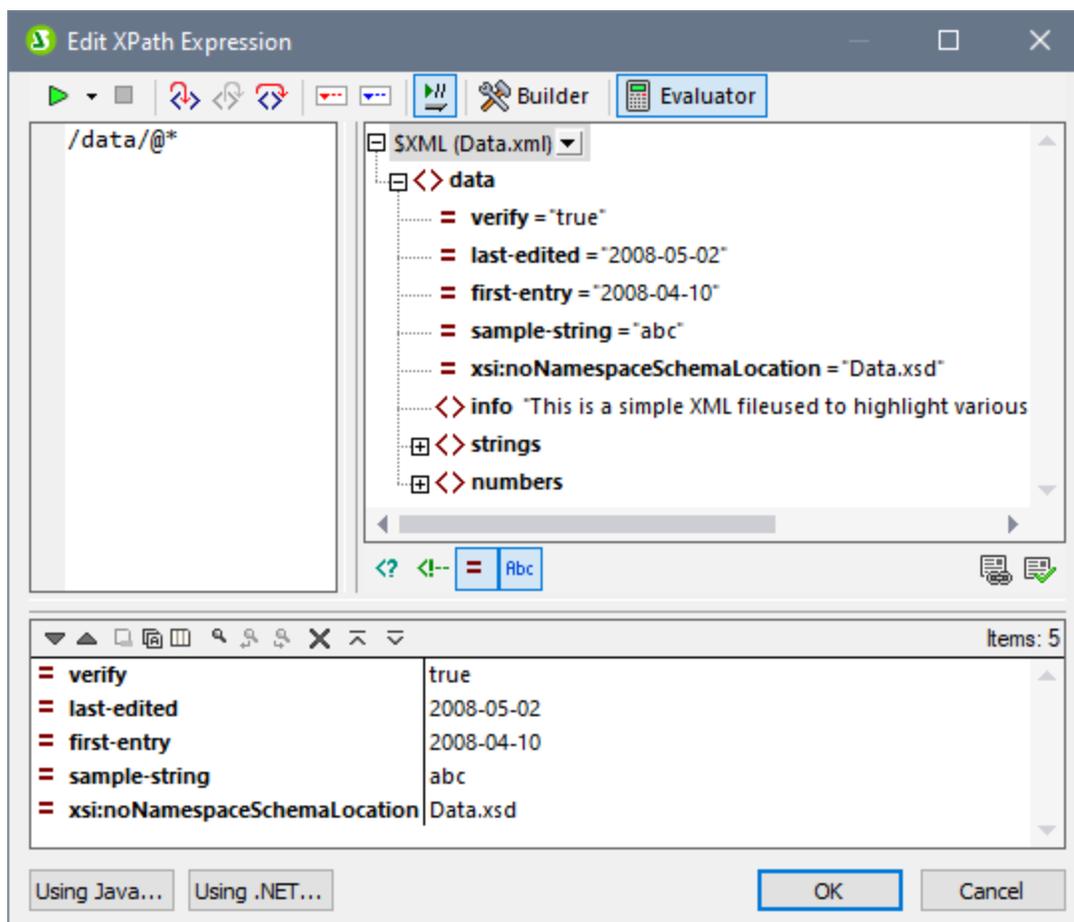
- In order for an expression to be evaluated against an XML file, that file must be assigned as the [Working XML file of one of the sources of the SPS](#) ⁹⁰.
- Results can be displayed even as you type the expression (select the **Evaluate on Typing** icon in the toolbar), or they can be displayed when 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](#)⁵³² 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).

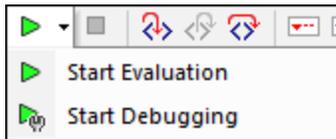
Toolbar of the Results pane

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>Clear</i>	Clears the result list
<i>Collapse multi-line results to a single line</i>	If the value column of a result item contains multi-line text (text that includes newline character/s), you can toggle between a multi-line and single-line display
<i>Show complete result</i>	Shows the entire content of nodes as the value of the node

11.2.2 Debugger

The Debugger enables you to debug an XPath expression in the context of a [Working XML File](#)⁹⁰. 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).

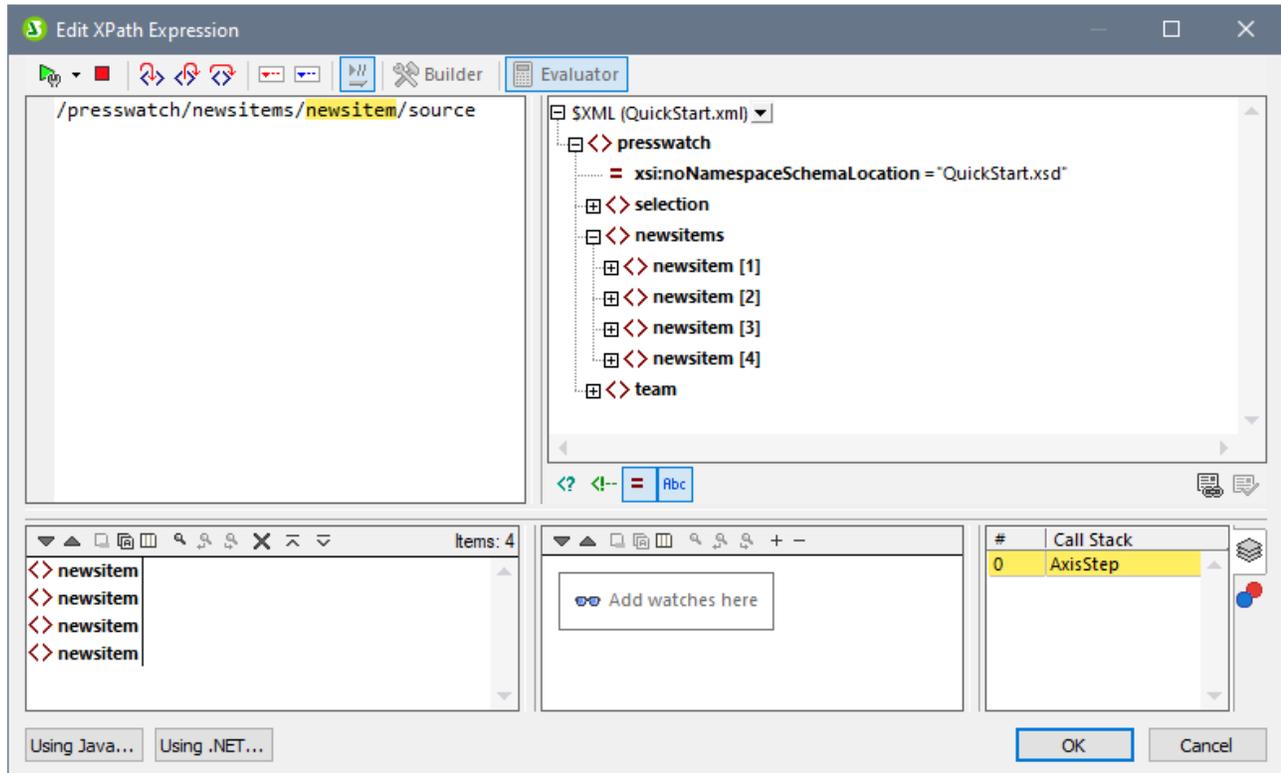
☐ Buttons for setting up Debug Mode

	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.

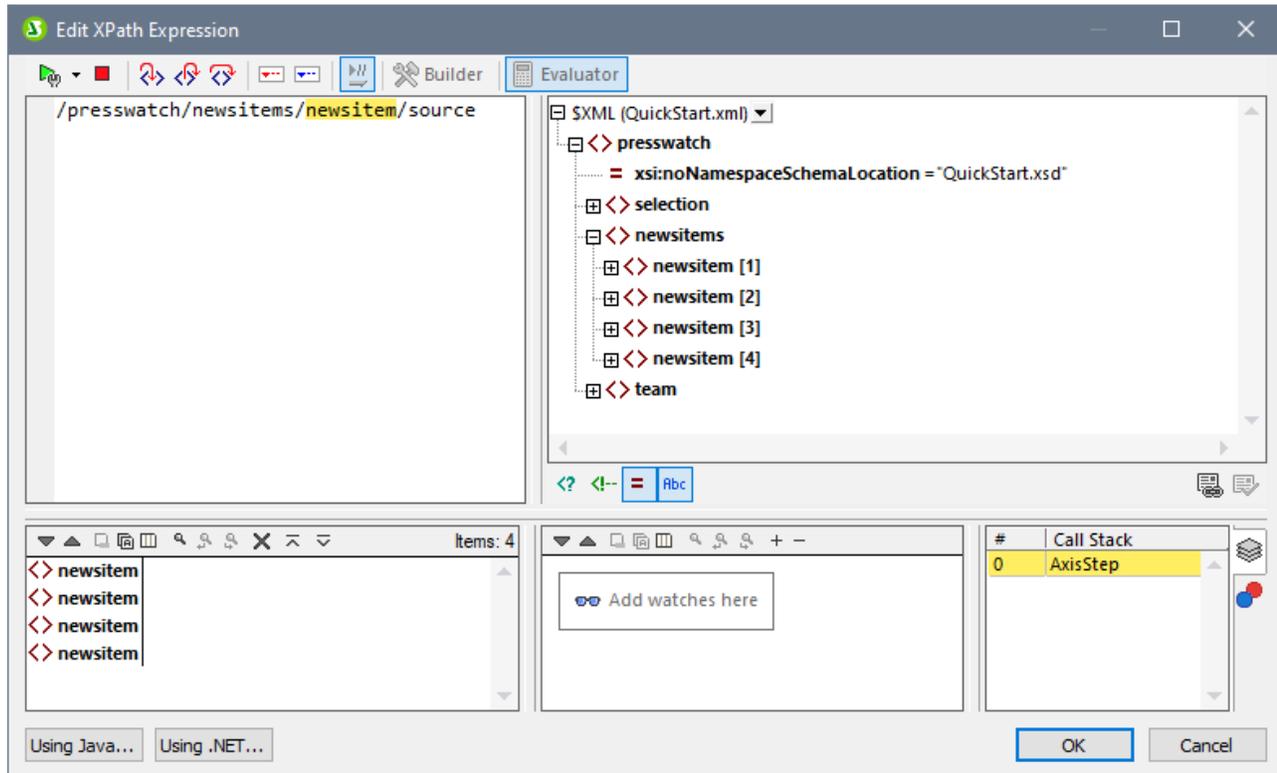
 *Buttons for debugging*

	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

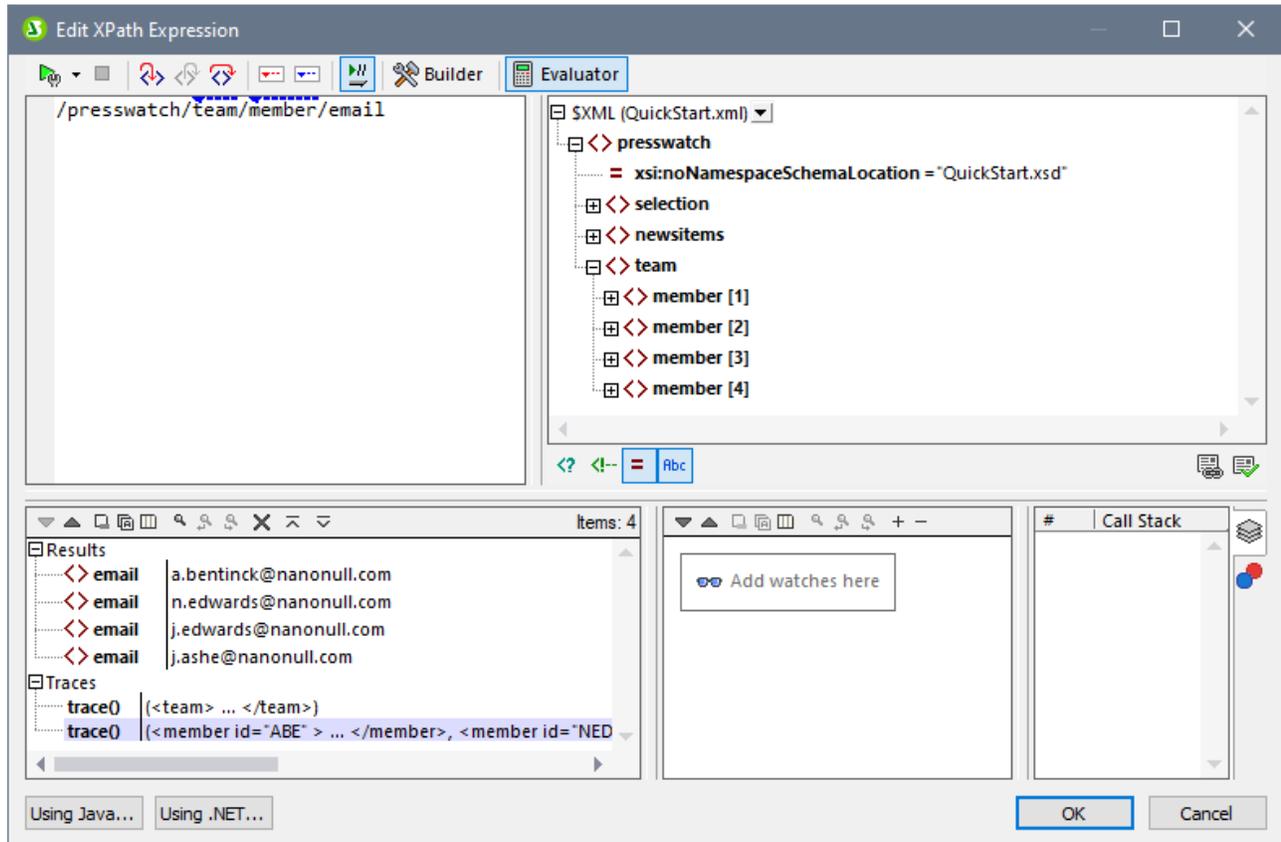
Breakpoints are points where you want the Debugger to stop after it has been started with **Start Debugger**. They are useful if you wish to analyze a specific part of the expression. When the Debugger stops at the breakpoint, you can check the result and could then use the **Step Into** functionality 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](#) ⁴⁰⁸ 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)**.

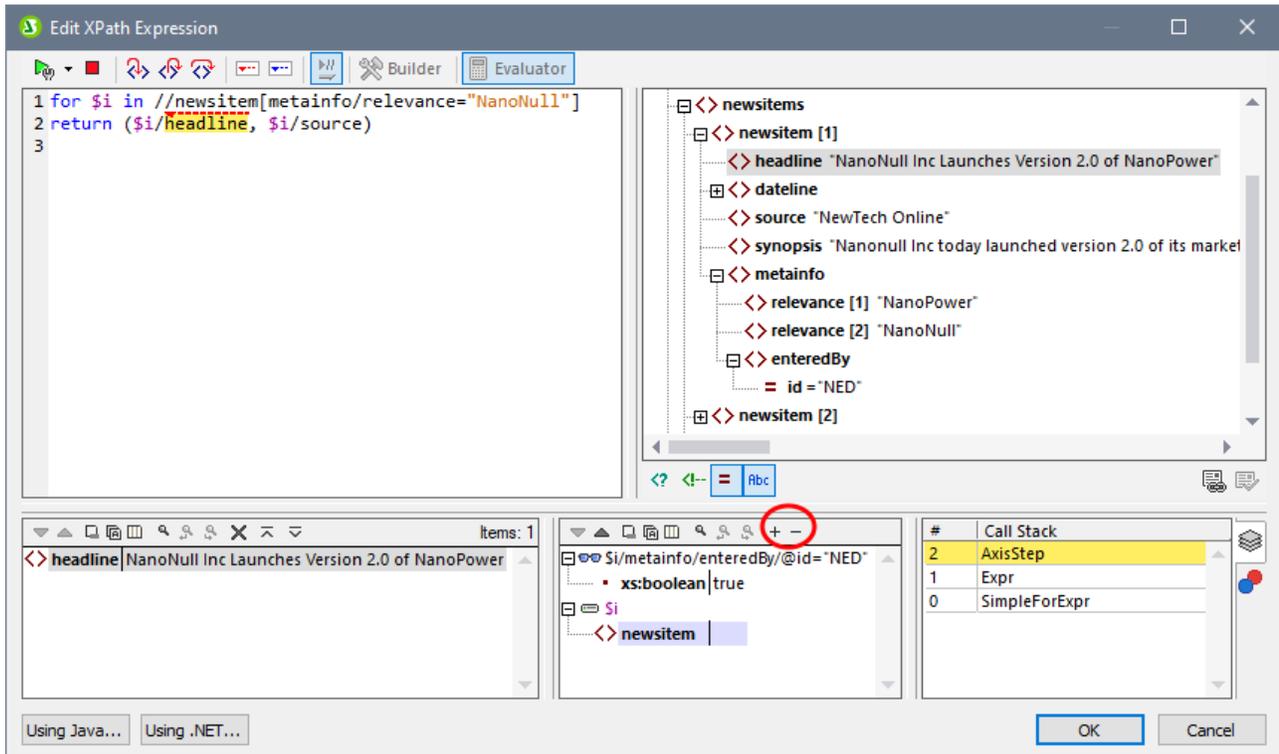


Note: If both a breakpoint and a tracepoint are set on a part of the expression, then the overline is composed of alternating red and blue dashes.

Also see [Debug Points](#)⁴⁰⁸ 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..

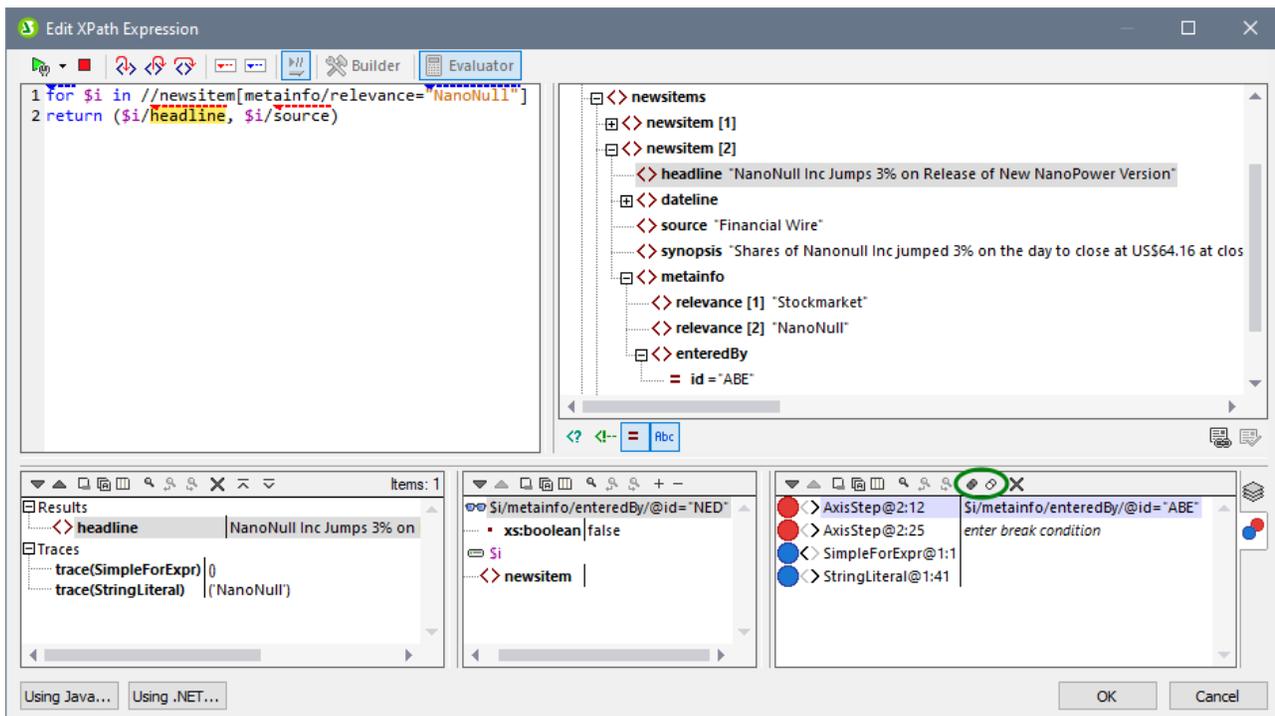
To enter a watch expression, click **Add Watch Entry** in the pane's toolbar (*encircled in red in the screenshot above*), then enter the expression and click **Enter** when done. To remove a watch expression, select it and click **Remove Selected Watch Entry** in the toolbar. If, during debugging, the 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.

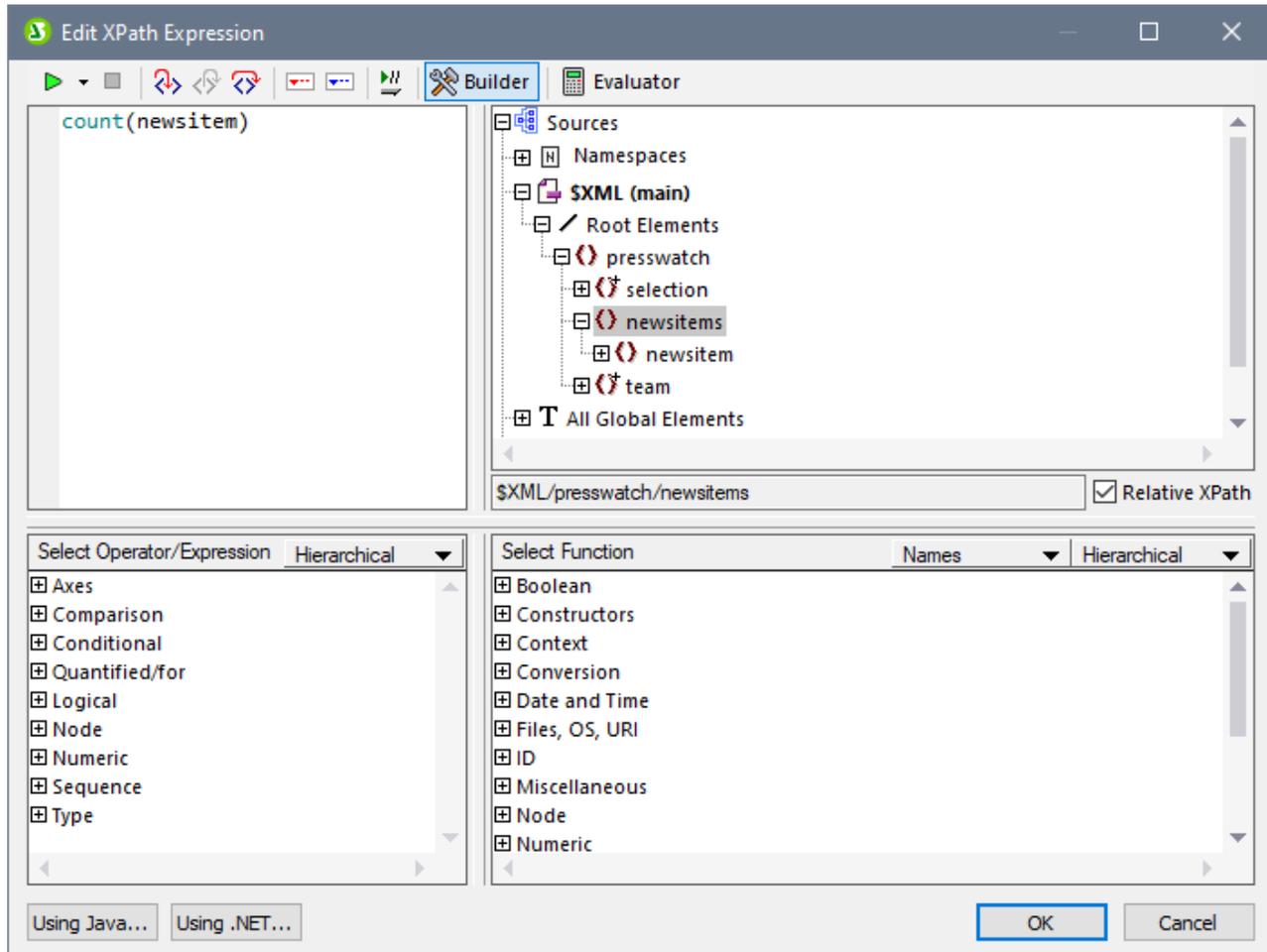
Toolbar commands in panes

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>Clear</i>	Clears the result list
<i>Collapse multi-line results to a single line</i>	If the value column of a result item contains multi-line text (text that includes newline character/s), you can toggle between a multi-line and single-line display
<i>Show complete result</i>	Shows the entire content of nodes as the value of the node

11.2.3 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.

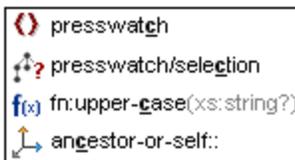
- Context node and schema tree
The *Selection* text box in the *Sources* pane immediately shows you the context node. The expression will be inserted at a location within this context node, and it will be evaluated with this node as its context.
- Inserting a node from the schema tree
In the *Sources* 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](#) ⁶⁰⁸ section of this **documentation**.

Intelligent editing during direct text entry

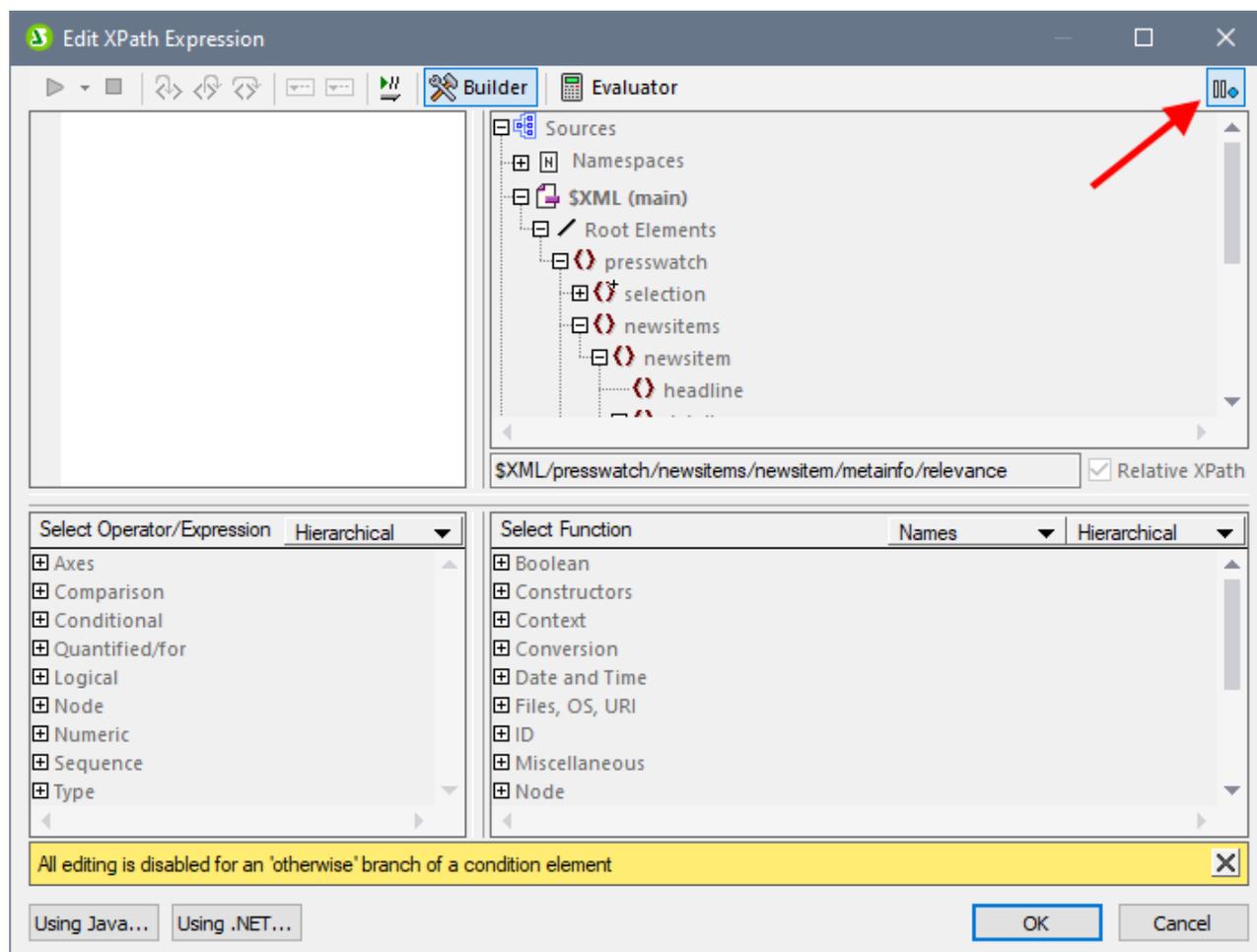
If you type an expression directly in the *Expression* text box, options that are available at that point are displayed in a popup (see *screenshot below*).



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](#)²⁴⁵.

11.3 Toolbars

A number of StyleVision commands are available as toolbar shortcuts, organized in the following toolbars:

- [Formatting](#) ⁴¹⁶
- [Table](#) ⁴¹⁷
- [Insert Design Elements](#) ⁴¹⁸
- [Design Filter](#) ⁴²⁰
- [Standard](#) ⁴²¹

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:

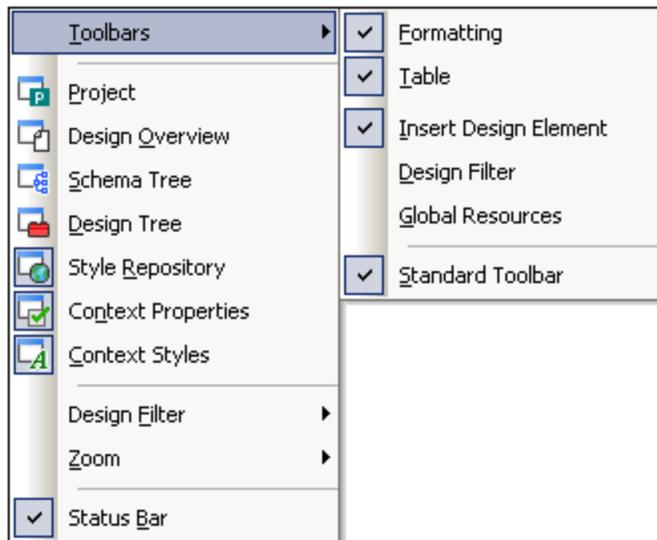
1. Grab the toolbar by its handle (if the toolbar is already in a toolbar area) or by its title bar (if the toolbar is floating).
2. Drag the toolbar to the desired toolbar area, if it exists, and drop it at the desired location in that toolbar area. If no toolbar area exists at the edge along which 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.

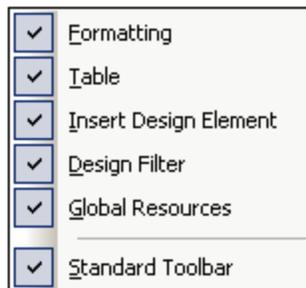
Switching the display of toolbars on and off

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.

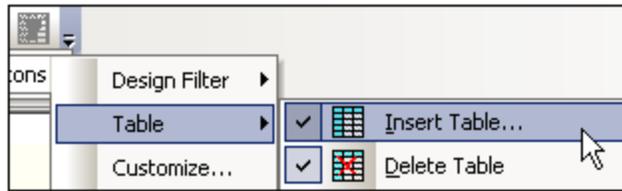


- In the Toolbars tab of the [Customize dialog](#) ⁵⁰⁹ (**Tools | Customize** ⁵⁰⁹), toggle the display of individual toolbars on or off by clicking a toolbar's check-box. When done, click the **Close** button to close the dialog.

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](#)⁵⁰⁹.

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.

11.3.1 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.

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

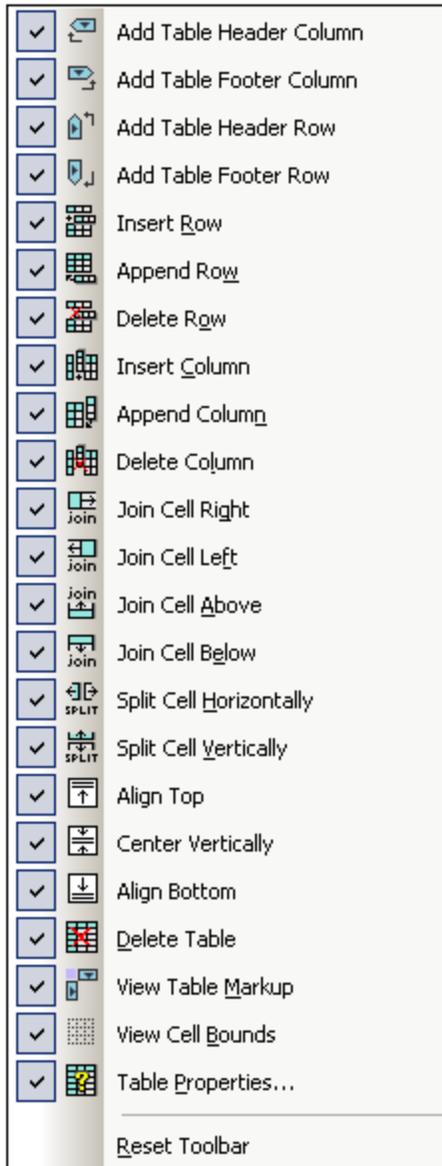
Lists can be inserted at the cursor insertion point, or the selection in the SPS can be converted to a list.

Hyperlinks

Inserts a hyperlink at the cursor insertion point. See [Hyperlink](#)⁴⁶⁷ for a description of how to use this command.

11.3.2 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.

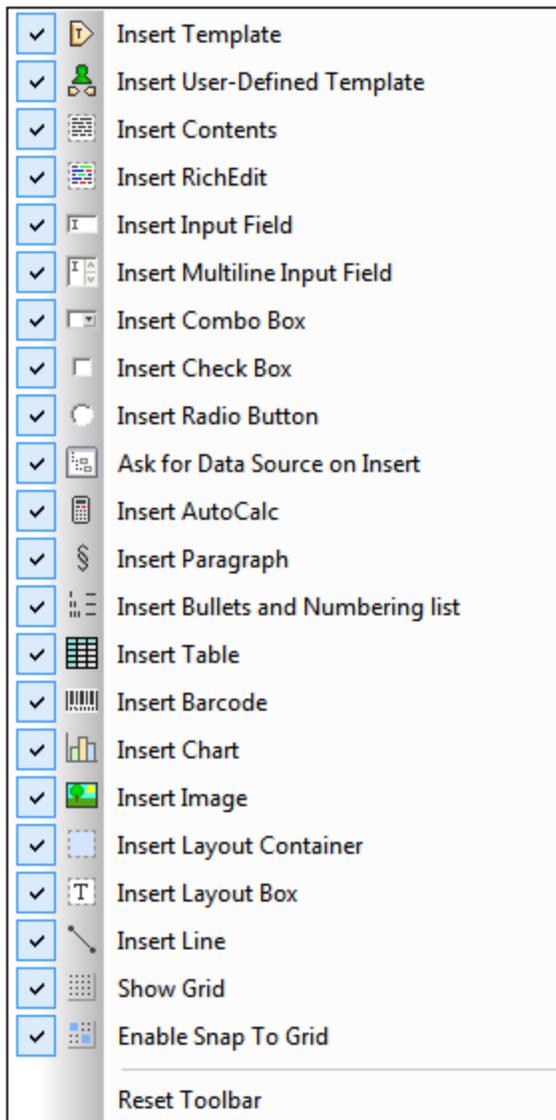
Table operations, properties, display

Placing the cursor in a static or dynamic table and clicking **Delete Table** ⁴⁸² 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.

11.3.3 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** ⁴¹⁹, which are context-node-sensitive (the majority of elements in the toolbar),
2. **Layout elements** ⁴²⁰, which are independent of node context, and
3. **Grid-related toggles** ⁴²⁰ 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 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](#)³⁹⁷ 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.

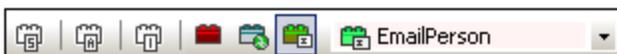
To re-size layout containers and layout boxes, place the cursor over the right or bottom border of the layout container or layout box and drag the border so as to obtain the desired size. To move a layout box, place the cursor over the top or left border of the layout box and, when the cursor turns to 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**).

11.3.4 Design Filter

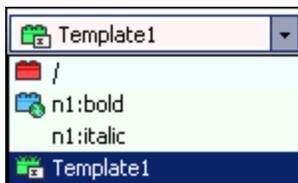
The **Design Filter toolbar** (*screenshot below*) contains commands that enable you to filter which templates are displayed in the design. Each icon in the toolbar is explained below.



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) .

Icon	Command	Description
	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.

11.3.5 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.

Btn	Command	Shortcut	Description
	New from XML Schema / DTD ⁴²³	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 ⁴²⁹	Ctrl+O	Opens an existing SPS document.
	Save Design ⁴³⁴	Ctrl+S	Saves the active SPS document.
	Save All ⁴³⁴	Ctrl+Shift+S	Saves all open SPS documents.
	Print ⁴⁴⁴	Ctrl+P	Prints the Authentic View of the Working XML file.

Btn	Command	Shortcut	Description
	Print Preview ⁴⁴⁴		Displays a print preview of the Authentic View of the Working XML File.
	Cut ⁴⁴⁷	Shift+Del	Cuts the selection and places it in the clipboard.
	Copy ⁴⁴⁷	Ctrl+C	Copies the selection to the clipboard.
	Paste ⁴⁴⁷	Ctrl+P	Pastes the clipboard item to the cursor location.
	Delete ⁴⁴⁷	Del	Deletes the selection.
	Undo ⁴⁴⁷	Alt+ Backspace	Undoes an editing change. An unlimited number of Undo actions can be performed at a time.
	Redo ⁴⁴⁷	Ctrl+Y	Redoes an undo.
	Find ⁴⁴⁷	Ctrl+F	Finds text in Authentic View and Output Views.
	Find Next ⁴⁴⁷	F3	Finds the next occurrence of the searched text.
	XSLT 1.0 ⁹²		Sets XSLT 1.0 as the stylesheet language.
	XSLT 2.0 ⁹²		Sets XSLT 2.0 as the stylesheet language.
	XSLT 3.0 ⁹²		Sets XSLT 3.0 as the stylesheet language.
	Spelling ⁴⁹⁰		Runs a spelling check on the SPS document.

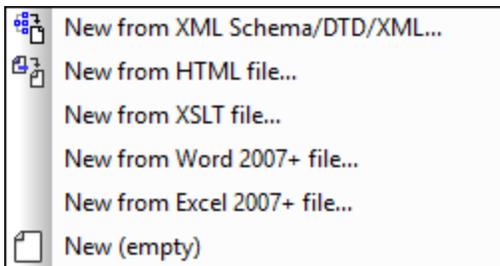
11.4 File Menu

The **File** menu contains commands for working with SPSs and related files. The following commands are available:

- [New](#)⁴²³, to create a new SPS from a variety of sources.
- [Open, Reload, Close, Close All](#)⁴²⁹, to open and close the active file, and to reload the active file.
- [Save Design, Design As, All](#)⁴³⁴, which are commands to save the active SPS and all open SPS files.
- [Export as MobileTogether Design File](#)⁴⁴⁰, to generate a MobileTogether design from the active SPS file.
- [Save Generated Files](#)⁴⁴⁰, to save output files that can be generated using the SPS.
- [Web Design](#)⁴⁴³, generates all the files required to run an ASPX application, in the folder location you specify.
- [Properties](#)⁴⁴³, 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](#)⁴⁴⁴, enabled in output views, these commands print what is displayed in the previews.
- [Most Recently Used Files, Exit](#)⁴⁴⁵, respectively, to select a recently used file to open, and to exit the program.

11.4.1 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:



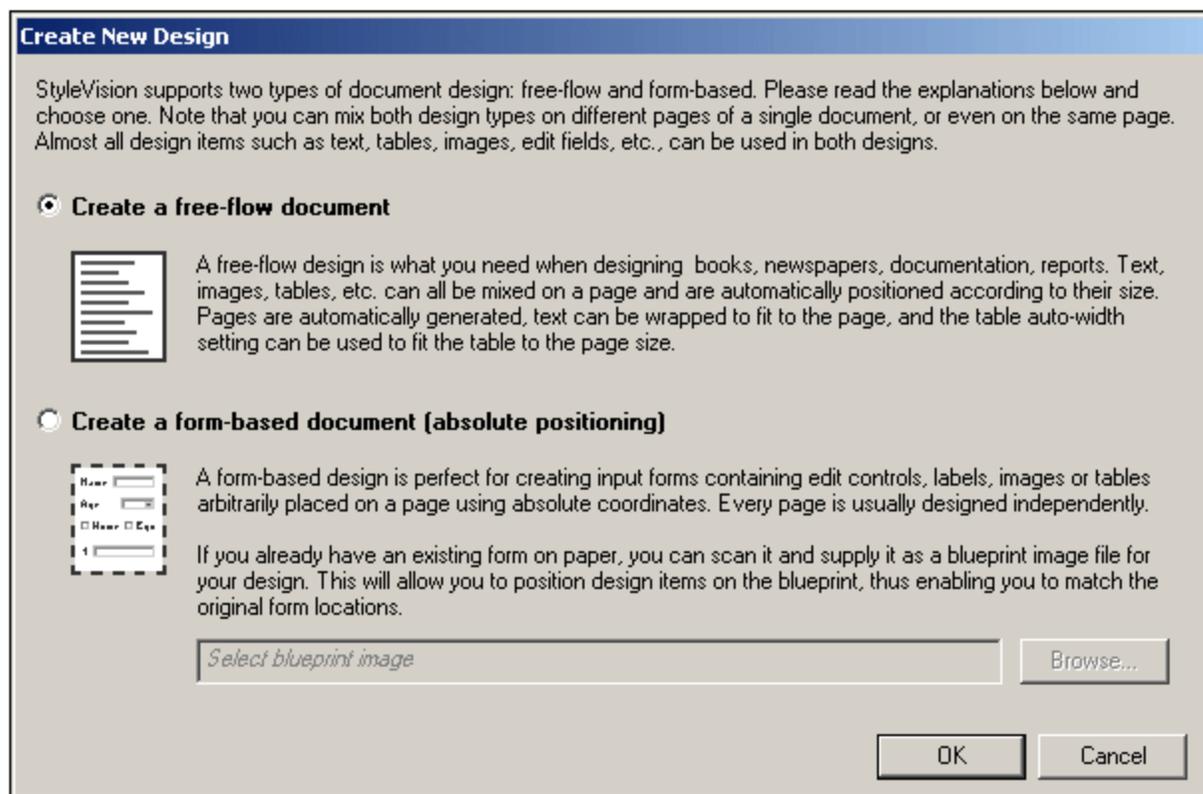
- A new SPS file based on an XML Schema or DTD or XML Schema generated from an XML file (**New from XML Schema / DTD / XML**). The selected schema is added to the [Design Overview sidebar](#)³² and a graphical tree representation is added to the schema tree (in the [Schema Tree sidebar](#)³⁵). In [Design View](#)²⁷, 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 based on a user-defined schema you create node-by-node from an [HTML file](#)³⁶⁷ (**New from HTML File**). The user-defined schema is added to the [Design Overview sidebar](#)³² and [Schema Tree sidebar](#)³⁵. In the schema tree, it will have a single document element (root element), and the HTML file is loaded in [Design View](#)²⁷.
- 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](#)³⁴⁰ for details.

- A new SPS that contains the [content of a MS Word document as the design's static text](#)¹⁰⁷.
- A new SPS that contains the [content of a MS Excel document as the design's static text](#)¹¹⁰.
- 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](#)²⁷.

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).

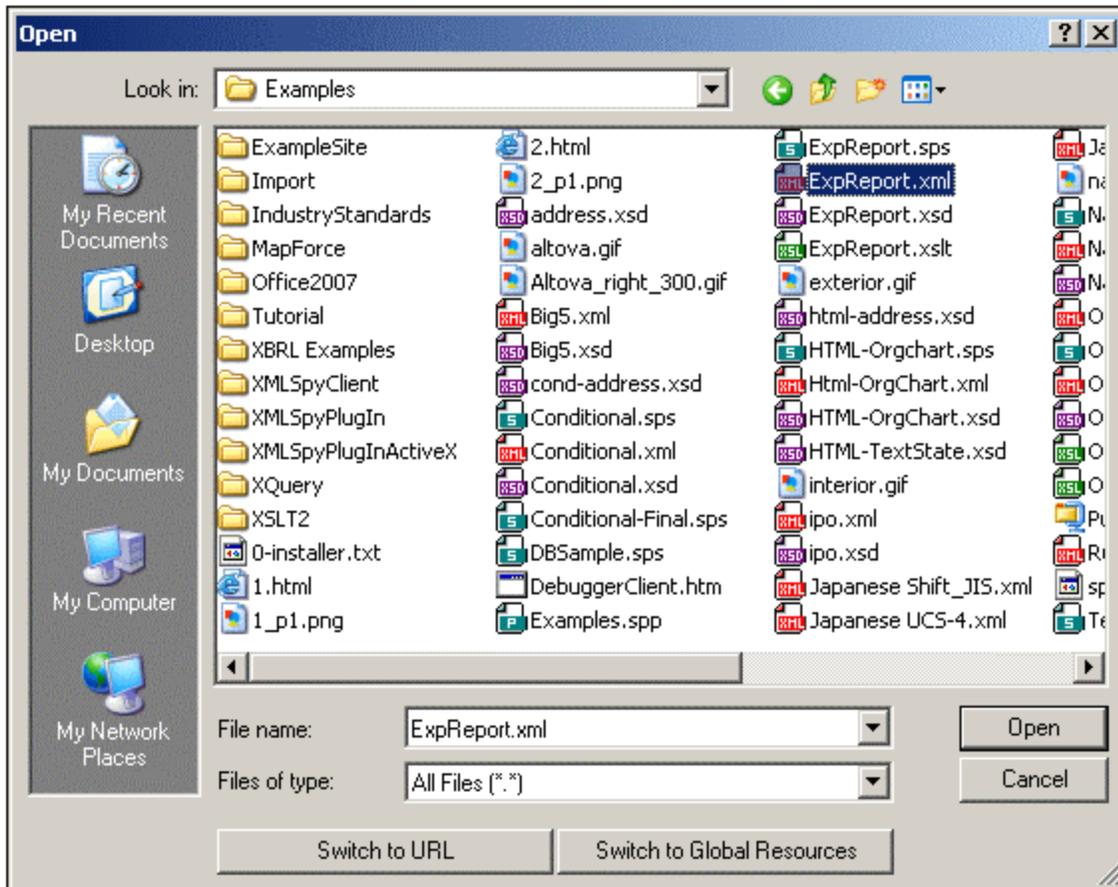


In a free-flowing document design, document content is laid out to fit the output media object or viewer (paper or screen). Items in the document content can only be placed relative to each other, and not absolutely. This kind of design is suited for documents such as reports, articles, and books.

In a form-based document, a single [Layout Container](#)¹⁵⁹ 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](#)¹⁵⁹. 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

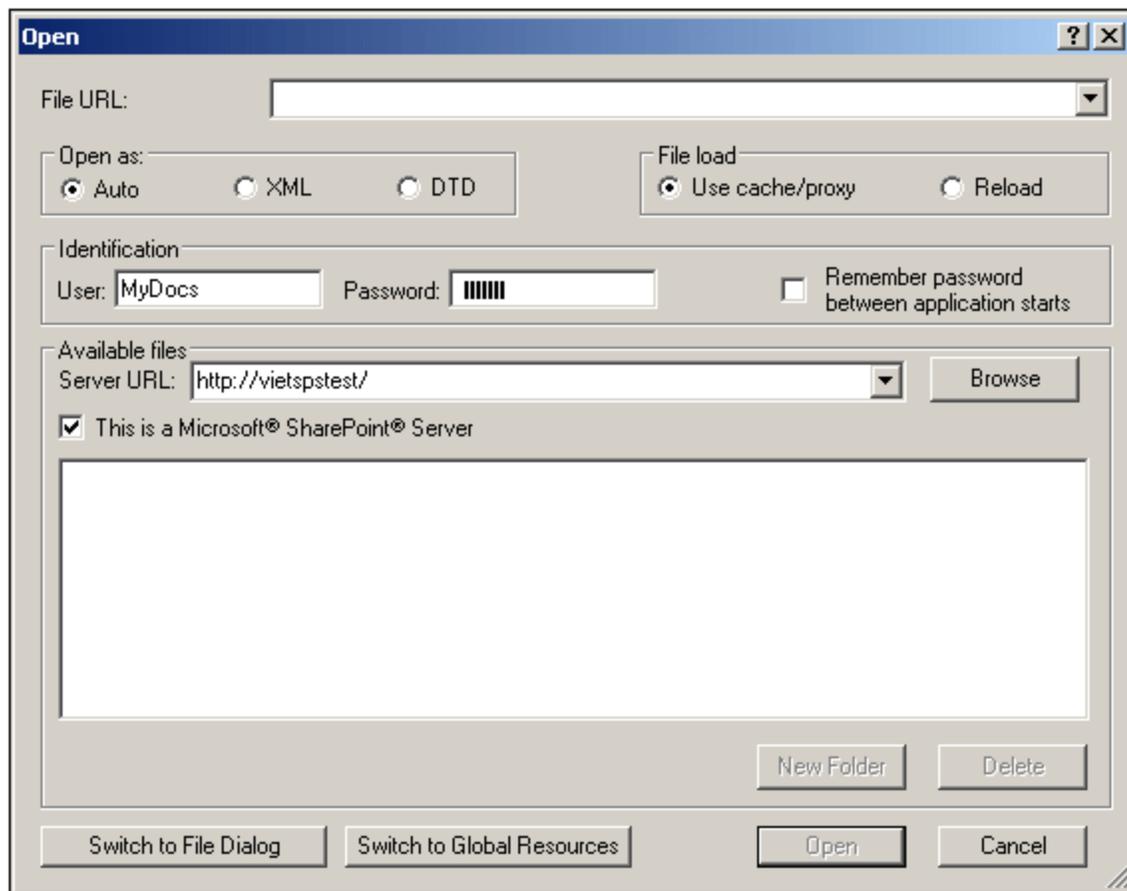
In several File Open and File Save dialogs, you can choose to select the required file or save a file via a URL or a global resource (see *screenshot below*). Select the **Switch to URL** or **Switch to Global Resource** to go to one of these selection processes.



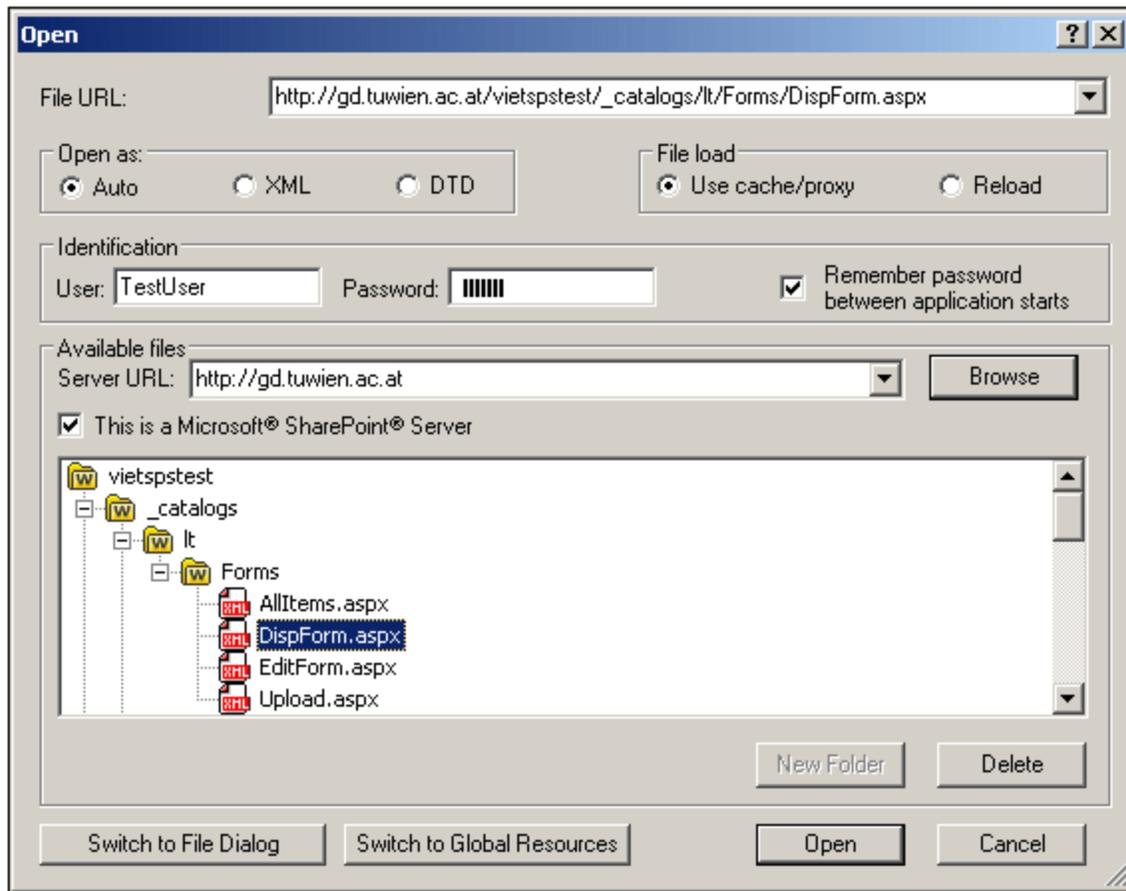
Selecting files via URLs

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.

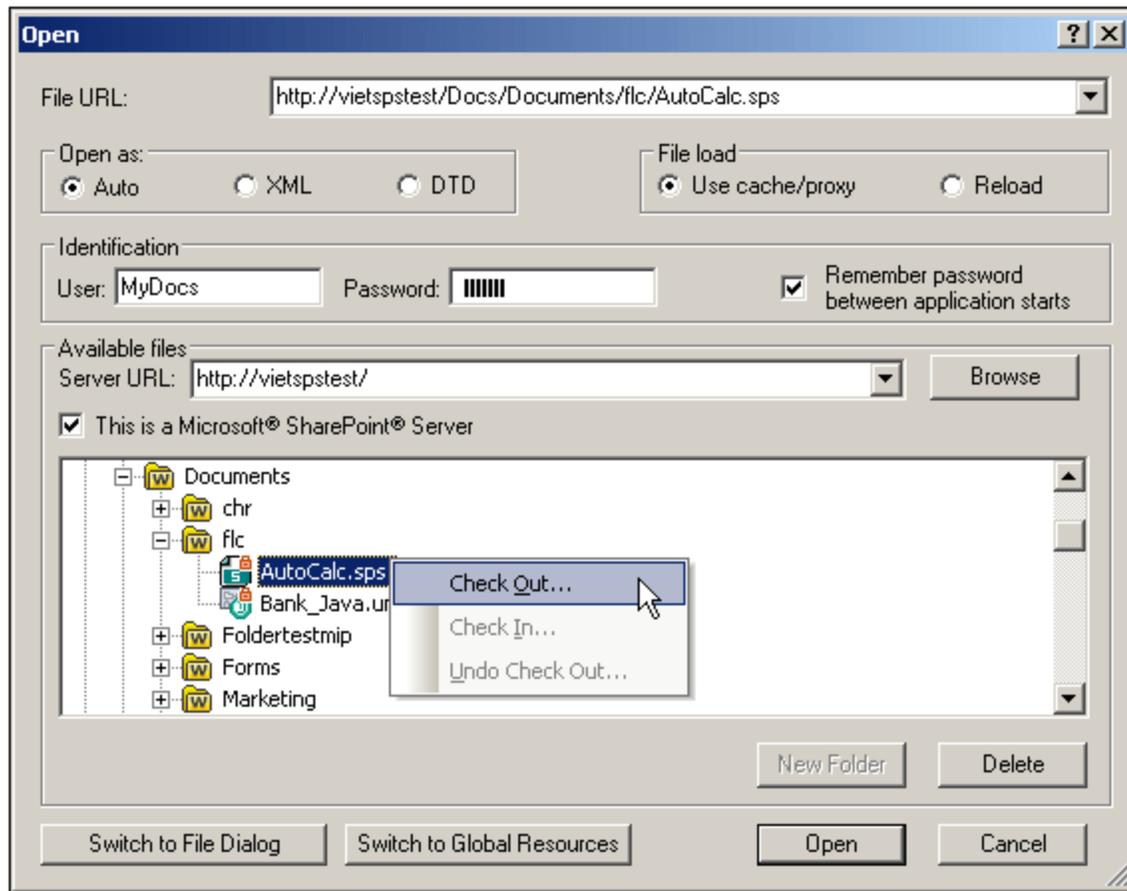
Note: The Browse function is only available on servers which support WebDAV and on Microsoft SharePoint Servers. The supported protocols are FTP, HTTP, and HTTPS.

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.

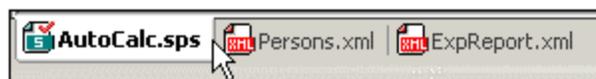


Right-clicking a file pops up a context menu containing commands available for that file (*screenshot above*).

- 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**.

11.4.2 Open, Reload, Close, Close All

The **Open** (**Ctrl+O**) command  allows you to open an existing SPS or PXF file. The familiar [Open dialog](#) ⁴²⁴ of Windows systems is opened and allows you to select a file with an extension of `.sps`.

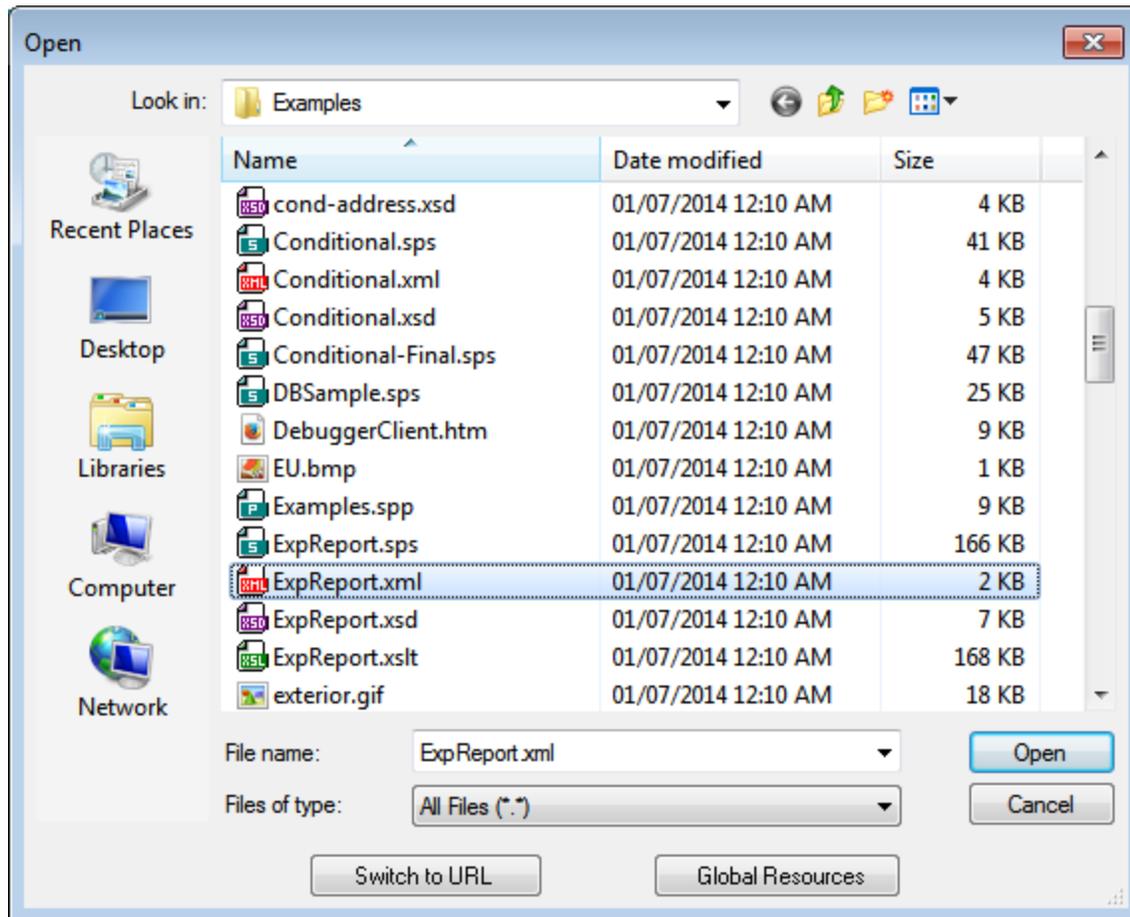
The **Reload** command reloads the SPS file from the file saved to disk. Any changes made since the file was last saved will be lost. The Working XML file will also be reloaded, enabling you to update the Working XML File if it has been changed externally.

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](#) ²⁶. 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

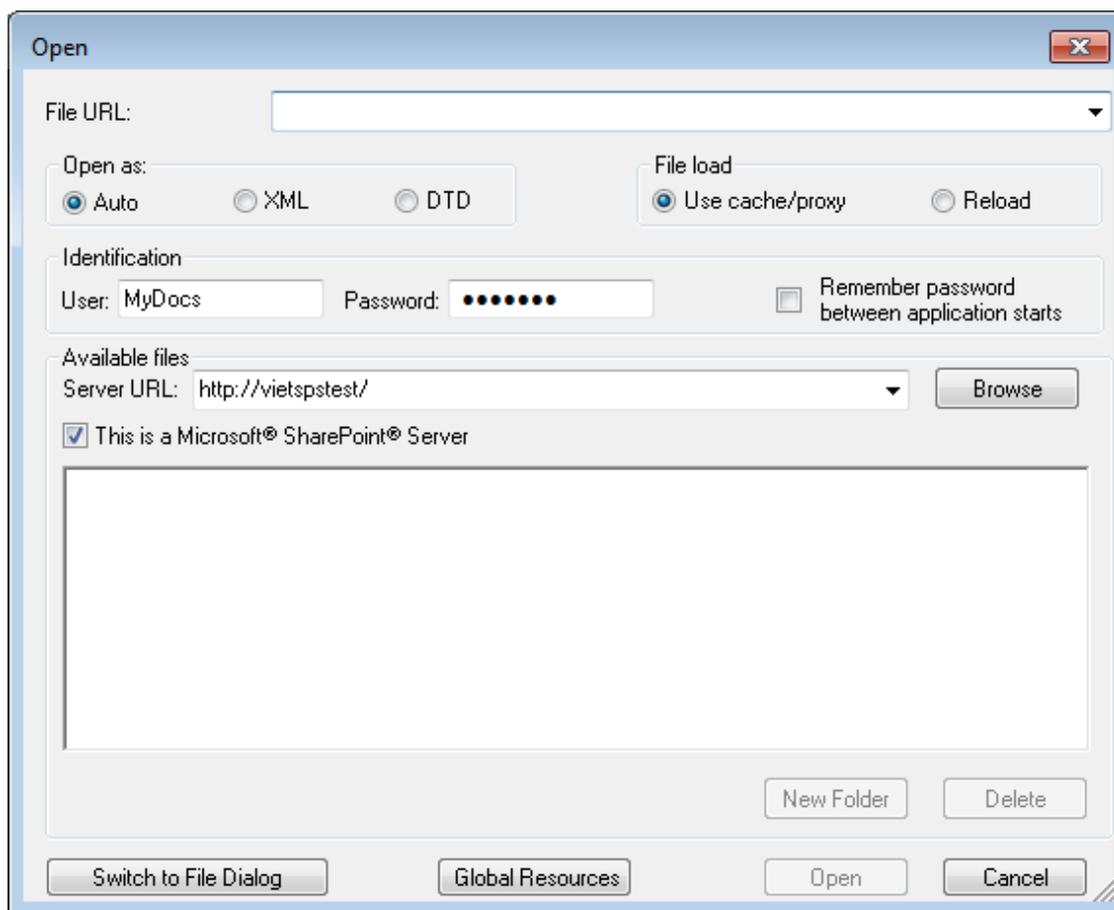
In several File Open and File Save dialogs, you can choose to select the required file or save a file via a URL or a global resource (*see screenshot below*). Click **Switch to URL** or **Global Resource** to go to one of these selection processes.



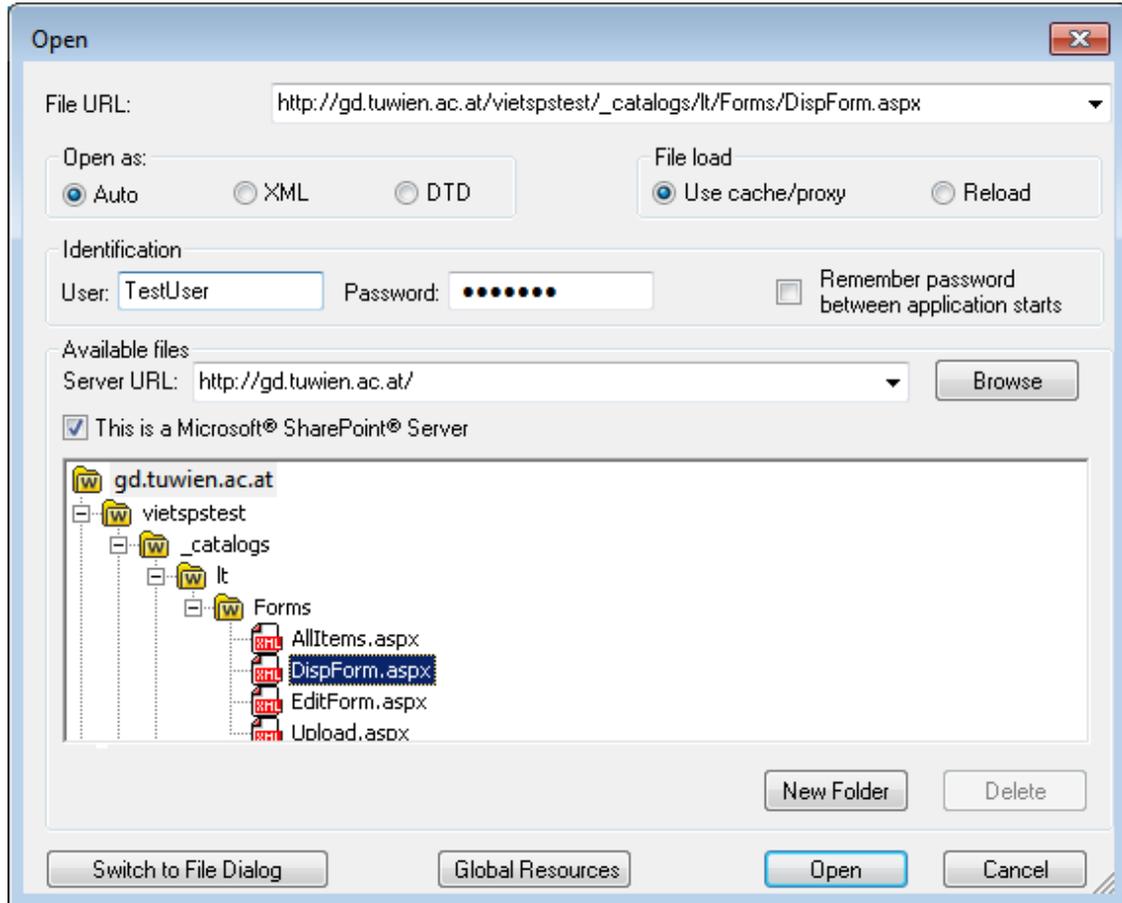
Selecting files via URLs

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.

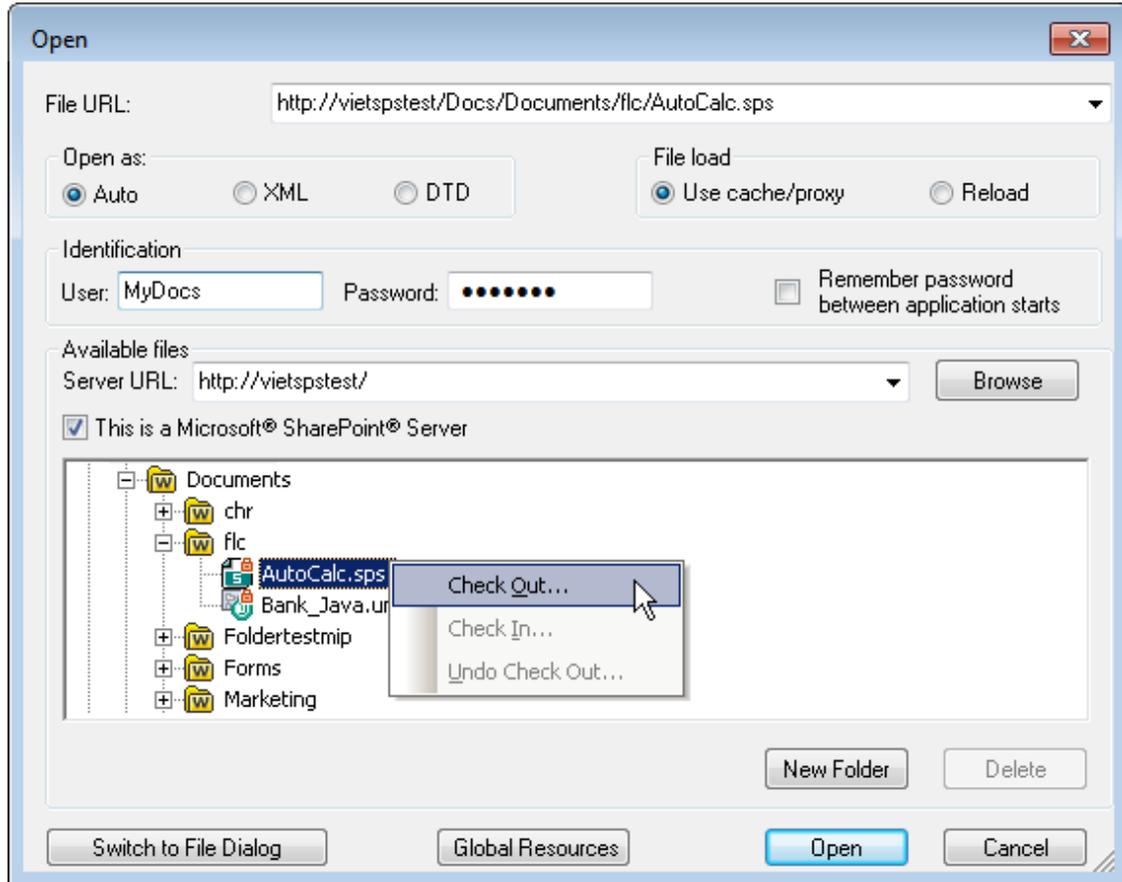
Note the following:

- 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.

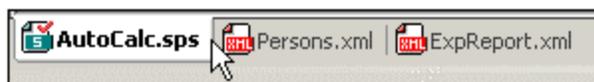


Right-clicking a file pops up a context menu containing commands available for that file (screenshot above).

- 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,. For a general description of Global Resources, see the section in this documentation.

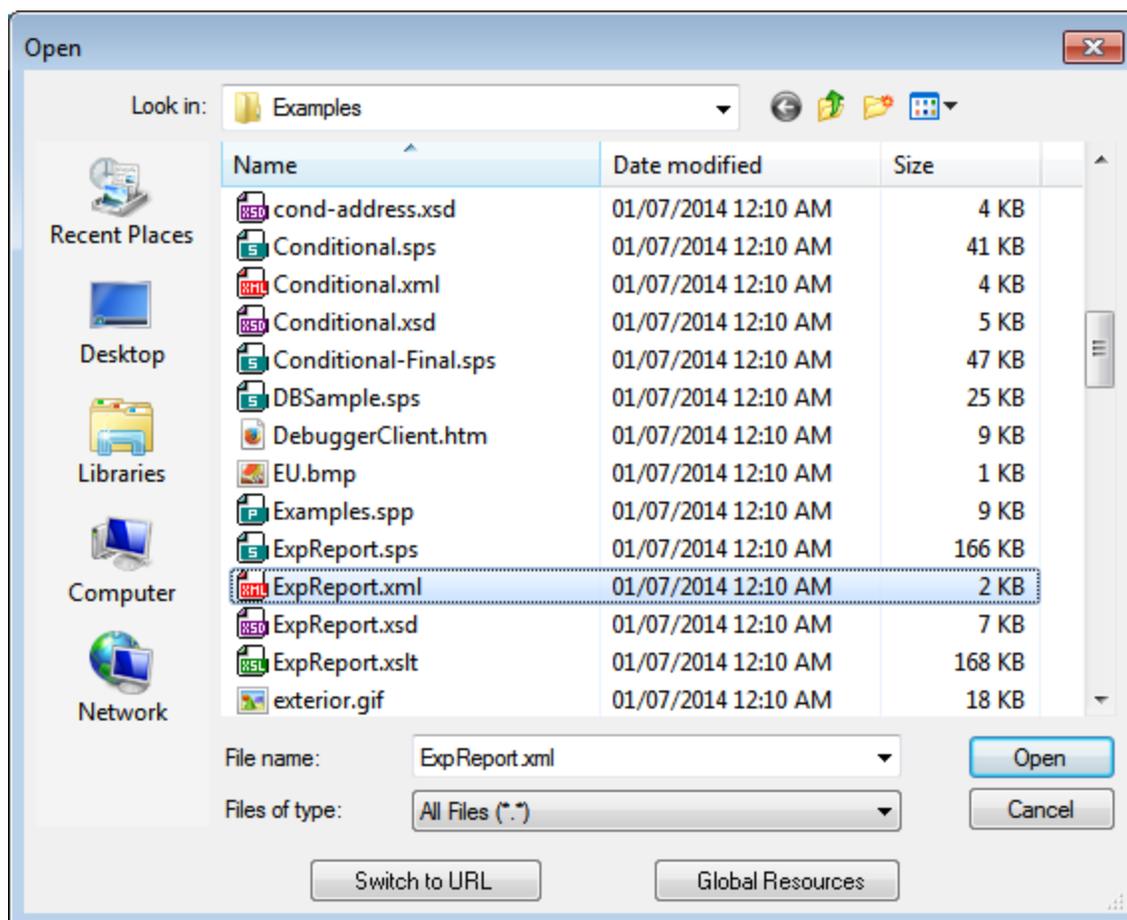
11.4.3 Save Design, Save All

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

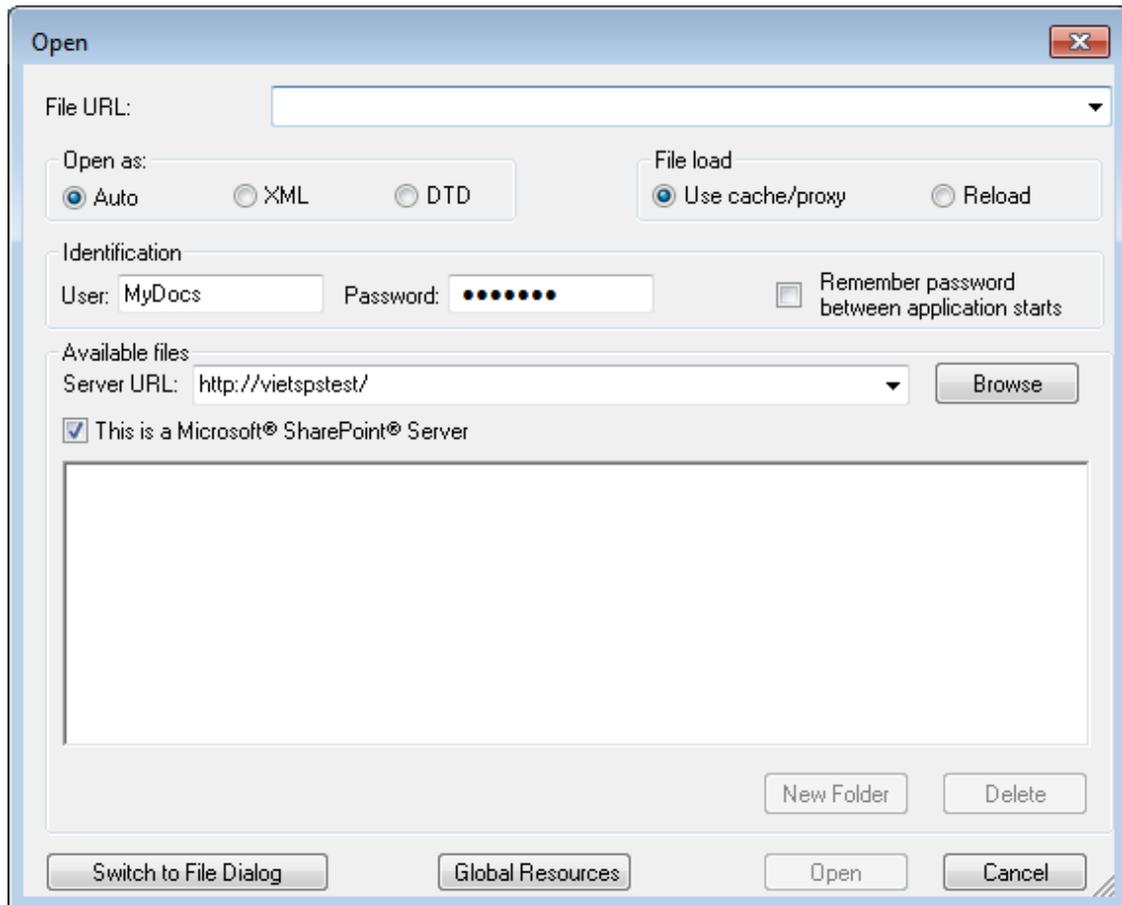
In several File Open and File Save dialogs, you can choose to select the required file or save a file via a URL or a global resource (*see screenshot below*). Click **Switch to URL** or **Global Resource** to go to one of these selection processes.



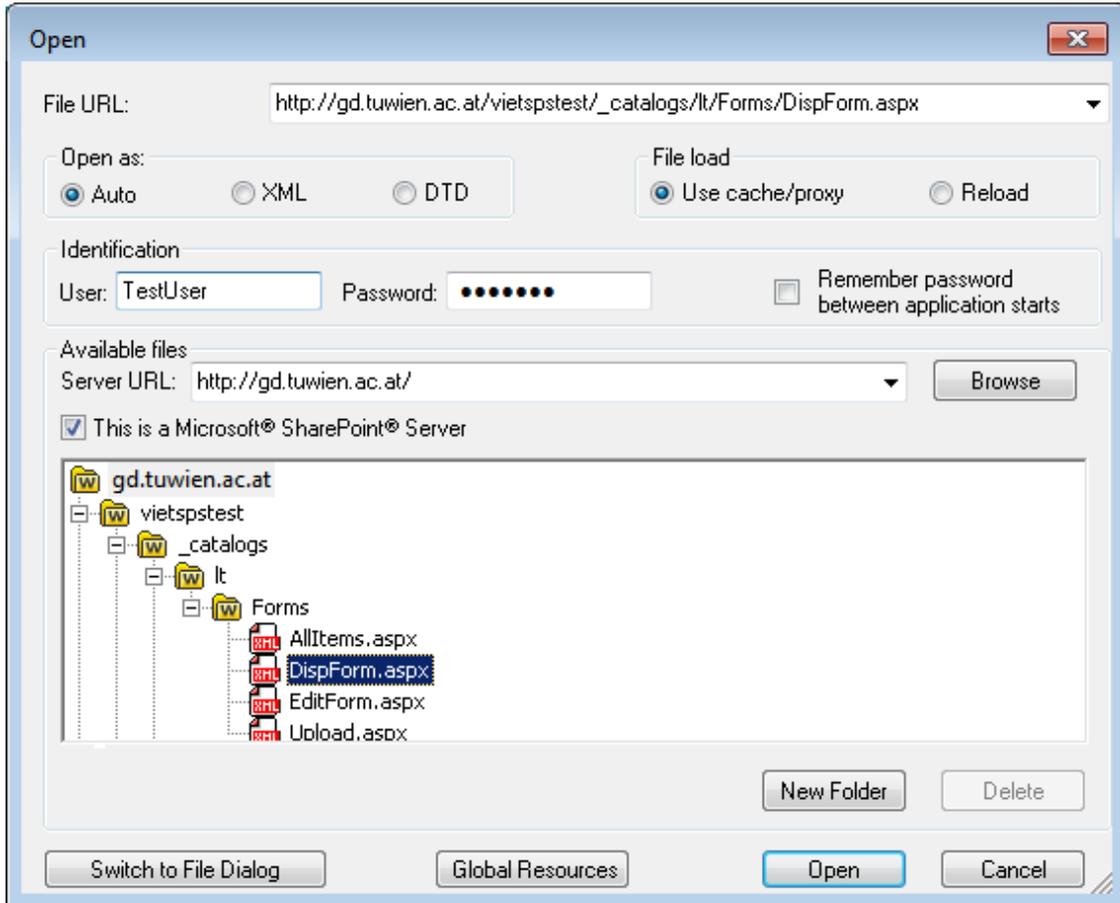
Selecting files via URLs

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.

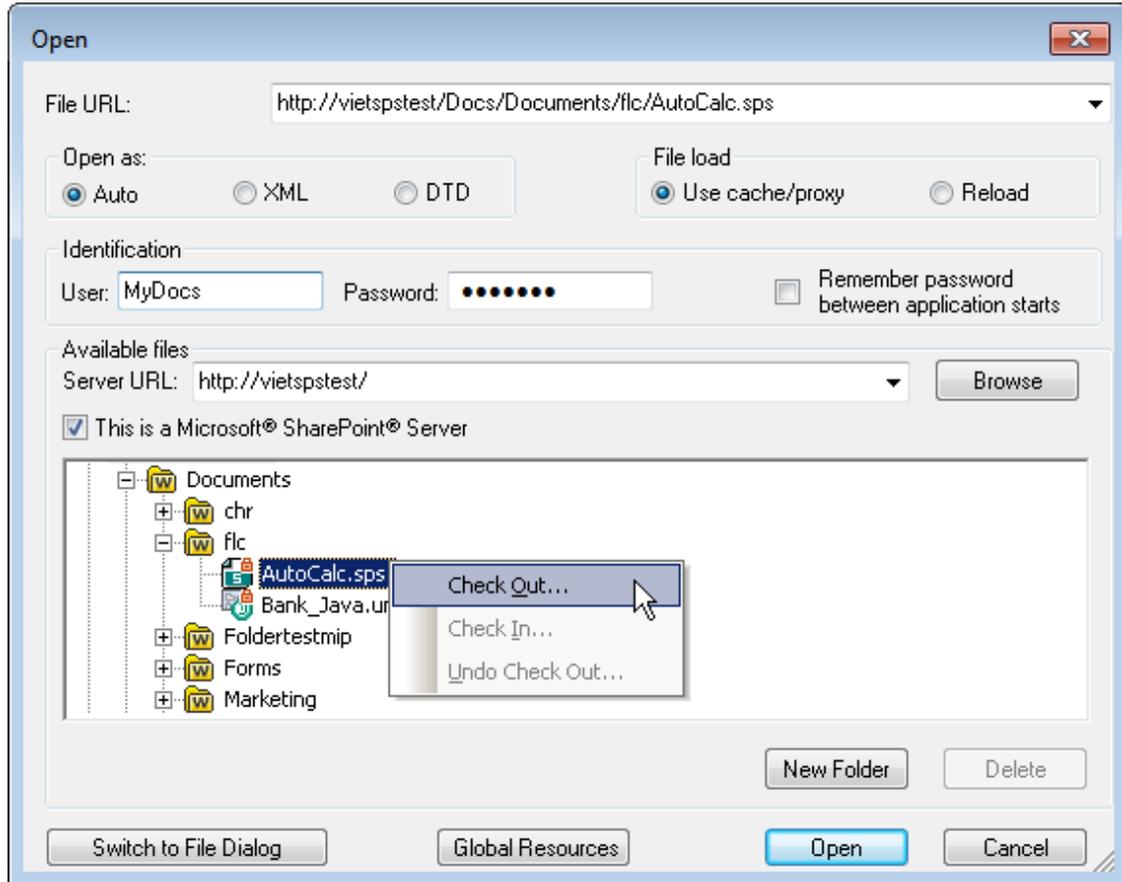
Note the following:

- 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.

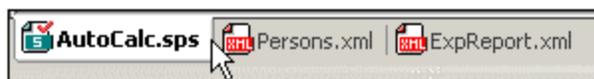


Right-clicking a file pops up a context menu containing commands available for that file (screenshot above).

- 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

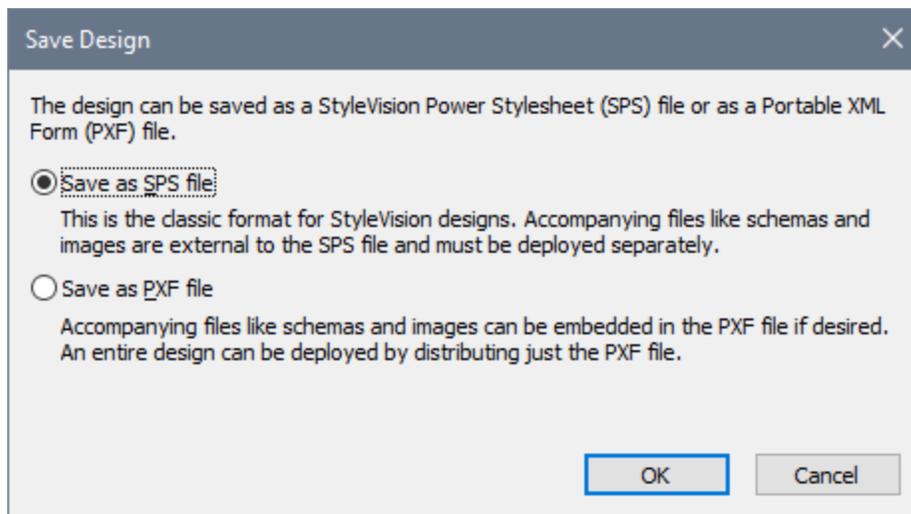
- command in the context menu. This has the effect of returning the file unchanged to the server.
- If you check out a file in one Altova application, you cannot check it out in another Altova application. The file is considered to be already checked out to you. The available commands at this point in 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,. For a general description of Global Resources, see the section in this documentation.

11.4.4 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 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](#)⁴³⁴ [command](#)⁴³⁴. 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.

11.4.5 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](#). Conversion options are available in the MobileTogether Design tab of the Options dialog ([Tools | Options](#)⁵¹³).

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](#)²¹⁵; (ii) [modules](#)²⁰¹.

11.4.6 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](#)⁹⁵.

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](#)²³¹ 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.
- An XSLT file, which is automatically generated from the currently active SPS file.

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.

11.4.7 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*).

Deploy Transformation

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\FlowForceServer2023\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\StyleVisionServer2023\etc`.

Note: For information about how to work with FlowForce Server, see the [FlowForce documentation](#).

11.4.8 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](#)³⁷⁴.

11.4.9 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 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 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](#) ⁴⁴³ 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.

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.

11.4.10 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.

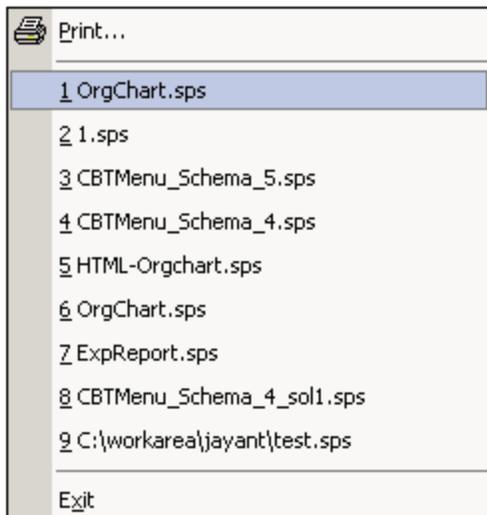
To navigate the pages of the preview, use the page navigation buttons at the bottom of the preview or by entering the page number in the Page text-box.

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**.

11.4.11 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.

11.5 Edit Menu

The **Edit** menu contains commands that aid the editing of SPS 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](#)⁴⁴⁷, to undo or restore your previous actions, and to select all content of the SPS.
- [Find, Find Next, Replace](#)⁴⁴⁷, to find text in the SPS and XSLT stylesheet previews.
- [Stylesheet Parameters](#)⁴⁵², to edit parameters declared globally for the SPS.
- [Collapse/Expand Markup](#)⁴⁵³, 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.

11.5.1 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.

11.5.2 Find, Find Next, Replace

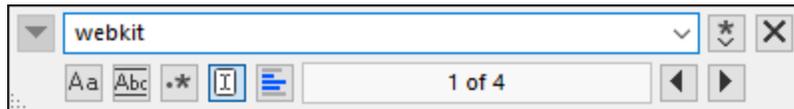
The **Find (Ctrl+F)** command  enables you to find text strings in Design View, JavaScript Editor, and XSLT stylesheets.

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](#)⁴⁸. The search and replace functionality of Design View is described in the topic [Find & Replace sidebar](#)⁴⁸.
- The search and replace functionality in JavaScript Editor and XSLT stylesheets is described in this topic.

XSLT stylesheets and JavaScript Editor

Clicking the **Find** command in the XSLT-for-HTML 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](#)⁴⁴⁶ 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

Clicking the **Find** command in HTML Preview opens a simple [Find and Replace](#)⁴⁴⁸ 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`.

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](#)⁴⁴⁷ 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)	<p>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).</p> <p>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.</p>
\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, <i>see above</i>).
\$	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.

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

\\	Use this to escape characters that appear in regex expression, for example: \\n
----	---

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.

- 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".
- 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.

```

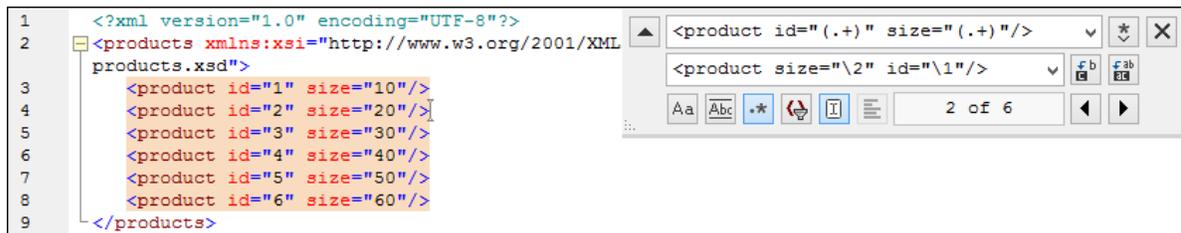
<?xml version="1.0" encoding="UTF-8"?>
<products xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="products.xsd">
  <product id="1" size="10"/>
  <product id="2" size="20"/>
  <product id="3" size="30"/>
  <product id="4" size="40"/>
  <product id="5" size="50"/>
 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

```

To solve the requirement, follow the steps below:

- Press **Ctrl+H** to open the Find and Replace dialog box.

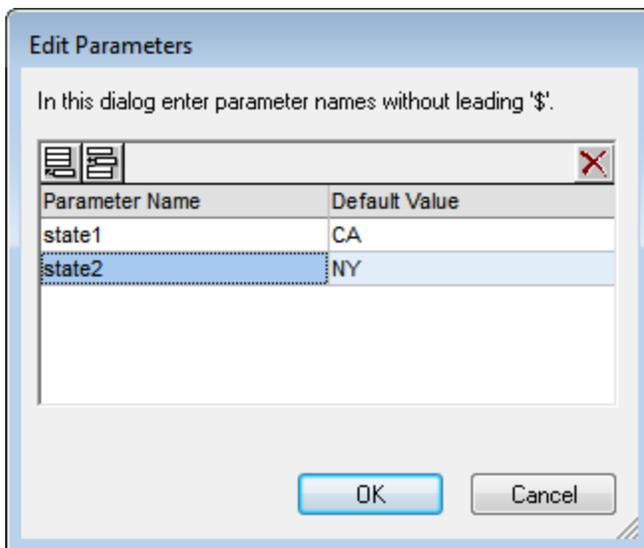
2. Click **Use regular expressions** .
3. In the Find field, enter the following: `<product id="(.)" size="(.)"/>`. This regular expression matches a product element in the XML document. Notice that, in order to match the value of each attribute (which is not known in advance), a tagged expression `(.)` 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`.

11.5.3 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.



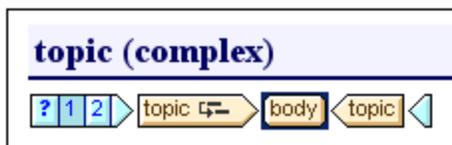
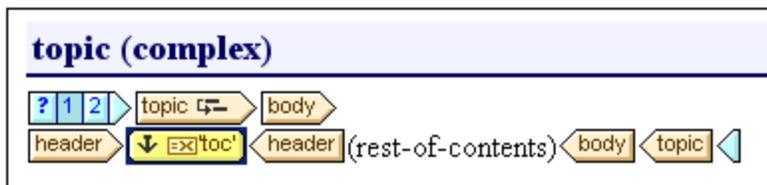
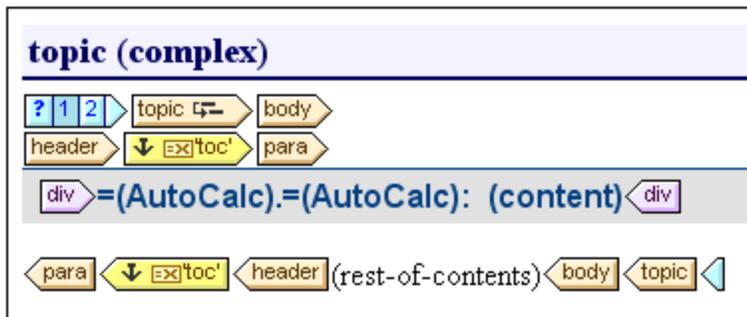
The following points should be noted:

- You can insert, append, edit and delete parameters for the entire stylesheet.
- 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](#) ³².

11.5.4 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.

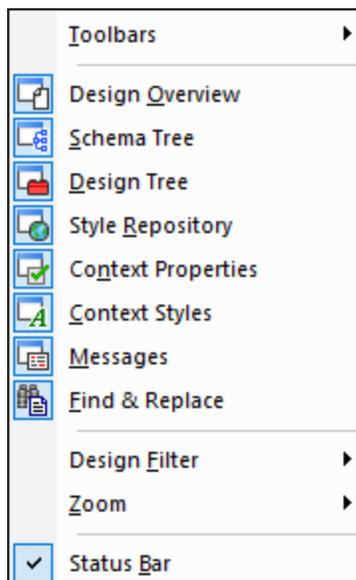
The screenshots below show how a series of tags are collapsed. Double-clicking a collapsed tag expands it.



Collapsing a tag can be useful for optimizing the display according to your editing needs.

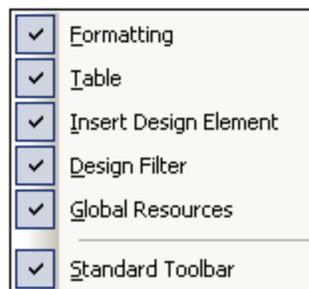
11.6 View Menu

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.



11.6.1 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.



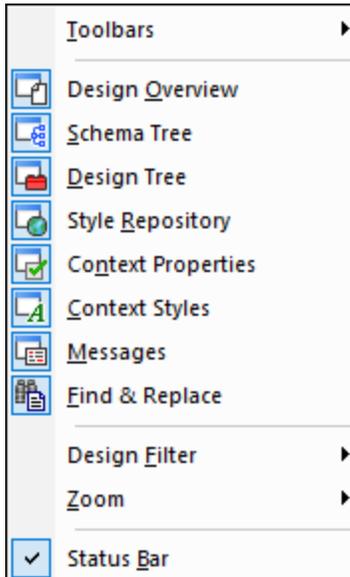
When a toolbar is checked, it is displayed. In the screenshot above all the toolbars are displayed. To toggle on or off the display of a toolbar, click the appropriate toolbar. For a complete description of toolbars, see the section [Reference | Toolbars](#) ⁴¹⁴.

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.

11.6.2 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, Output View, no document open) is retained for that particular View till changed.

11.6.3 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	Toggles the display of the main template on and off.

Icon	Command	Description
	template	
	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](#) ⁴²⁰ 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.

11.7 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](#)⁴¹⁴. 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.

11.7.1 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](#)⁴¹⁸, 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.

11.7.2 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.

11.7.3 Form Controls

Mousing over the **Form Controls** command rolls out a submenu (*screenshot below*) containing commands to insert various form controls ([data-entry devices](#)¹⁴⁸).



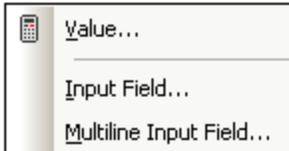
How to create each of these form controls is described in the section [Using Data-Entry Devices](#)¹⁴⁸. After a form control has been created, its properties can be edited by selecting it and then editing the required property in the [Properties sidebar](#)⁴⁴.

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](#)⁴¹⁸, 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.

11.7.4 Auto-Calculation

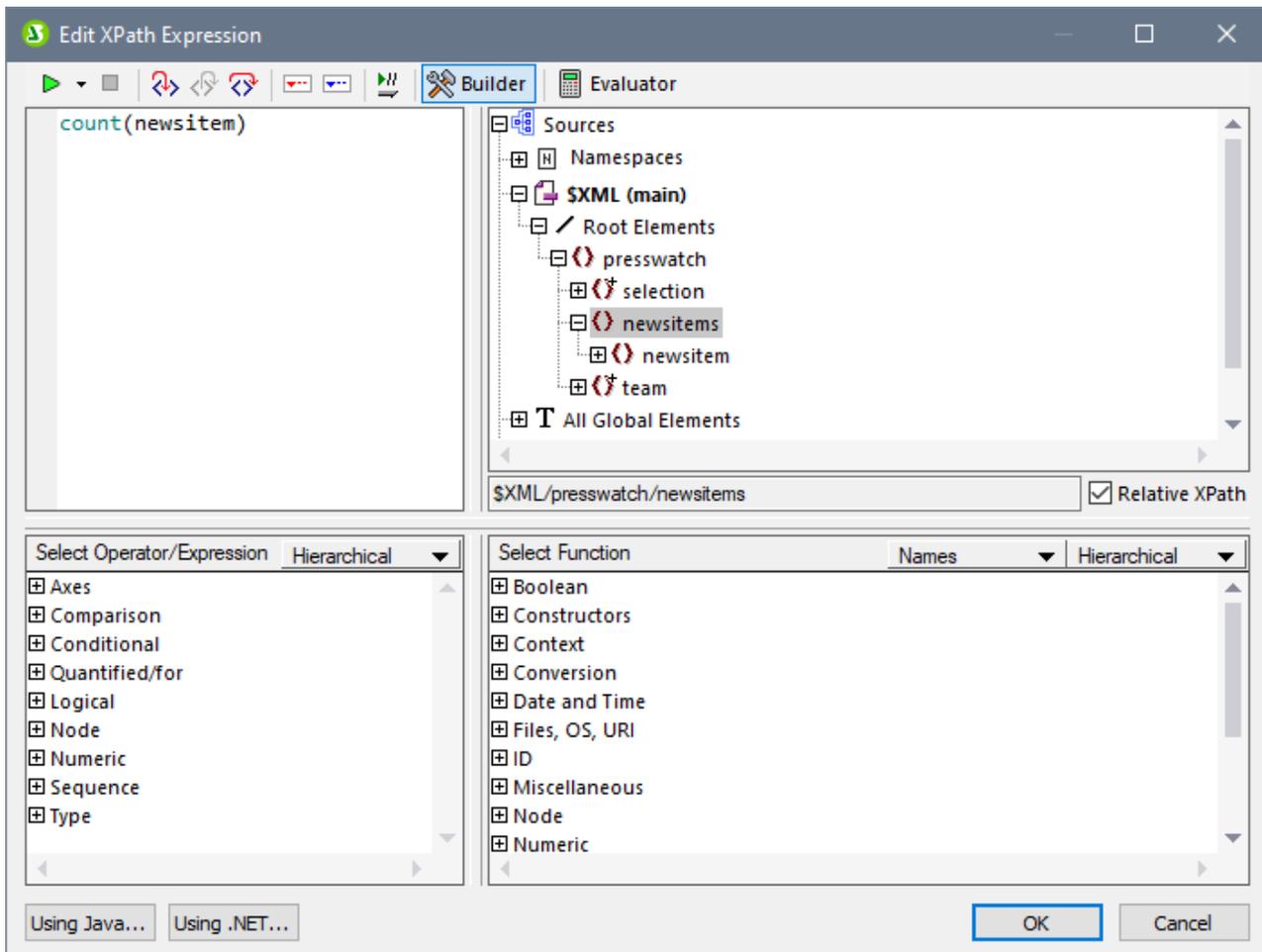
An **Auto-Calculation** uses an XPath expression to calculate a value. This value is displayed at the point where the Auto-Calculation is inserted. An Auto-Calculation can be inserted in the SPS as a text value, input field, or multiline input field. Place the cursor at the location where the Auto-Calculation 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](#)⁴¹⁸.



The value of the Auto-Calculation will be displayed accordingly in the output document.

The XPath expression for the Auto-Calculation

On selecting how the Auto-Calculation should be represented, the [Edit XPath Expression dialog](#)³⁹⁷ (*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.

11.7.5 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](#) ⁴¹⁸.

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.

11.7.6 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](#)⁴¹⁸ also pops up the Insert Image dialog.

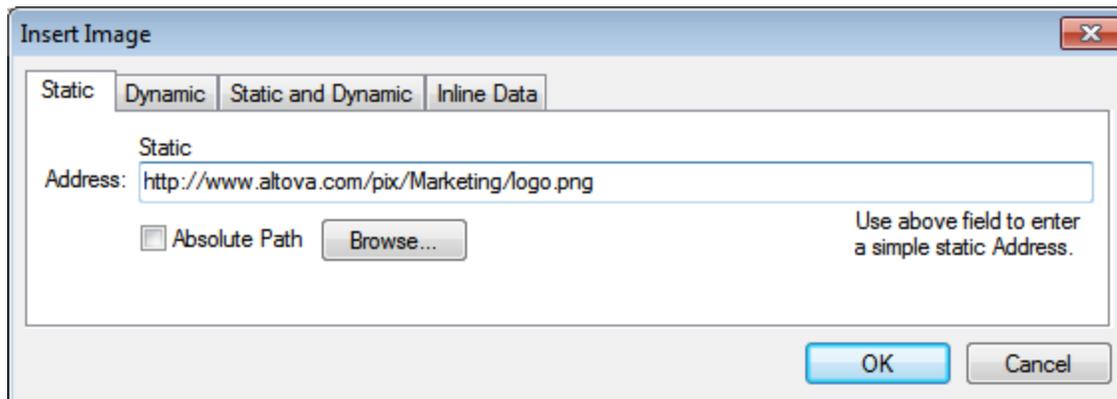
The Insert Image dialog has four tabs, each of which provides a different way to specify the image location. These are:

- *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

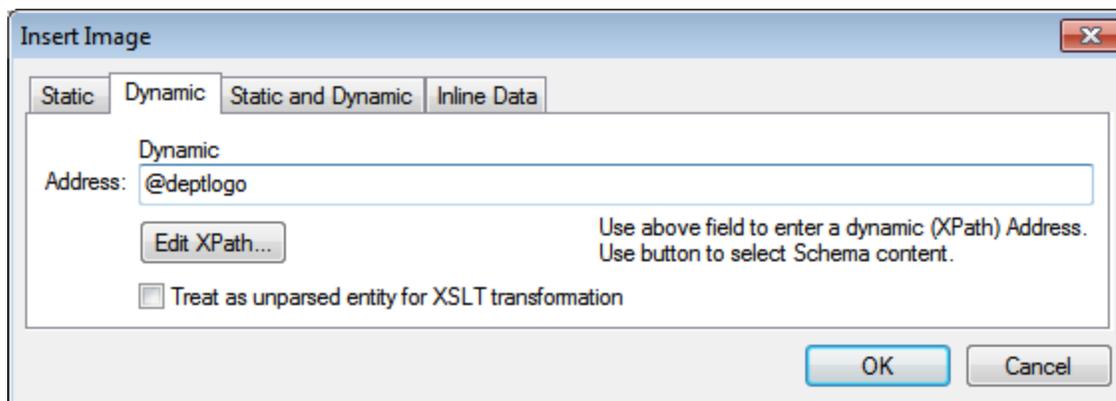
The image URI is entered directly in the *Address* field (see *screenshot below*). In the screenshot below the image URI is: `http://www.altova.com/pix/Marketing/logo.png`.



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.

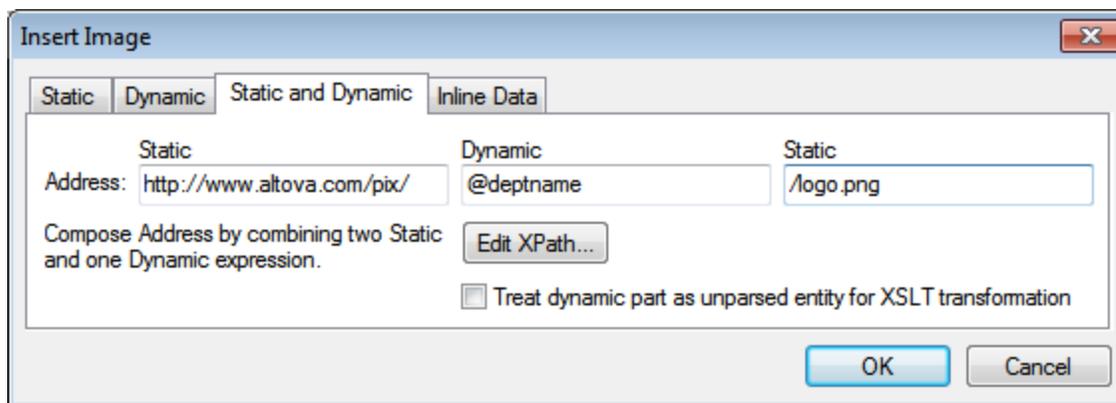


Click the **Edit XPath** button to pop up the [XPath Expression Builder](#)⁴⁰⁹. In the schema tree of the XPath Expression Builder, the context node will be highlighted.

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](#)³³⁸.

Static and Dynamic

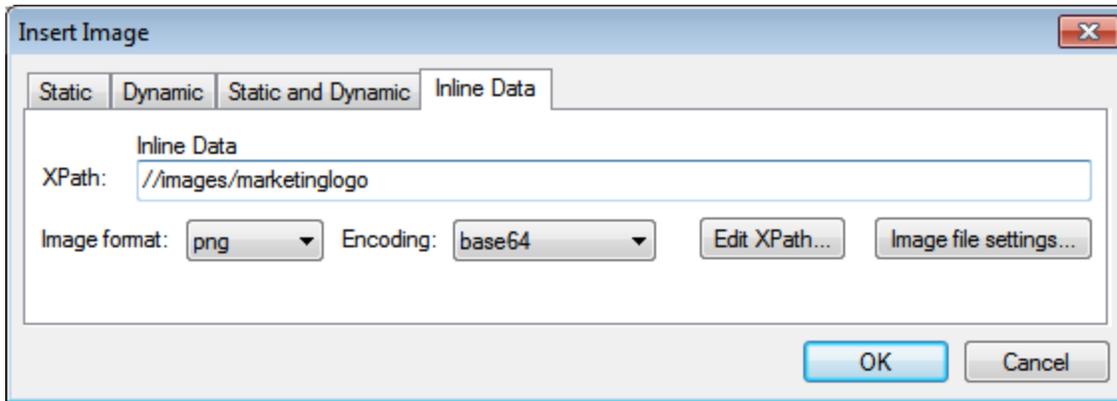
Use both the static and dynamic mechanisms together to generate the URI.



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](#)⁴⁰⁹ for the dynamic part.

Inline data

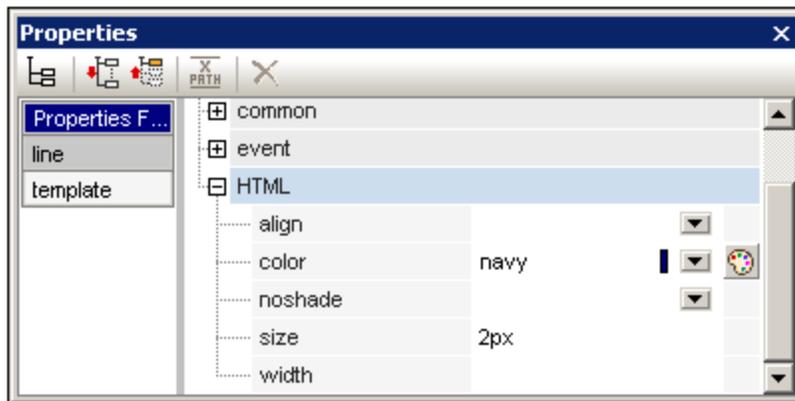
An image can be stored in an XML file as Base-16 or Base-64 encoded text. The XPath expression in the Insert Image dialog (see *screenshot below*) selects the node containing the encoded text. The Encoding combo box specifies the encoding used in the source XML so that StyleVision can correctly read 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.

11.7.7 Horizontal Line

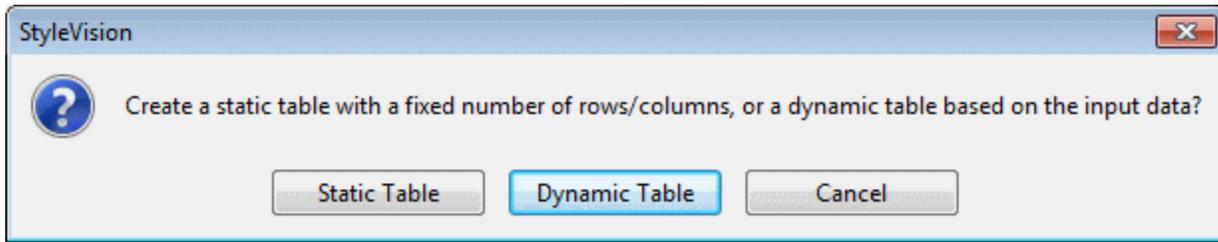
The **Horizontal Line** command inserts a horizontal line at the cursor insertion point. This command is not available when an SPS component is selected. To set properties for the horizontal line, select the line in the design, and in the Properties sidebar, select *line*, and specify values for properties in the *HTML* group (see *screenshot below*).



You can specify the following properties for the line: its *color*, *size* (thickness), *width* (in the design), *alignment*, and the *noshade* property.

11.7.8 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](#)¹²⁰ and [Dynamic SPS Tables](#)¹²¹.

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.

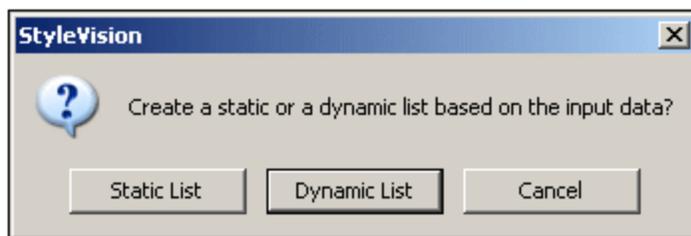
11.7.9 Bullets and Numbering



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.

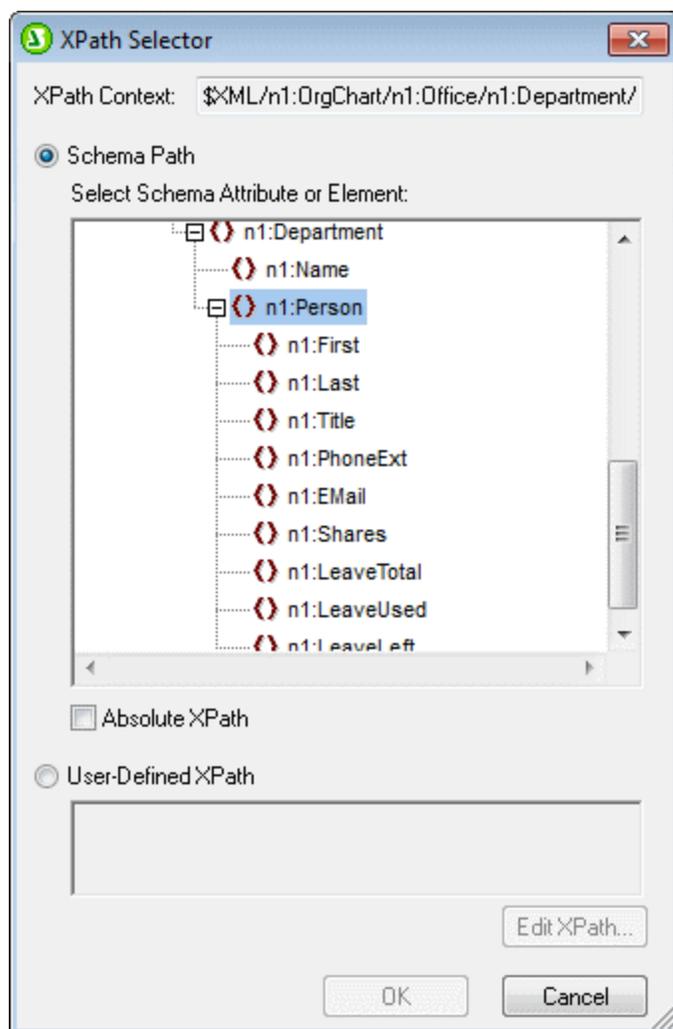
To create a list do the following:

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*).



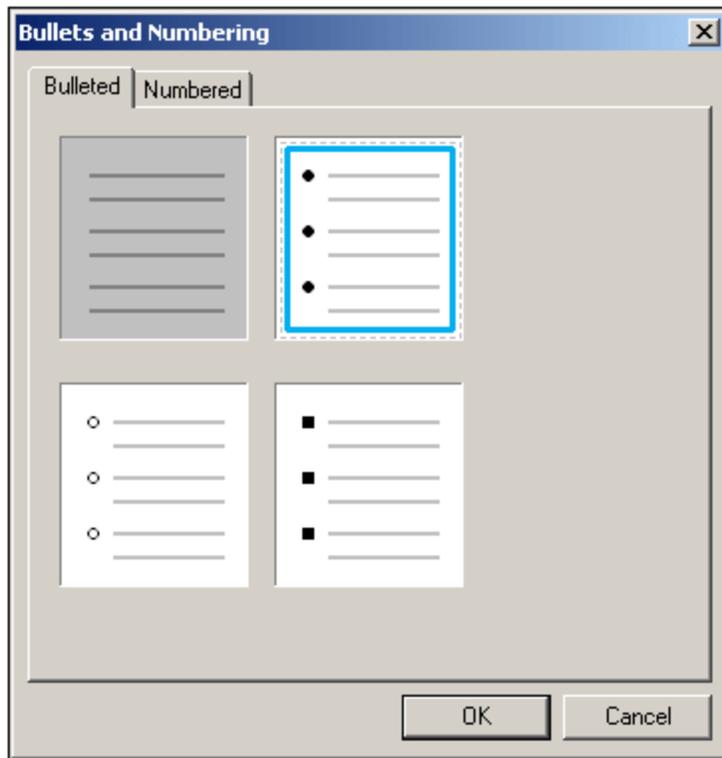
If you click **Static List**, the Bullets and Numbering dialog described in Step 3 pops up. If you click **Dynamic List**, the XPath Selector dialog pops up (*screenshot below*).

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.



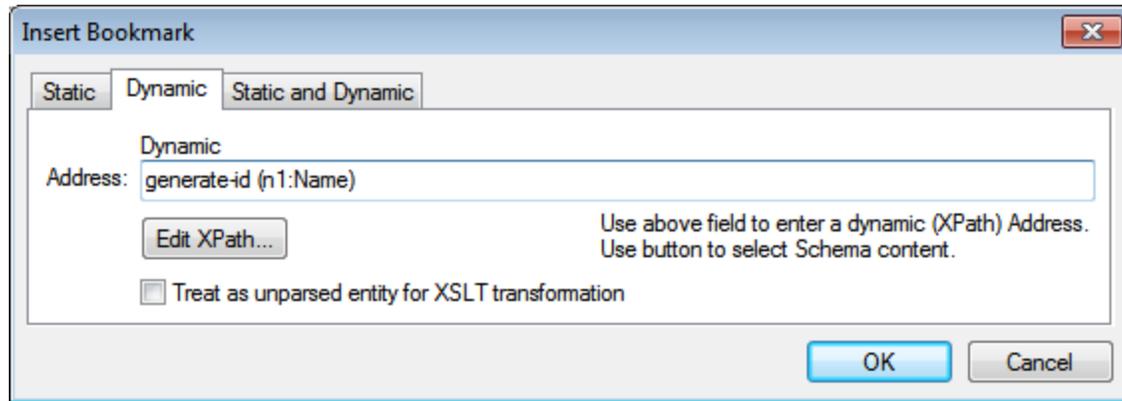
Note: A static list can also be created by placing the cursor at the location where the list is to be created and then clicking the Bulleted List icon or Numbered List icon in the [Insert Design Elements toolbar](#)⁴¹⁸ as required. A dynamic list can also be created by dragging a node from the Schema Tree into the design.

11.7.10 Bookmark

The **Bookmark** command allows you to insert a bookmark (or anchor) anywhere in the SPS. A bookmark can be referenced by a [Hyperlink](#)⁴⁶⁷.

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](#) ²⁹⁸, 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](#) ³⁰³. 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.

11.7.11 Hyperlink



The **Hyperlink** command enables you to insert a link from any part of the output document (HTML) to an anchor within the output document or to an external document or document fragment.

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](#)⁴¹⁸.
3. In the [Insert Hyperlink dialog](#)³⁰⁰ 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](#)³⁰³.
 - 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](#)³³⁸.

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.

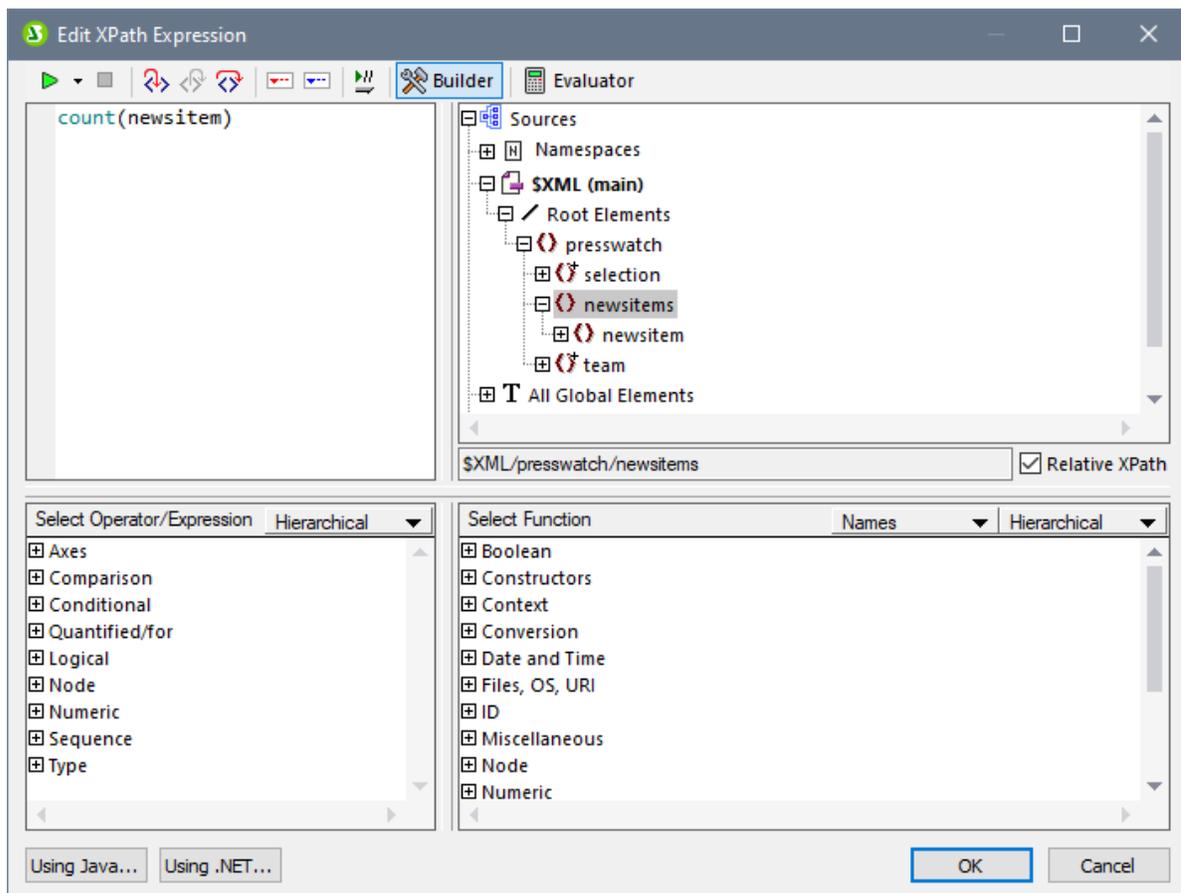
11.7.12 Condition, Output-Based Condition

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](#)³⁹⁷ 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.

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.

11.7.13 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.)



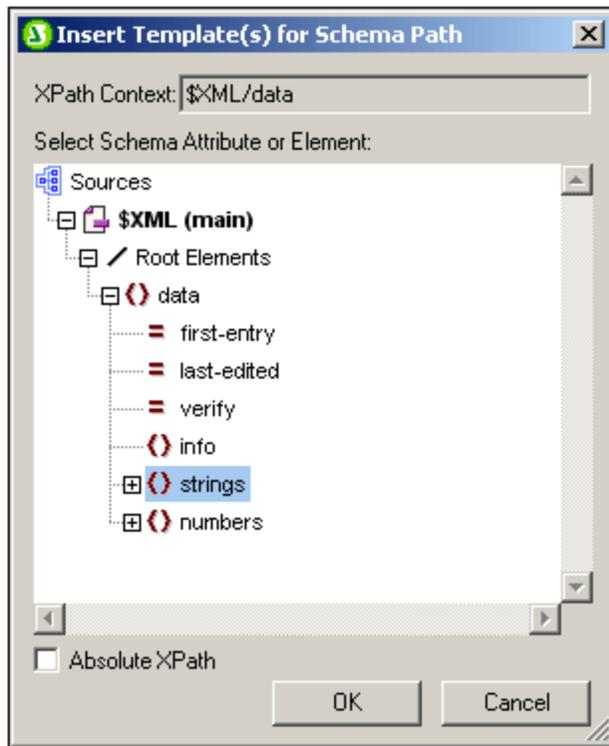
Content inside a Disabled component is ignored in the output. So you can add content that you want to ignore to the Disabled component. The Disabled component thus serves as a way to comment out content.

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**.

11.7.14 Template

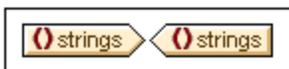
The **Template** command inserts, at the cursor insertion point, an empty template for the schema tree node you select. Insert a template as follows.

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).



11.7.15 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.

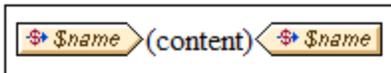
2. Click the **Insert | User-Defined Template** command. This pops up the [Edit XPath Expression dialog](#) ³⁹⁷.
3. Enter the XPath expression to select the node you want. There are a few points to note in this connection: (i) The XPath expression will be evaluated in the context of the node within which 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](#) ¹¹².

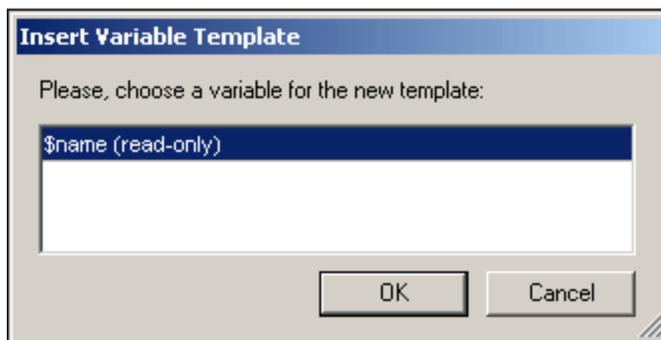
11.7.16 Variable Template

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](#) ²⁶³ defined in the SPS. Select the variable for which you wish to add a variable template.
4. Click **OK** to finish.

11.7.17 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.

11.7.18 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](#)⁴¹⁸. 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](#)¹⁵⁹.

11.7.19 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](#)²⁸¹
- [TOC Bookmark](#)²⁷⁸
- [TOC Bookmark \(Wizard\)](#)²⁷⁴
- [TOC Reference](#)²⁸⁴
- [TOC Reference | Entry Text / Leader / Page Reference](#)²⁸⁴
- [Hierarchical Numbering](#)²⁸⁴
- [Sequential Numbering](#)²⁸⁴
- [Level](#)²⁷⁵
- [Level Reference](#)²⁸³
- [Template Serves as Level](#)²⁷⁵

Note: These commands are also available as commands in a context menu, depending on where you right click in the design.

11.7.20 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](#)²³¹.

11.7.21 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](#)¹¹⁵ or a [User-Defined XML Text Block](#)¹¹⁶. How to use these two components is described in the section [SPS File: Content | User-Defined Elements, XML Text Blocks](#)¹¹⁵.

11.8 Enclose With Menu

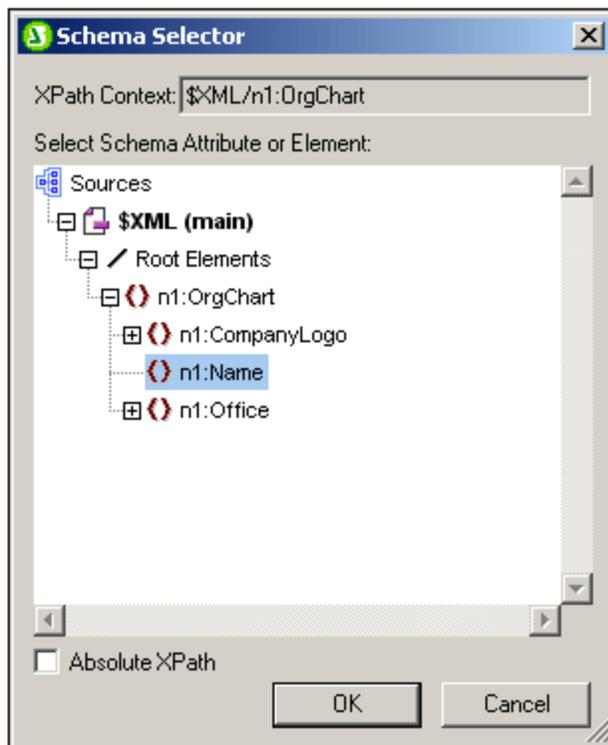
The **Enclose with** menu provides commands enabling you to enclose a selection in the design with a variety of design components. Some of these commands are available as [toolbar icons](#)⁴¹⁴ that enable you to insert the component in the design (equivalent commands are available in the [Insert menu](#)⁴⁵⁷). 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.

11.8.1 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.

11.8.2 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](#)³⁹⁷ 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](#)²¹⁹.

11.8.3 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](#)²²².
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](#)²²².

11.8.4 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](#)⁴¹⁸.

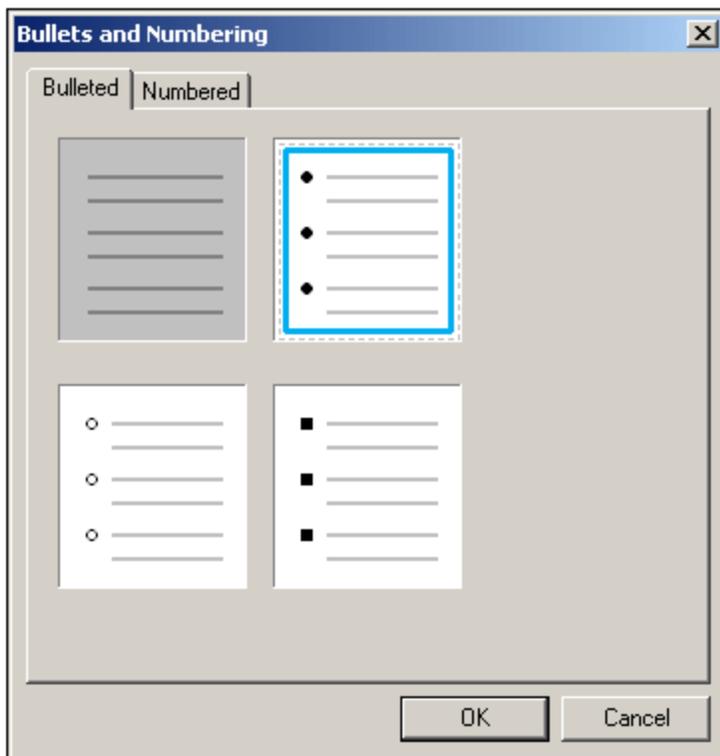
The **Special Paragraph** command allows you to assign a predefined format to the selected node. The available predefined formats can also be selected from the combo box in the toolbar.

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.

11.8.5 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](#)⁴¹⁸.

11.8.6 Bookmarks and Hyperlinks

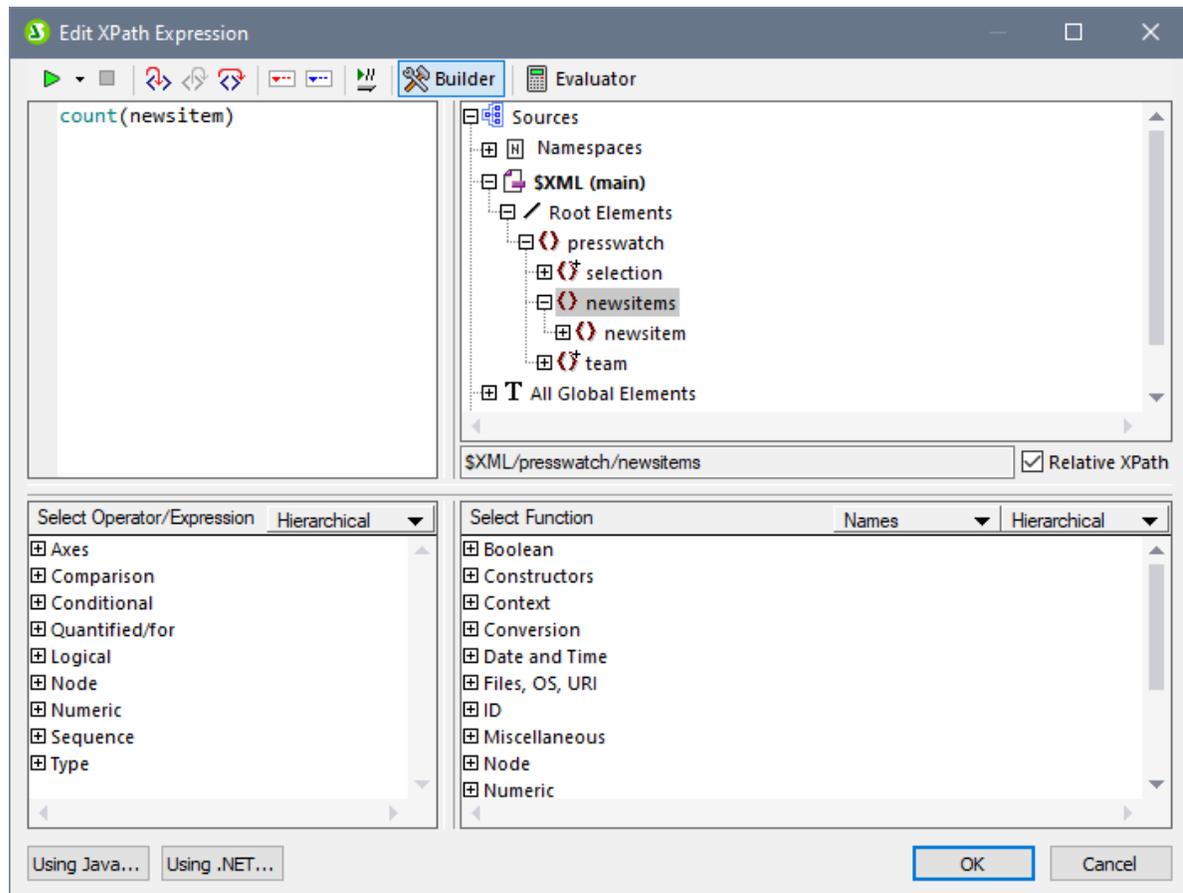
The **Enclose with | Bookmark** and **Enclose With | Hyperlink** commands are enabled when some text or component in the SPS design is selected. These commands enable a bookmark and hyperlink, respectively, to be created around the selection. For more information about how bookmarks and hyperlinks work and how to create them, see the section [Advanced Features | Table of Contents, Referencing, Bookmarks](#)²⁹⁸.

11.8.7 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](#)³⁹⁷ 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.

Editing the XPath expressions of branches

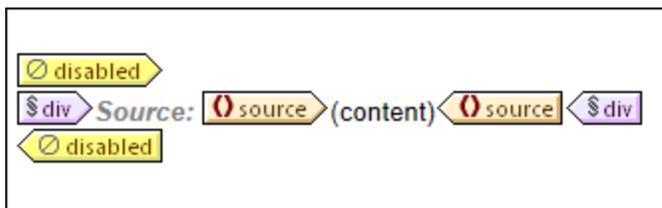
To edit the XPath expression of a branch, select the branch in Design View. Then, in the Properties sidebar, select `condition branch | when`. Click the **Edit** button  for the XPath item. This pops up the Edit XPath Expression dialog (*screenshot above*), in which you can edit the expression. Click **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.

11.8.8 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**.

11.8.9 TOC Bookmarks and TOC Levels

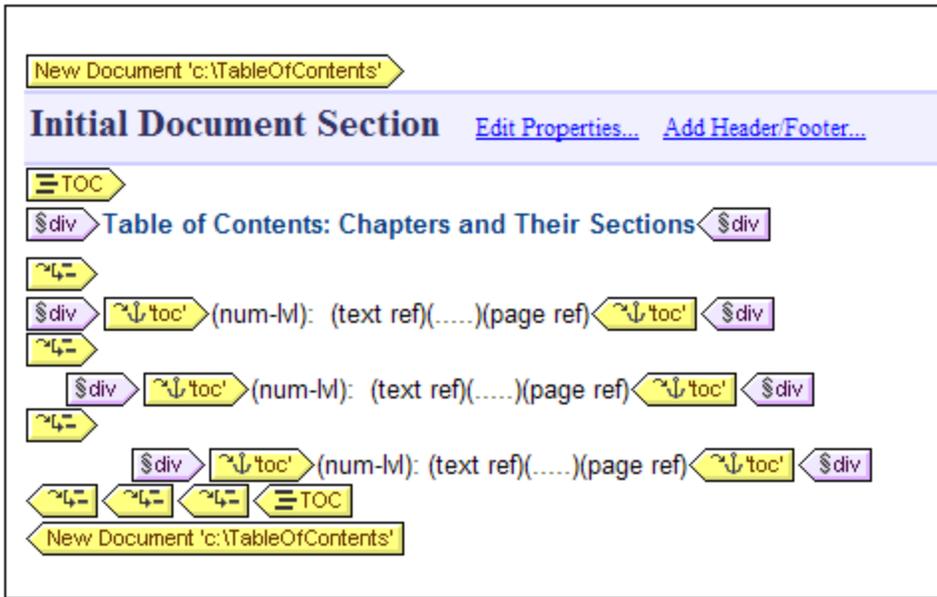
When a component in the design is selected, it can be enclosed with one or more relevant Table of Contents (TOC) components. The list of TOC commands is as follows. For the details of how to use them click on the respective links, which will take you to the section on how to use that particular TOC component.

- [TOC Bookmark](#)²⁷⁸
- [TOC Bookmark \(Wizard\)](#)²⁷⁴
- [Level](#)²⁷⁵
- [Level Reference](#)²⁸³

Note: These commands are also available as commands in a context menu, depending on where you right click in the design.

11.8.10 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](#)²³¹.

11.8.11 User-Defined Element

The **Enclose with | User-Defined Element** command creates a [User-Defined Element](#)¹¹⁵ around the selection in the design. How to use user-defined elements is described in the section [SPS File: Content | User-Defined Elements](#)¹¹⁵.

11.9 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](#)¹²⁸ and [HTML properties for tables and its sub-components](#)¹²⁸.

The Table commands are available in the **Table** menu (see *list below*) and as icons in the [Table toolbar](#)⁴¹⁷. 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](#)⁴⁸² 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](#)¹²⁸ and/or [Properties sidebar](#)¹²⁸, define the required property for that table component.

The following commands are available in the Table menu:

- [Insert Table, Delete Table](#)⁴⁸²
- [Add Table Headers, Footers](#)⁴⁸³
- [Append/Insert Row/Column](#)⁴⁸³
- [Delete Row, Column](#)⁴⁸⁴
- [Join Cell Left, Right, Below, Above](#)⁴⁸⁴
- [Split Cell Horizontally, Vertically](#)⁴⁸⁴
- [View Cell Bounds, Table Markup](#)⁴⁸⁵
- [Table Properties](#)⁴⁸⁵
- [Vertical Alignment of Cell Content](#)⁴⁸⁶

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.

11.9.1 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](#) ¹²¹.

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.

11.9.2 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.

11.9.3 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.

11.9.4 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.

11.9.5 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.

11.9.6 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.

11.9.7 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.

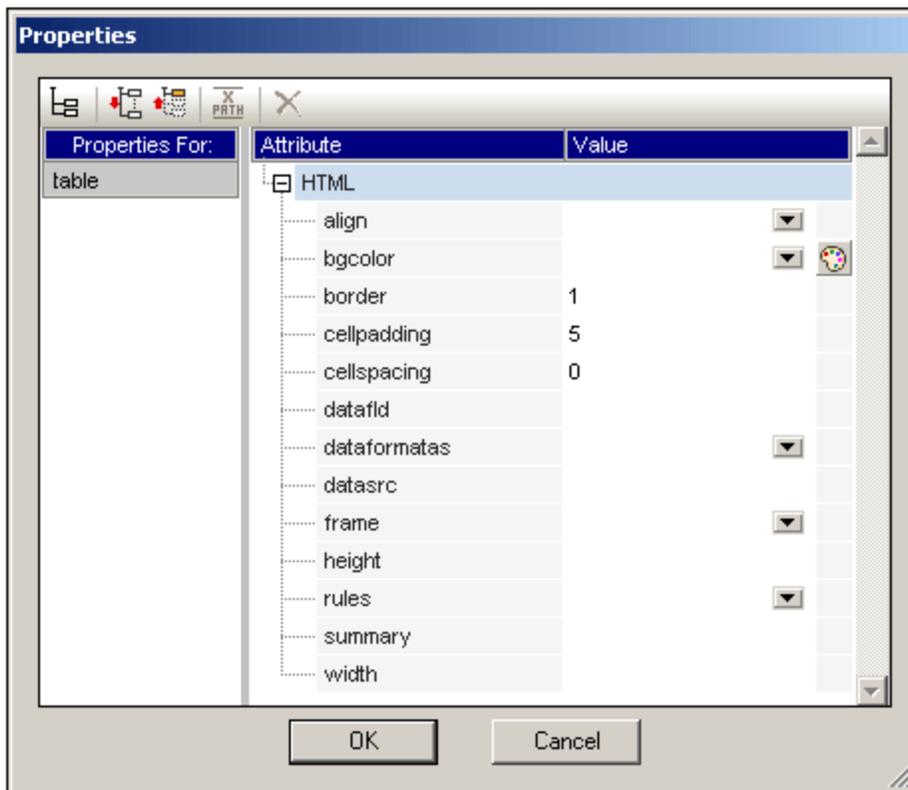


The **View Table Markup** command toggles the display of the blue column and row markers on and off.

11.9.8 Table Properties



The **Table Properties** command is enabled when the cursor is placed inside a [static or dynamic table](#)¹¹⁸. Clicking the command, pops up the Properties sidebar, with the *Table* component selected (*screenshot below*).

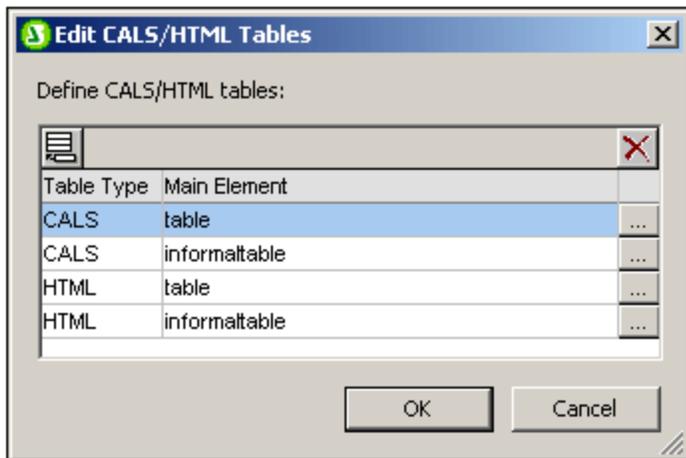


You can now edit the properties of the table. Click **OK** when done.

11.9.9 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](#) ¹¹⁸.

11.9.10 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.

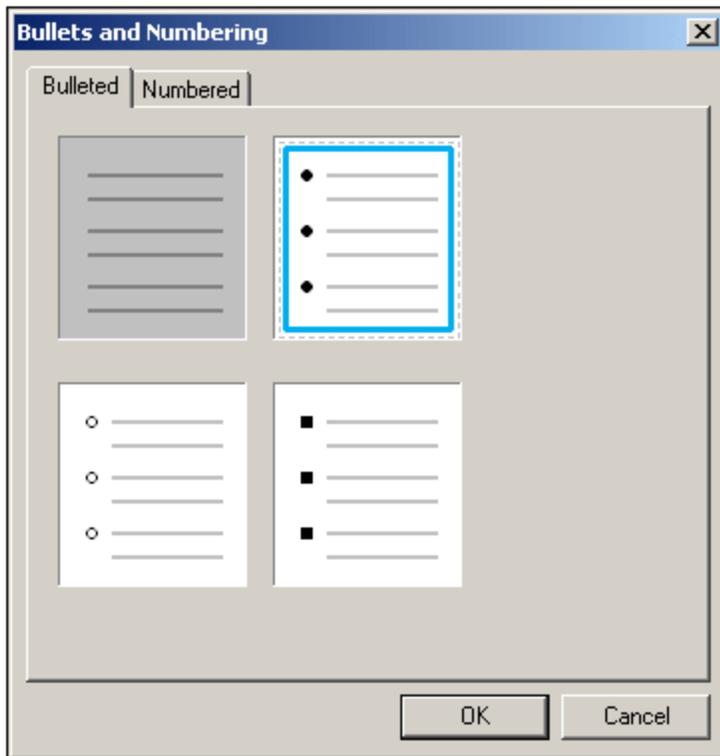
11.10 Properties Menu

The **Properties** menu contains commands that enable you to insert lists and define datatype formats for the [input formatting](#)³¹⁰ feature. The description of the commands is organized into the following sub-sections:

- [Bullets and Numbering](#)⁴⁸⁷ command, to insert lists.
- [Predefined Format Strings](#)⁴⁸⁷ command, to define numeric datatype formats for a given SPS.

11.10.1 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.



11.10.2 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](#)³¹⁰ dialog. In the Value Formatting dialog, you can either create a format directly or select from a drop-down list of predefined formats.

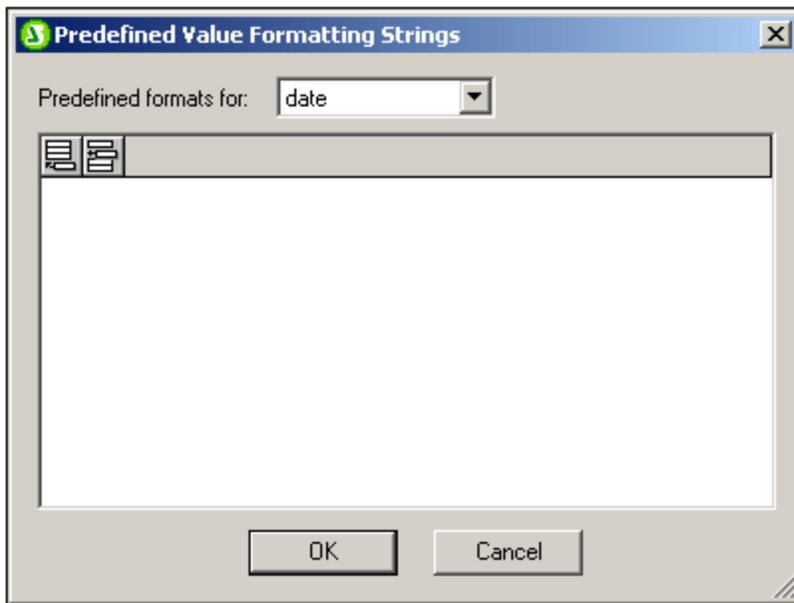
The predefined formats that are available in the dropdown list are of two types:

- 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.

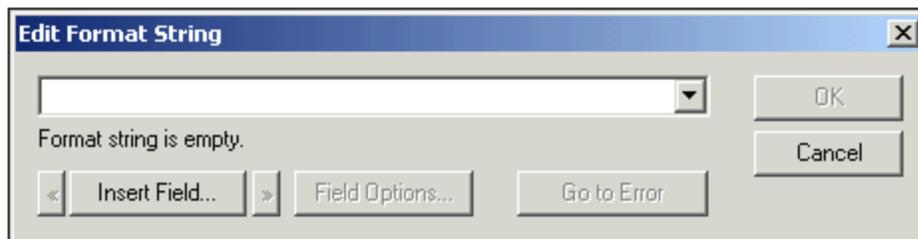
Creating a predefined value formatting string

A predefined value format string is specific to a datatype. To create a predefined value formatting string, do the following:

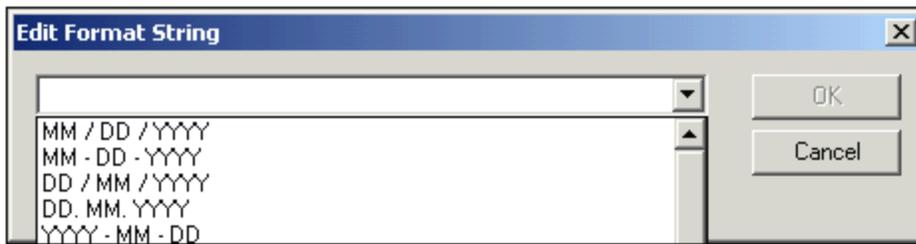
1. Click **Properties | Predefined Value Formatting Strings**. The following dialog appears:



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](#)³¹⁰. 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.

11.11 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:

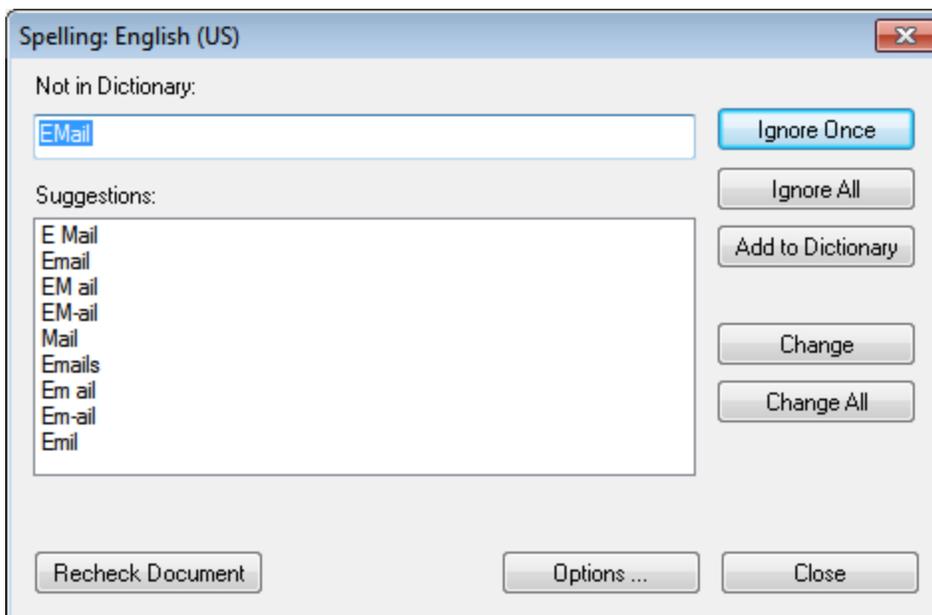
- [Spelling](#) ⁴⁹⁰
- [Spelling Options](#) ⁴⁹¹
- [XML Schema Manager](#) ¹⁸²
- [Customize](#) ⁵⁰⁹

11.11.1 Spelling

The **Spelling** command runs a spelling check on the SPS (in Design View). 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.

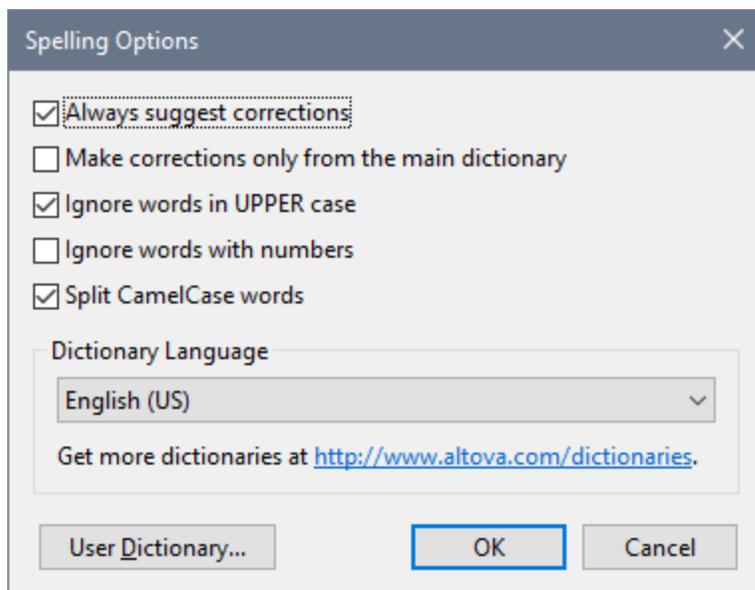


You can then select an entry from the list in the Suggestions pane and click **Change** or **Change All** to change the highlighted instance of this spelling or all its instances, respectively. (Double-clicking a word in the Suggestions list causes it to replace the unknown word.) Alternatively, you 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](#)⁴⁹¹ dialog, in which you can specify options for the spelling check.

11.11.2 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](#).

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

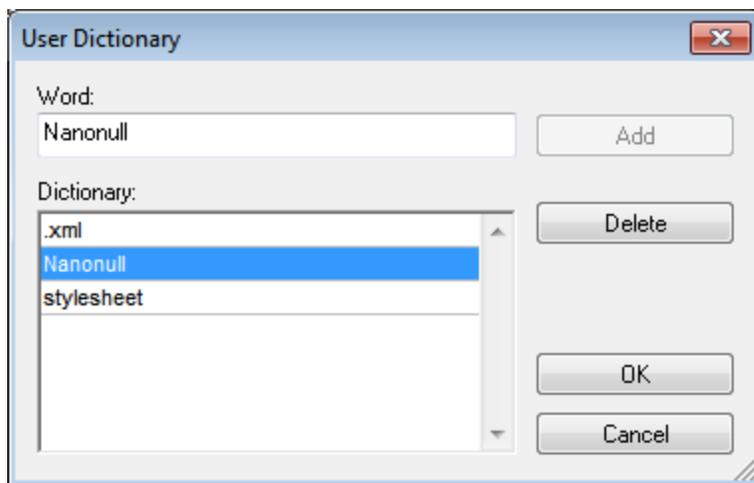
dictionaries must be done with administrator rights, otherwise installation will fail with an error.



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.

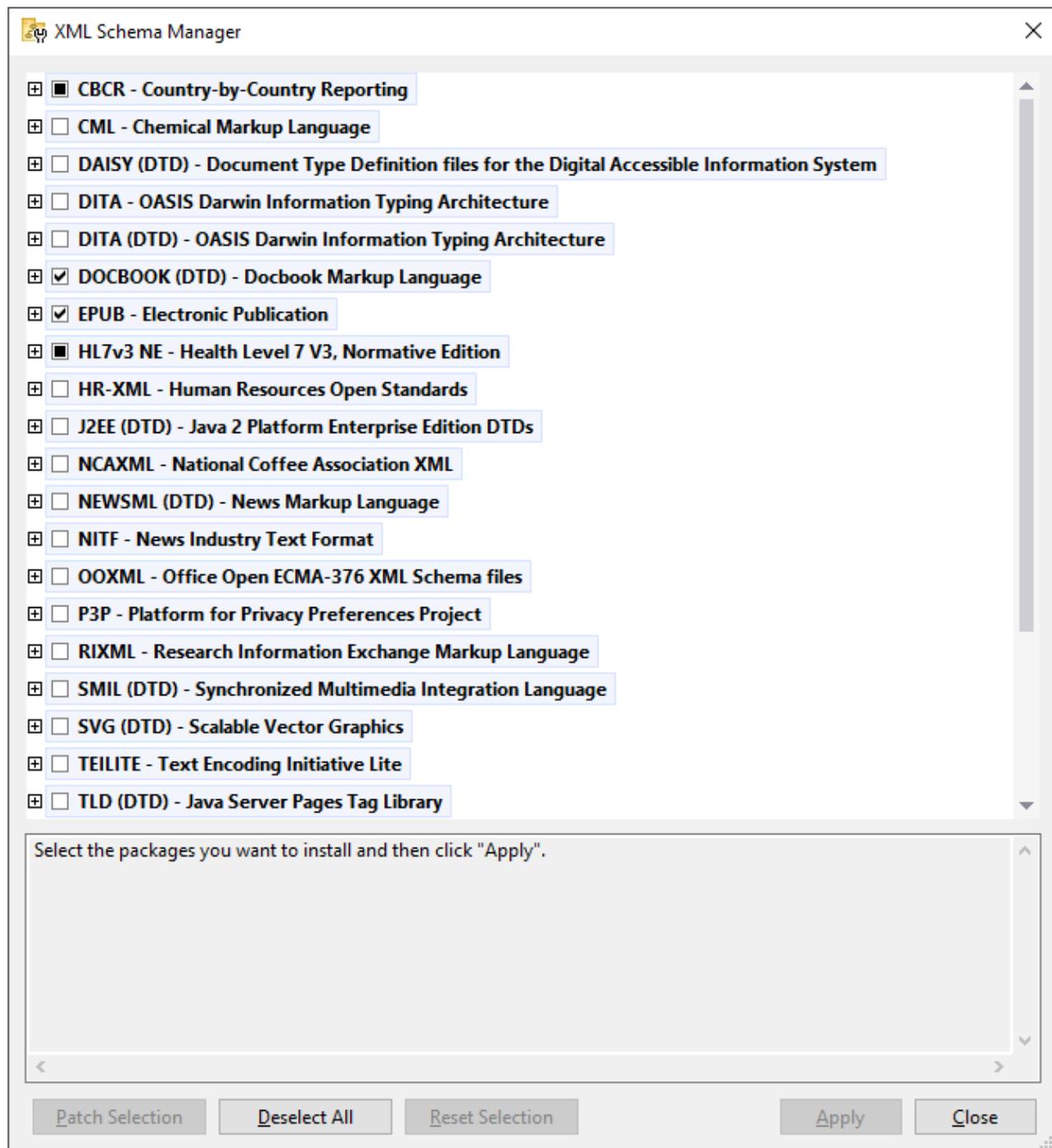
Words may also be added to the User Dictionary during a spelling check. If an unknown word is encountered during a spelling check, then the [Spelling dialog](#)⁴⁹⁰ pops up prompting you for the action you wish to take. If you click the **Add to Dictionary** button, then the unknown word is added to the user dictionary.

The user dictionary is located at: C:\Users\\Documents\Altova\SpellChecker\Lexicons\user.dic

11.11.3 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 XBRL-enabled 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 Enterprise Edition 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.

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`¹⁹³ 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:

Windows	C:\ProgramData\Altova\pkgs\.cache
---------	-----------------------------------

<i>Linux</i>	<code>/var/opt/Altova/pkgcache</code>
<i>macOS</i>	<code>/var/Altova/pkgcache</code>

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](#)¹⁹⁶ 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](#)¹⁹⁴ command, and (ii) run the [initialize](#)¹⁹² command. (Alternatively, run the `reset` command with the `--i` option.)

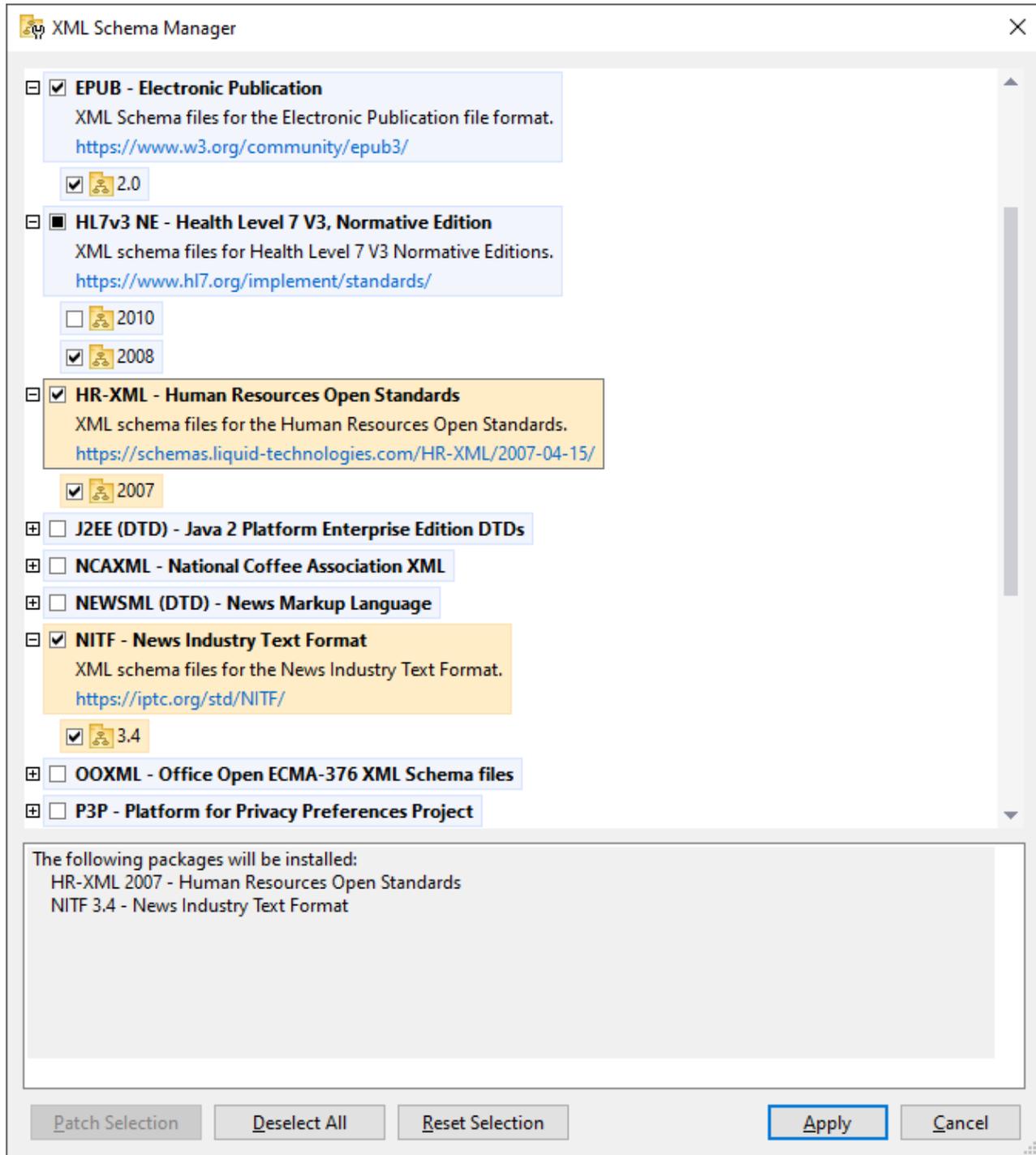
11.11.3.1 Run Schema Manager

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](#)¹⁹¹.

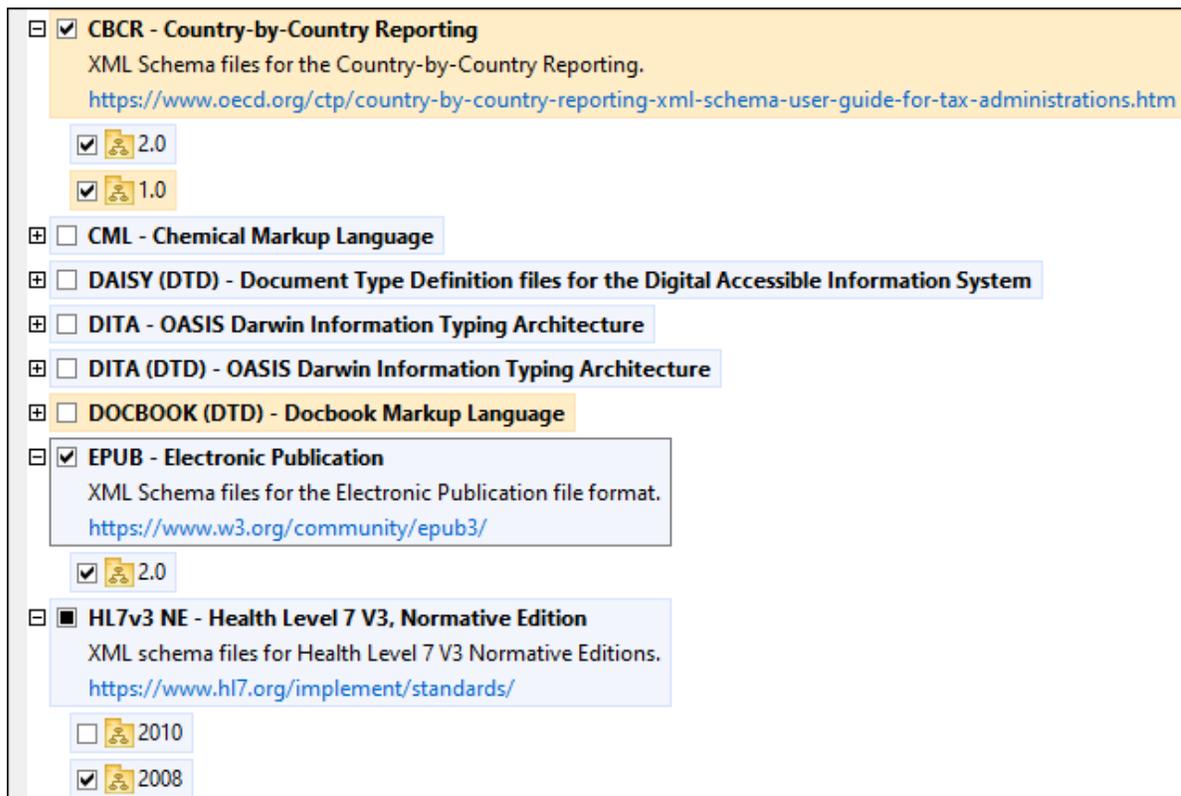
To display help for the commands, run the following:

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

11.11.3.2 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`

11.11.3.3 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](#)¹⁸⁷.) 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`¹⁹³ command. This lists any schemas for which upgrades are available.
2. Run the `upgrade`¹⁹⁶ 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`¹⁹³ 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`¹⁹³ command.) You can enter as many schemas as you like. For details, see the description of the `install`¹⁹³ 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**.

11.11.3.4 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`¹⁹⁵ 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`¹⁹³ command.) You can enter as many schemas as you like. For details, see the description of the `uninstall`¹⁹⁵ 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`¹⁹⁴ command.

After running this command, make sure to run the `initialize` command in order to recreate the cache directory. Alternatively, run the `reset` command with the `-i` option.

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

11.11.3.5 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.

11.11.3.5.1 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
```

11.11.3.5.2 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](#)¹⁹³ 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
```

11.11.3.5.3 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
```

11.11.3.5.4 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](#)¹⁹³ 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](#)¹⁸².

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
```

11.11.3.5.5 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`

11.11.3.5.6 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](#)¹⁹² 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](#)¹⁹⁶ 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`

11.11.3.5.7 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`¹⁹³ 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](#)¹⁸².

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
```

11.11.3.5.8 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](#)¹⁹⁴ and [initialize](#)¹⁹².

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
```

11.11.3.5.9 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](#)¹⁹³ 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> .

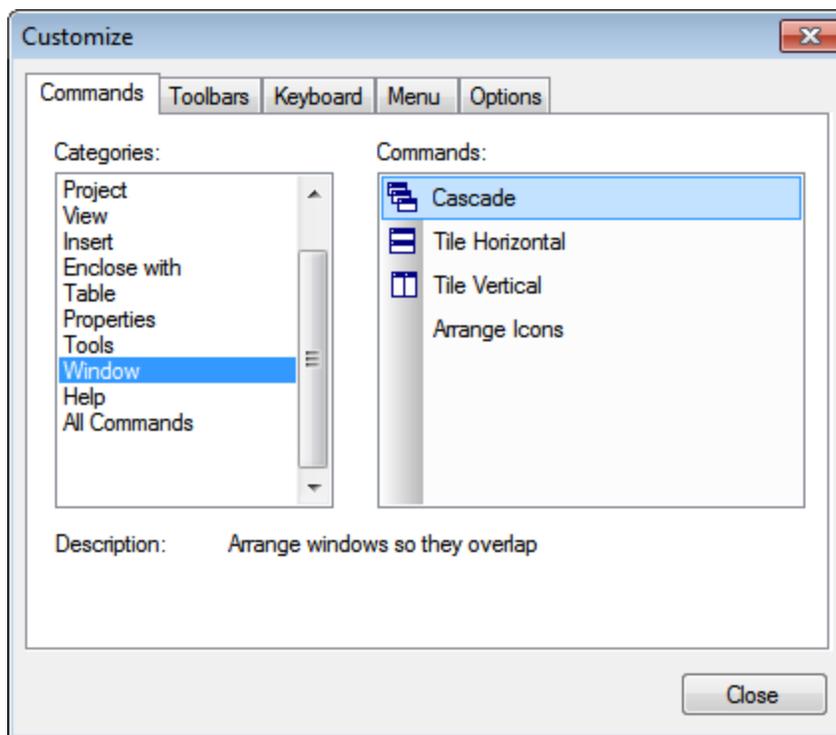
--help, --h Display help for the command.

11.11.4 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**.

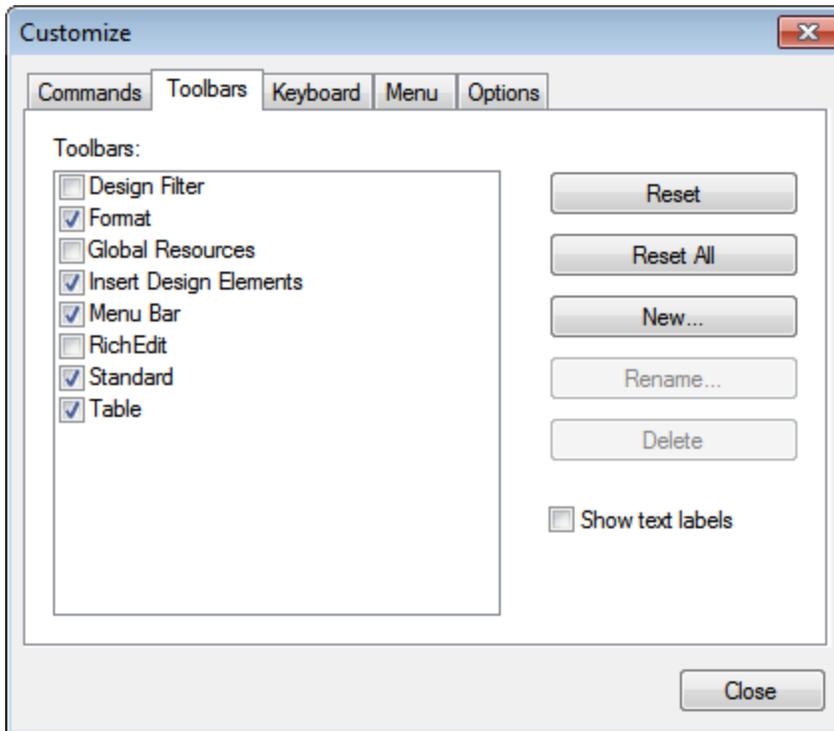
Note:

- 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 (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.

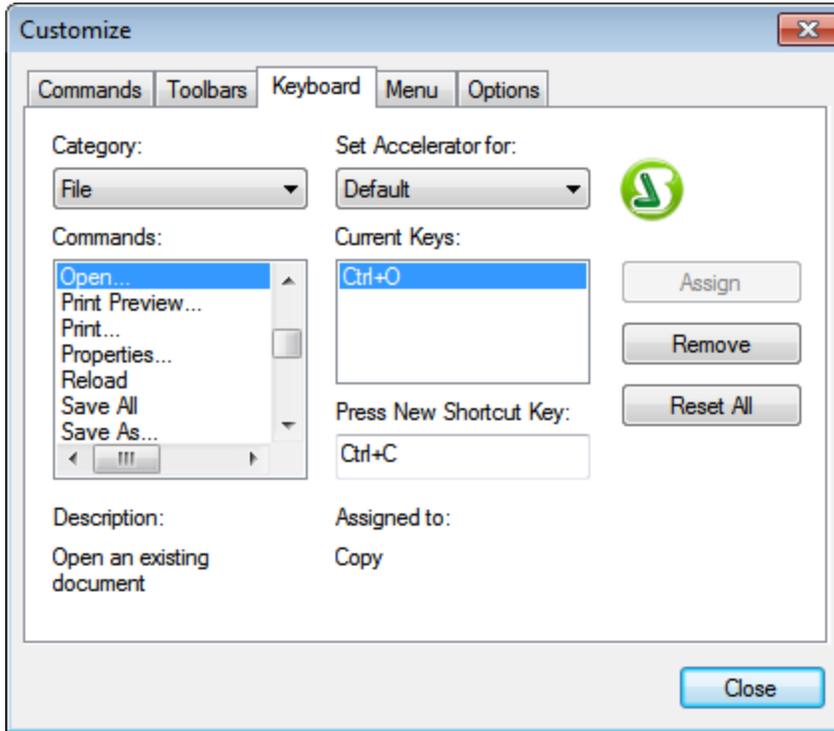
In the Toolbars tab of the Customize dialog, you can toggle a toolbar on and off by clicking in its checkbox. When a toolbar is selected (in the Toolbars tab), you can cause the text labels of that toolbar's items to be displayed by clicking the **Show text labels** check box. You can also reset a selected 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

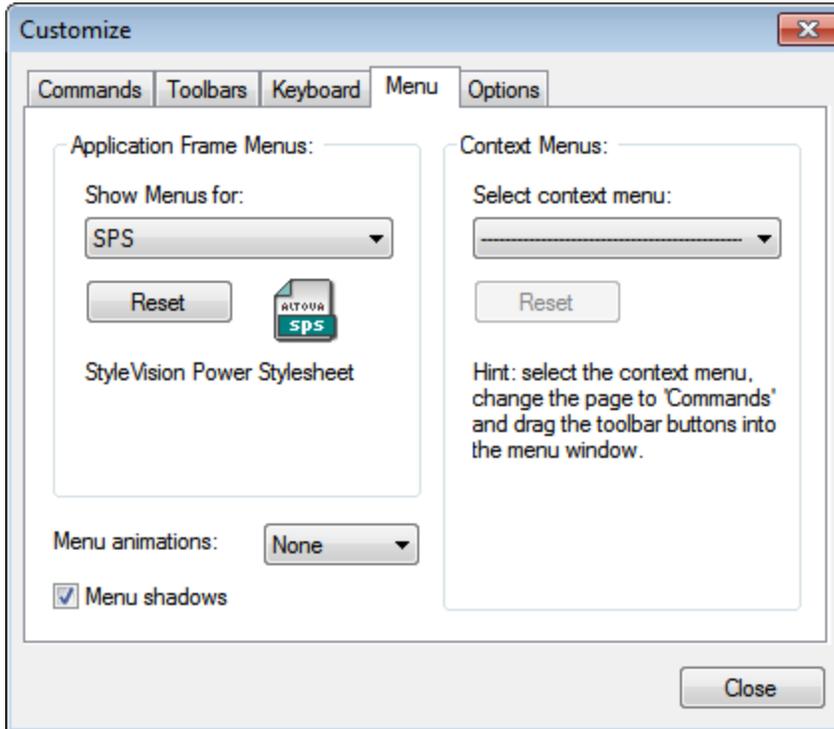
1. Click the **Reset All** button to go back to the original, installation-time shortcuts. A dialog box appears prompting you to confirm whether you want to reset all keyboard assignments.
2. Click **Yes** if you want to reset all keyboard assignments.

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.

11.11.5 Restore Toolbars and Windows

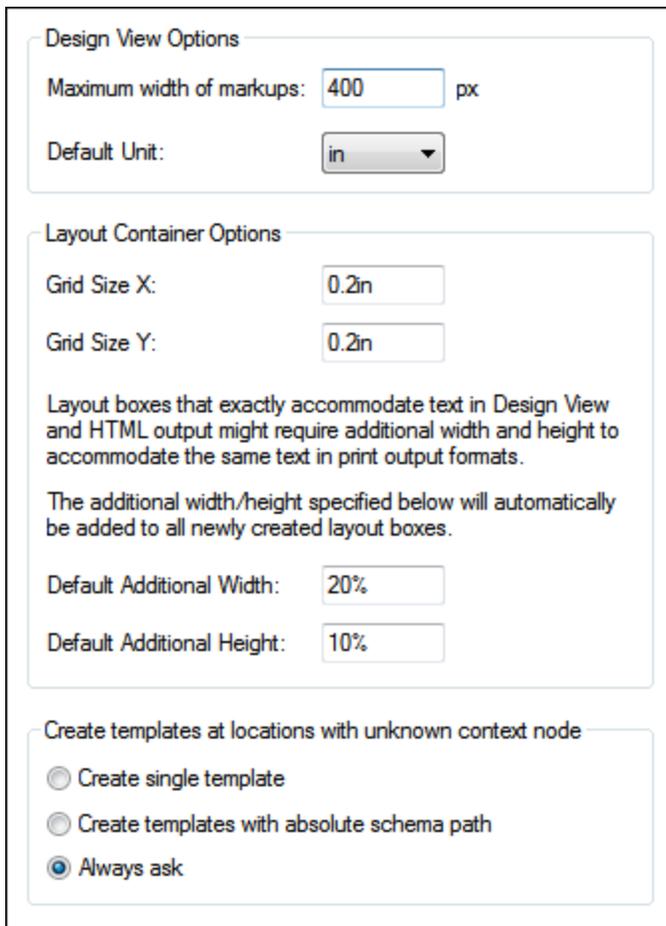
This command restores toolbars, windows, entry helpers and other GUI components to their default state. You will need to restart StyleVision for the changes to take effect.

11.11.6 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*.

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.

Default encoding

To set the default encoding of the output HTML file, open the dropdown menu of the combo box and select the desired option from the list of encoding options, and click **OK**. Every new SPS you create from this point on, will set the HTML output encoding 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" />
```

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](#)⁴⁴³ command.

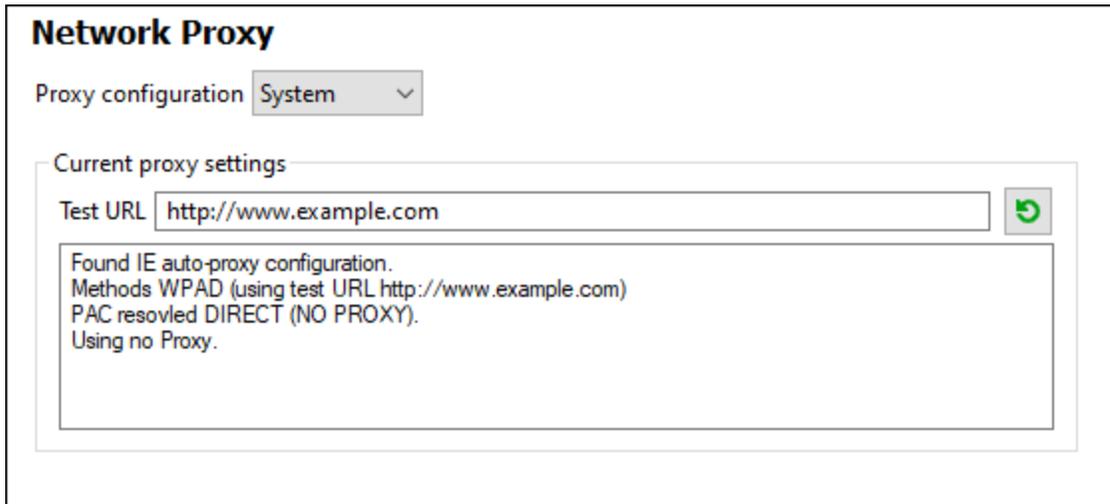
XSL options

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 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 `<1oca1>` 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

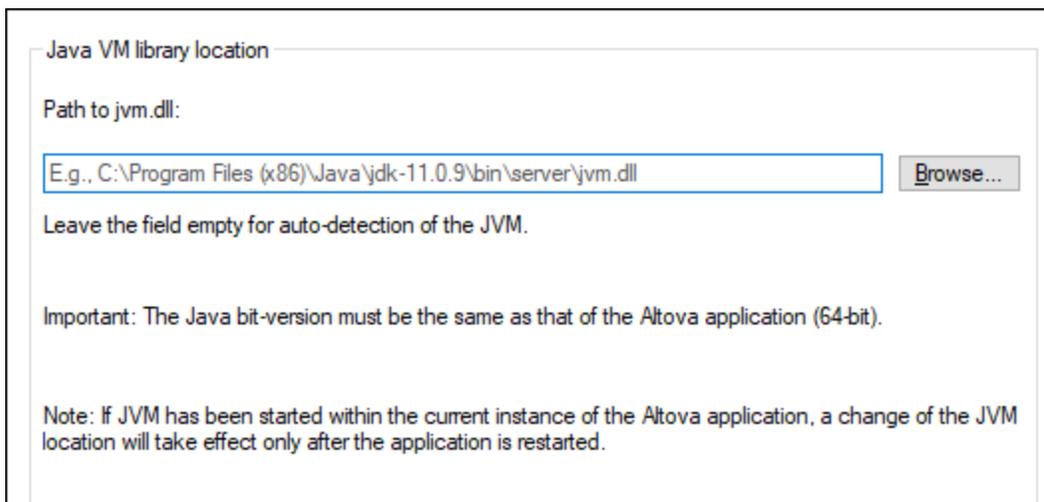
Provides a verbose log of the proxy detection. It can be refreshed with the **Refresh** button to the right of the *Test URL* field (for example, when changing the test URL, or when the proxy settings have been changed).

- *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.

11.12 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.

11.13 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](#)⁵²⁰, which lets you enter your license key-code once you have purchased the product.

The description of the Help menu commands is organized into the following sub-sections:

- [Table of Contents, Index, Search](#)⁵²⁰
- [Activation, Order Form, Registration, Updates](#)⁵²⁰
- [Other Commands](#)⁵²⁴

11.13.1 Table of Contents, Index, Search

☐ Table of Contents

Opens the onscreen help manual of StyleVision with the Table of Contents displayed in the left-hand-side pane of the Help window. The Table of Contents provides an overview of the entire Help document. Clicking an entry in the Table of Contents takes you to that topic.

☐ Index

Opens the onscreen help manual of StyleVision with the Keyword Index displayed in the left-hand-side pane of the Help window. The index lists keywords and lets you navigate to a topic by double-clicking the keyword. If a keyword is linked to more than one topic, a list of these topics is displayed.

☐ Search

Opens the onscreen help manual of StyleVision with the Search dialog displayed in the left-hand-side pane of the Help window. To search for a term, enter the term in the input field and press Enter or List Topics. The Help system performs a full-text search on the entire Help documentation and returns a list of hits. Double-click any item to display that item.

11.13.2 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. Enter your name, company, and e-mail address in the dialog and click **Request**. A license file is sent to the e-mail address you entered and should reach you in a few minutes. Save the license file to a suitable location.

When you clicked **Request**, an entry field appeared at the bottom of the Request dialog. This field takes the path to the license file. Browse for or enter the path to the license file and click **OK**. (In the **Software Activation** dialog, you can also click **Upload a New License** to access a dialog in which the path to the license file is entered.) The software will be unlocked for a period of 30 days.

- **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.

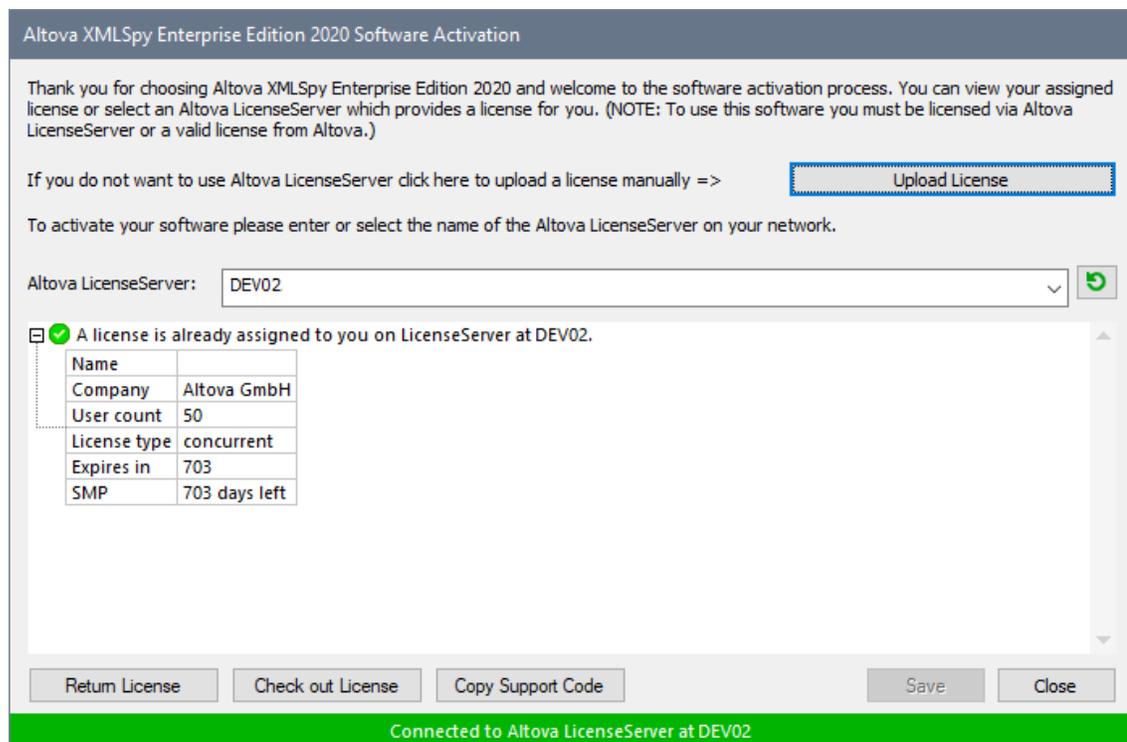
Activate your software

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.

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.

11.13.3 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.

FAQ on the Web

A link to Altova's FAQ database on the Internet. The FAQ database is constantly updated as Altova support staff encounter new issues raised by customers.

Download Components and Free Tools

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.

12 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](#) ⁵²⁶

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

[Technical Data](#) ⁶³²

Provides technical information about StyleVision.

- OS and memory requirements
- Altova XML Parser
- Altova XSLT and XQuery Engines
- Unicode support
- Internet usage

[License Information](#) ⁶³⁴

Contains information about the way StyleVision is distributed and about its licensing.

- Electronic software distribution
- License metering
- Copyright
- End User License Agreement

12.1 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 `E1` in a path expression `E1/E2` does not evaluate to a sequence of nodes.

If you encounter this kind of error, modify either the XSLT/XQuery document or the instance document as appropriate.

This section describes implementation-specific features of the engines, organized by specification:

- [XSLT 1.0](#) ⁵²⁶
- [XSLT 2.0](#) ⁵²⁶
- [XSLT 3.0](#) ⁵²⁸
- [XQuery 1.0](#) ⁵²⁸
- [XQuery 3.1](#) ⁵³¹

12.1.1 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, ` `.

12.1.2 XSLT 2.0

This section:

- [Engine conformance](#) ⁵²⁷
- [Backward compatibility](#) ⁵²⁷
- [Namespaces](#) ⁵²⁷
- [Schema awareness](#) ⁵²⁷
- [Implementation-specific behavior](#) ⁵²⁸

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. The only time the backwards compatibility of the XSLT 2.0 engine comes into effect is when using the XSLT 2.0 engine to process an XSLT 1.0 stylesheet. Note that there could be differences in the outputs produced by the XSLT 1.0 Engine and the backwards-compatible XSLT 2.0 engine.

Namespaces

Your XSLT 2.0 stylesheet should declare the following namespaces in order for you to be able to use the type constructors and functions available in XSLT 2.0. The prefixes given below are conventionally used; you could use alternative prefixes if you wish.

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 the behavior of certain XSLT 2.0 functions.

xsl:result-document

Additionally supported encodings are (the Altova-specific): `x-base16tobinary` and `x-base64tobinary`.

function-available

The function tests for the availability of in-scope functions (XSLT, XPath, and extension functions).

unparsed-text

The `href` attribute accepts (i) relative paths for files in the base-uri folder, and (ii) absolute paths with or without the `file://` protocol. Additionally supported encodings are (the Altova-specific): `x-binarytobase16` and `x-binarytobase64`.

unparsed-text-available

The `href` attribute accepts (i) relative paths for files in the base-uri folder, and (ii) absolute paths with or without the `file://` protocol. Additionally supported encodings are (the Altova-specific): `x-binarytobase16` and `x-binarytobase64`.

Note: The following encoding values, which were implemented in earlier versions of RaptorXML's predecessor product, AltovaXML, are now deprecated: `base16tobinary`, `base64tobinary`, `binarytobase16` and `binarytobase64`.

12.1.3 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](#)⁵²⁶. 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.

12.1.4 XQuery 1.0

This section:

- [Engine conformance](#)⁵²⁹
- [Schema awareness](#)⁵²⁹

- [Encoding](#) ⁵²⁹
- [Namespaces](#) ⁵²⁷
- [XML source and validation](#) ⁵³⁰
- [Static and dynamic type checking](#) ⁵³⁰
- [Library modules](#) ⁵³⁰
- [External functions](#) ⁵³⁰
- [Collations](#) ⁵³¹
- [Precision of numeric data](#) ⁵³¹
- [XQuery instructions support](#) ⁵³¹

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 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](#)⁵³². To use a specific collation, supply its URI as given in the [list of supported collations](#)⁵³². 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.

12.1.5 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.

Implementation-specific characteristics are the same as for [XQuery 1.0](#)⁵²⁸.

12.2 XSLT and XPath/XQuery Functions

This section lists Altova extension functions and other extension functions that can be used in XPath and/or XQuery expressions. Altova extension functions can be used with Altova's XSLT and XQuery engines, and provide functionality additional to that available in the function libraries defined in the W3C standards.

General points

The following general points should be noted:

- Functions from the core function libraries defined in the W3C specifications can be called without a prefix. That's because the XSLT and XQuery engines read non-prefixed functions as belonging to a default functions namespace which is that specified in the XPath/XQuery functions specifications <http://www.w3.org/2005/xpath-functions>. If this namespace is explicitly declared in an XSLT or XQuery document, the prefix used in the namespace declaration can also optionally be used on function names.
- In general, if a function expects a sequence of one item as an argument, and a sequence of more than one item is submitted, then an error is returned.
- All string comparisons are done using the Unicode codepoint collation.
- Results that are QNames are serialized in the form `[prefix:]localname`.

Precision of xs:decimal

The precision refers to the number of digits in the number, and a minimum of 18 digits is required by the specification. For division operations that produce a result of type `xs:decimal`, the precision is 19 digits after the decimal point with no rounding.

Implicit timezone

When two `date`, `time`, or `dateTime` values need to be compared, the 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.

12.2.1 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

[XSLT functions](#) ⁵³⁴

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](#) ⁵³⁷
- [Geolocation](#) ⁵⁵⁴
- [Image-related](#) ⁵⁶⁵
- [Numeric](#) ⁵⁷⁰
- [Sequence](#) ⁵⁹¹
- [String](#) ⁶⁰⁰
- [Miscellaneous](#) ⁶⁰⁶

12.2.1.1 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 `p1`, `p2`, `p3`... `pN`. Note the following points about usage: (i) The variables must be defined with names of the form `pX`, where `X` is an integer; (ii) the `altova:evaluate` function's arguments (see *signature above*), from the second argument onwards, provide the values of the variables, with the sequence of the arguments corresponding to the numerically ordered sequence of variables: `p1` to `pN`: The second argument will be the value of the variable `p1`, the third argument that of the variable `p2`, and so on; (iii) The variable values must be of type `item*`.

▣ Example

```
<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
<xsl:value-of select="altova:evaluate($xpath, 10, 20, 'hi')" />
outputs "hi 20 10"
```

In the listing above, notice the following:

- The second argument of the `altova:evaluate` expression is the value assigned to the variable `$p1`, the third argument that assigned to the variable `$p2`, and so on.
- Notice that the fourth argument of the function is a string value, indicated by its being enclosed in quotes.
- The `select` attribute of the `xs:variable` element supplies the XPath expression. Since this expression must be of type `xs:string`, it is enclosed in single quotes.

▣ Examples to further illustrate the use of variables

- ```
<xsl:variable name="xpath" select="'$p1'" />
<xsl:value-of select="altova:evaluate($xpath, //Name[1])" />
Outputs value of the first Name element.
```
- ```
<xsl:variable name="xpath" select="'$p1'" />
<xsl:value-of select="altova:evaluate($xpath, '//Name[1]')" />
Outputs "//Name[1]"
```

The `altova:evaluate()` extension function is useful in situations where an XPath expression in the XSLT stylesheet contains one or more parts that must be evaluated dynamically. For example, consider a

situation in which a user enters his request for the sorting criterion and this criterion is stored in the attribute `UserReq/@sortkey`. In the stylesheet, you could then have the expression: `<xsl:sort select="altova:evaluate(../UserReq/@sortkey)" order="ascending"/>`. The `altova:evaluate()` function reads the `sortkey` attribute of the `UserReq` child element of the parent of the context node. Say the value of the `sortkey` attribute is `Price`, then `Price` is returned by the `altova:evaluate()` function and becomes the value of the `select` attribute: `<xsl:sort select="Price" order="ascending"/>`. If this `sort` instruction occurs within the context of an element called `Order`, then the `Order` elements will be sorted according to the values of their `Price` children. Alternatively, if the value of `@sortkey` were, say, `Date`, then the `Order` elements would be sorted according to the values of their `Date` children. So the sort criterion for `Order` is selected from the `sortkey` attribute at runtime. This could not have been achieved with an expression like: `<xsl:sort select="../UserReq/@sortkey" order="ascending"/>`. In the case shown above, the sort criterion would be the `sortkey` attribute itself, not `Price` or `Date` (or any other current content of `sortkey`).

Note: The static context includes namespaces, types, and functions—but not variables—from the calling environment. The base URI and default namespace are inherited.

☐ *More examples*

- Static variables: `<xsl:value-of select="$i3, $i2, $i1" />`
Outputs the values of three variables.
- Dynamic XPath expression with dynamic variables:
`<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />`
`<xsl:value-of select="altova:evaluate($xpath, 10, 20, 30)" />`
Outputs "30 20 10"
- Dynamic XPath expression with no dynamic variable:
`<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />`
`<xsl:value-of select="altova:evaluate($xpath)" />`
Outputs error: No variable defined for \$p3.

▼ `encode-for-rtf` [altova:]

```
altova:encode-for-rtf(input as xs:string, preserveallwhitespace as xs:boolean,
preservenewlines as xs:boolean) as xs:string XSLT2 XSLT3
```

Converts the input string into code for RTF. Whitespace and new lines will be preserved according to the boolean value specified for their respective arguments.

[[Top](#)⁵³⁴]

XBRL functions

Altova XBRL functions can be used only with editions of Altova products that have XBRL support.

▼ `xbml-footnotes` [altova:]

```
altova:xbml-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](#) ⁵³⁴]

12.2.1.2 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](#) ⁵³⁸
- [Add a duration to xs:date and return xs:date](#) ⁵⁴⁰
- [Add a duration to xs:time and return xs:time](#) ⁵⁴²
- [Format and retrieve durations](#) ⁵⁴¹
- [Remove timezone from functions that generate current date/time](#) ⁵⁴²
- [Return days, hours, minutes, and seconds from durations](#) ⁵⁴⁴
- [Return weekday as integer from date](#) ⁵⁴⁵
- [Return week number as integer from date](#) ⁵⁴⁵
- [Build date, time, or duration type from lexical components of each type](#) ⁵⁴⁸
- [Construct date, dateTime, or time type from string input](#) ⁵⁴⁹
- [Age-related functions](#) ⁵⁵¹
- [Epoch time \(Unix time\) functions](#) ⁵⁵²

▼ Listed alphabetically

[altova:add-days-to-date](#) ⁵⁴⁰

[altova:add-days-to-dateTime](#) ⁵³⁸
[altova:add-hours-to-dateTime](#) ⁵³⁸
[altova:add-hours-to-time](#) ⁵⁴²
[altova:add-minutes-to-dateTime](#) ⁵³⁸
[altova:add-minutes-to-time](#) ⁵⁴²
[altova:add-months-to-date](#) ⁵⁴⁰
[altova:add-months-to-dateTime](#) ⁵³⁸
[altova:add-seconds-to-dateTime](#) ⁵³⁸
[altova:add-seconds-to-time](#) ⁵⁴²
[altova:add-years-to-date](#) ⁵⁴⁰
[altova:add-years-to-dateTime](#) ⁵³⁸
[altova:age](#) ⁵⁵¹
[altova:age-details](#) ⁵⁵¹
[altova:build-date](#) ⁵⁴⁸
[altova:build-duration](#) ⁵⁴⁸
[altova:build-time](#) ⁵⁴⁸
[altova:current-dateTime-no-TZ](#) ⁵⁴²
[altova:current-date-no-TZ](#) ⁵⁴²
[altova:current-time-no-TZ](#) ⁵⁴²
[altova:date-no-TZ](#) ⁵⁴²
[altova:dateTime-from-epoch](#) ⁵⁵²
[altova:dateTime-from-epoch-no-TZ](#) ⁵⁵²
[altova:dateTime-no-TZ](#) ⁵⁴²
[altova:days-in-month](#) ⁵⁴⁴
[altova:epoch-from-dateTime](#) ⁵⁵²
[altova:hours-from-dateTimeDuration-accumulated](#) ⁵⁴⁴
[altova:minutes-from-dateTimeDuration-accumulated](#) ⁵⁴⁴
[altova:seconds-from-dateTimeDuration-accumulated](#) ⁵⁴⁴
[altova:format-duration](#) ⁵⁴¹
[altova:parse-date](#) ⁵⁴⁹
[altova:parse-dateTime](#) ⁵⁴⁹
[altova:parse-duration](#) ⁵⁴¹
[altova:parse-time](#) ⁵⁴⁹
[altova:time-no-TZ](#) ⁵⁴²
[altova:weekday-from-date](#) ⁵⁴⁵
[altova:weekday-from-dateTime](#) ⁵⁴⁵
[altova:weeknumber-from-date](#) ⁵⁴⁷
[altova:weeknumber-from-dateTime](#) ⁵⁴⁷

[[Top](#) ⁵³⁷]

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](#)⁵³⁷]

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](#) ⁵³⁷]

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"

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`, `Seconds` as `xs:integer`) as `xs:time` [XP3.1](#) [XQ3.1](#)

Adds a duration in seconds to a time. The second argument is the number of seconds to be added to the `xs:time` supplied as the first argument. The result is of type `xs:time`. The Seconds component can be in the range of 0 to 59.999.

▣ Examples

- `altova:add-seconds-to-time`(`xs:time`("14:00:00"), 20) returns 14:00:20
- `altova:add-seconds-to-time`(`xs:time`("14:00:00"), 20.895) returns 14:00:20.895

Remove the timezone part from date/time datatypes [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`

[[Top](#) ⁵³⁷]

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.

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](#)⁵³⁷]

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](#)⁵³⁷]

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](#) ⁵³⁷]

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](#) ⁵³⁷]

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](#) ⁵³⁷]

12.2.1.3 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>	<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>

▼ 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](#) ⁵⁶⁵ 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) $D^{\circ}M'S.SS"N/S$ $D^{\circ}M'S.SS"E/W$ <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 $+/-D^{\circ}M'S.SS"$ $+/-D^{\circ}M'S.SS"$ <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 $+/-D.DD$ $+/-D.DD$ <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: `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	$33^{\circ}51'21.91"S$ $151^{\circ}13'11.73"E$

▼ 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`⁵⁶⁵ function and the Exif metadata's `@Geolocation`⁵⁶⁵ 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"W")` returns the sequence of two `xs:decimals` (48.858222222222, 24.2945)
- `altova:parse-geolocation("48°51'29.6"N 24°17'40.2"W")` 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](#)⁵⁶⁵ function and the Exif metadata's [@Geolocation](#)⁵⁶⁵ 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)
 $D^{\circ}M'S.SS''N/S$ $D^{\circ}M'S.SS''W/E$
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

▼ 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`⁵⁶⁵ function and the Exif metadata's `@Geolocation`⁵⁶⁵ 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°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

▼ `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](#) ⁵⁶⁵ 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
`+/-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

☐ 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) <code>D°M'S.SS"N/S D°M'S.SS"E/W</code> <i>Example:</i> 33°55'11.11"N 22°44'66.66"W

2
Decimal degrees, with suffixed orientation (N/S, E/W) <code>D.DDN/S D.DDE/W</code> <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

▼ 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](#)⁵⁶⁵ function and the Exif metadata's [@Geolocation](#)⁵⁶⁵ 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")` 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).

GPSPLatitude	GPSPLatitudeRef	GPSPLongitude	GPSPLongitudeRef	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](#)⁵⁶⁵ function and the Exif metadata's [@Geolocation](#)⁵⁶⁵ 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"E") 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

12.2.1.4 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>	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

▼ 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

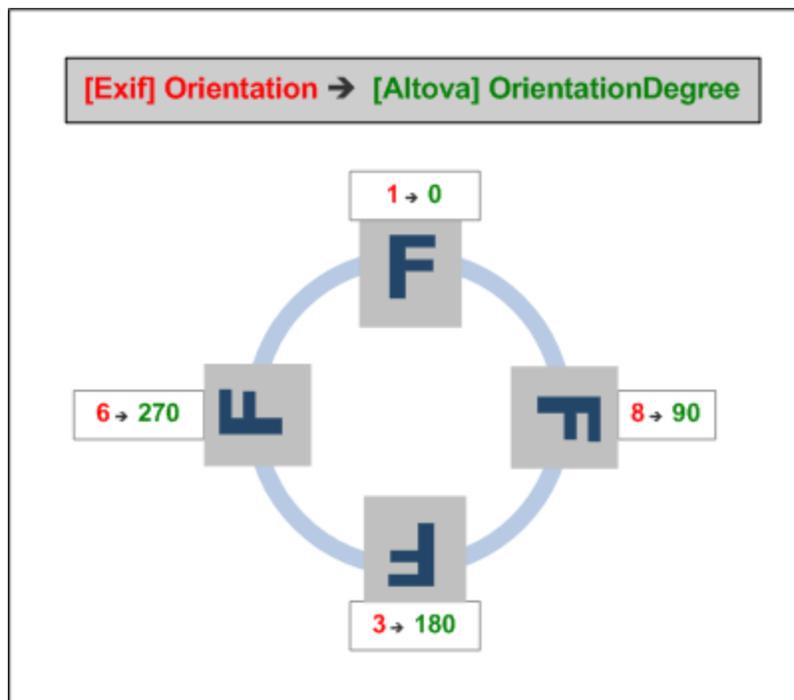
The Altova XPath/XQuery Engine generates the custom attribute `Geolocation` from standard Exif metadata tags. `Geolocation` is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (see table below).

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](#) ⁵⁶⁵]

12.2.1.5 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](#)⁵⁷⁰]

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](#)⁵⁷⁰]

Number-formatting functions

[[Top](#)⁵⁷⁰]

12.2.1.6 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>	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

▼ 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	Sequence of strings	

substitutions	("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            |

### 12.2.1.7 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>     | <b>XP1</b> <b>XP2</b> <b>XP3.1</b>     |
| <i>XSLT functions (used in XPath expressions in XSLT):</i>      | <b>XSLT1</b> <b>XSLT2</b> <b>XSLT3</b> |
| <i>XQuery functions (used in XQuery expressions in XQuery):</i> | <b>XQ1</b> <b>XQ3.1</b>                |

## ▼ 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)

### 12.2.1.8 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>     | <b>XP1</b> <b>XP2</b> <b>XP3.1</b>     |
| <i>XSLT functions (used in XPath expressions in XSLT):</i>      | <b>XSLT1</b> <b>XSLT2</b> <b>XSLT3</b> |
| <i>XQuery functions (used in XQuery expressions in XQuery):</i> | <b>XQ1</b> <b>XQ3.1</b>                |

#### ▼ 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(InputChangeString 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 `InputChangeString` 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(InputChangeString as xs:string) as xs:string` XP2 XQ1 XP3.1 XQ3.1

`altova:create-hash-from-string(InputChangeString as xs:string, HashAlgo as xs:string) as xs:string` XP2 XQ1 XP3.1 XQ3.1

Generates a hash string from `InputChangeString` 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(AsString 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"

### 12.2.1.9 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>     | <b>XP1 XP2 XP3.1</b>     |
| <i>XSLT functions (used in XPath expressions in XSLT):</i>      | <b>XSLT1 XSLT2 XSLT3</b> |
| <i>XQuery functions (used in XQuery expressions in XQuery):</i> | <b>XQ1 XQ3.1</b>         |

▼ `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](#)<sup>606</sup> ]

## 12.2.2 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](#)<sup>609</sup> and [.NET](#)<sup>617</sup>, as well as [MSXSL scripts for XSLT](#)<sup>623</sup>. 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](#)<sup>609</sup>
- [.NET Extension Functions](#)<sup>617</sup>
- [MSXSL Scripts for XSLT](#)<sup>623</sup>

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.

### 12.2.2.1 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](#) <sup>614</sup>
- [Java: Static Methods and Static Fields](#) <sup>614</sup>
- [Java: Instance Methods and Instance Fields](#) <sup>615</sup>
- [Datatypes: XPath/XQuery to Java](#) <sup>616</sup>
- [Datatypes: Java to XPath/XQuery](#) <sup>617</sup>

#### 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](#)<sup>610</sup> and [User-Defined Jar Files](#)<sup>613</sup>. 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.

### 12.2.2.1.1 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](#)<sup>611</sup>.)
- The class file is not packaged. The XSLT or XQuery file is in the same folder as the class file. ([See example below](#)<sup>611</sup>.)
- The class file is in a package. The XSLT or XQuery file is at some random location. ([See example below](#)<sup>612</sup>.)
- The class file is not packaged. The XSLT or XQuery file is at some random location. ([See example below](#)<sup>612</sup>.)

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()" />

</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()" />

</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)"/>
 </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)"/>
</xsl:template>

</xsl:stylesheet>

```

**Note:** When a path is supplied via the extension function, the path is added to the ClassLoader.

### 12.2.2.1.2 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) "/>
</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.

### 12.2.2.1.3 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](#)<sup>617</sup>, 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](#)<sup>615</sup>):
 

```
<xsl:value-of select="date:toString(date:new()) " xmlns:date="java:java.util.Date" />
```

### 12.2.2.1.4 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>

```

### 12.2.2.1.5 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 `java.lang.Object.getClass` method.
3. The object obtained by the `getClass` method is passed as the argument to the `java.lang.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`.

### 12.2.2.1.6 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.

### 12.2.2.1.7 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).

### 12.2.2.2 .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](#) <sup>620</sup>
- [.NET: Static Methods and Static Fields](#) <sup>620</sup>
- [.NET: Instance Methods and Instance Fields](#) <sup>621</sup>

- [Datatypes: XPath/XQuery to .NET](#) <sup>622</sup>
- [Datatypes: .NET to XPath/XQuery](#) <sup>623</sup>

## 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.
<code>loc</code>	The locale, for example, <code>en-US</code> . The default is <code>neutral</code> .

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 semicolon. 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=b9f091b72dccb8";
```

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.

### 12.2.2.2.1 .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](#)<sup>617</sup>, 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](#)<sup>615</sup>):
 

```
<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" />
```

### 12.2.2.2.2 .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>
```

### 12.2.2.2.3 .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.

#### 12.2.2.2.4 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</code> , <code>string</code>
<code>xs:boolean</code>	<code>BooleanValue</code> , <code>bool</code>
<code>xs:integer</code>	<code>IntegerValue</code> , <code>decimal</code> , <code>long</code> , <code>integer</code> , <code>short</code> , <code>byte</code> , <code>double</code> , <code>float</code>
<code>xs:float</code>	<code>FloatValue</code> , <code>float</code> , <code>double</code>
<code>xs:double</code>	<code>DoubleValue</code> , <code>double</code>
<code>xs:decimal</code>	<code>DecimalValue</code> , <code>decimal</code> , <code>double</code> , <code>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.

#### 12.2.2.2.5 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).

#### 12.2.2.3 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>
```

```
</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>
 Total Retail Price =
 $<xsl:value-of select="user:AddMargin(50)" />

 Total Wholesale Price =
 $<xsl:value-of select="50" />

 </p>
 </body>
 </html>
 </template>
</xsl:stylesheet>
```

```
</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.

## 12.3 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](#) <sup>626</sup>
- [MS Access](#) <sup>627</sup>
- [MS SQL Server](#) <sup>628</sup>
- [MySQL](#) <sup>628</sup>
- [ODBC](#) <sup>629</sup>
- [Oracle](#) <sup>630</sup>
- [Sybase](#) <sup>631</sup>

### 12.3.1 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

### 12.3.2 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

### 12.3.3 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

### 12.3.4 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

### 12.3.5 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

### 12.3.6 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.

### 12.3.7 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

## 12.4 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](#) <sup>632</sup>
- [Altova Engines](#) <sup>632</sup>
- [Unicode Support](#) <sup>633</sup>
- [Internet Usage](#) <sup>633</sup>

### 12.4.1 OS and Memory Requirements

#### Operating System

Altova software applications are available for the following platforms:

- Windows 7 SP1 with Platform Update, Windows 8, Windows 10, Windows 11
- Windows Server 2008 R2 SP1 with Platform Update or newer

#### Memory

Since the software is written in C++ it does not require the overhead of a Java Runtime Environment and typically requires less memory than comparable Java-based applications. However, each document is loaded fully into memory so as to parse it completely and to improve viewing and editing speed. As a result, the memory requirement increases with the size of the document.

Memory requirements are also influenced by the unlimited Undo history. When repeatedly cutting and pasting large selections in large documents, available memory can rapidly be depleted.

### 12.4.2 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.

### 12.4.3 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.

### 12.4.4 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.

## 12.5 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](#).

### 12.5.1 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](http://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.

## 12.5.2 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](https://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.

### 12.5.3 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), 14, 28**

**.NET extension functions,**

- constructors, 620
- datatype conversions (.NET to XPath/XQuery), 623
- datatype conversions (XPath/XQuery to .NET), 622
- for XSLT and XQuery, 617
- in XPath expressions, 397, 409
- instance methods, instance fields, 621
- overview, 617
- static methods, static fields, 620
- support for, in Authentic View, 397, 409

## A

**Abbreviations,**

- used in user manual, 23

**About StyleVision, 524**

**Activating the software, 520**

**Adding schema, 423**

**Additional editing procedures, 337**

**Aligning table cell content,**

- in SPSs, 486

**Altova extensions,**

- chart functions (see chart functions), 533

**Altova website, 524**

**Altova XML Parser,**

- about, 632

**AltovaXML,**

- and FOP, 387

**Append,**

- column to table in SPS, 483
- row to table in SPS, 483

**Appendices, 525**

**ASP.NET application, 374**

**ASPX web application, 374**

**Assign predefined formats,**

- in Quick Start tutorial, 67

**Authentic Browser, 17**

**Authentic Desktop, 17**

**Authentic View,**

- in Altova products, 17

**Auto Hide,**

- feature of Design Entry Helpers, 30

**Auto-Calculations, 240**

- and conditions, 249
- and output escaping, 308
- command for inserting in design, 459
- creating, editing, formatting, 240
- example files, 242
- examples, 254
- formatting of date results, 359
- how to use, 240
- in Quick Start tutorial, 73
- Java and .NET functions in (Enterprise edition only), 240
- moving, 240
- symbol in Design View, 393
- updating node with value of, 459

**Automated processing, 383**

**Auto-numbering, 293**

## B

**Background Information, 632**

**Barcodes, 155**

**Base year,**

- in input formatting, 310

**Batch files,**

- and scheduled tasks, 391

**Blueprints for layout, 159**

**Bookmarks, 154, 298**

- command for inserting in design, 466
- creating and editing, 298
- deleting, 298
- enclosing with, 478

**Bookmarks (anchors),**

- symbol in Design View, 393

**Borders,**

- of SPS tables, 485

**Bullets and Numbering, 138, 140, 464, 487**

- enclosing with, 477

**Buttons, 153**

## C

**CALS/HTML tables, 133, 486**

**Catalog customization, 98**

**Catalog files, 96**

**Catalog mechanism overview, 96**

**Catalogs and environment variables, 100**

**Catalogs in RaptorXML, 97**

**CDATA sections, 103**

**Cell (of table),**

split horizontally, 484

split vertically, 484

**Cells,**

joining in SPS tables, 484

**Change To command, 169**

**Character references,**

and output escaping, 308

**Check boxes, 149**

**Class attributes,**

in Quick Start tutorial, 67

**Close (SPS) command, 429**

**Column,**

append to SPS table, 483

delete from table in SPS, 484

insert in SPS table, 483

**Columns (of tables),**

hiding in HTML output, 132

**Combo box,**

in Quick Start tutorial, 77

**Combo boxes, 151**

**Command line, 383**

and parameters, 263

and scheduled tasks, 391

**Command line utility, 19**

**Commands,**

customizing, 509

**Commenting out content, 470, 480**

**Companion software,**

for download, 524

**Complex global template, 215**

**Component download center,**

at Altova web site, 524

**Composite styles, 332**

**Condition,**

command for inserting in design, 468

**Conditional templates, 468**

see under: Conditions, 245

symbol in Design View, 393

**Conditions,**

and Auto-Calculations, 249

editing, 248

enclosing with, 478

in Quick Start tutorial, 77

setting up, 245

**Consecutive markup, 27**

**Content editing procedures, 102**

**Contents,**

command for inserting in design, 457

**Contents placeholder,**

in Quick Start tutorial, 55

inserting node as contents, 103

**Context node,**

in XPath dialog, 397, 409

**Copy command, 447**

**Copyright information, 634**

**CoreCatalog.xml, 97**

**Creating new SPS document,**

in Quick Start tutorial, 51

**Cross references, 297**

**CSS files,**

managing in Design Overview sidebar, 32

**CSS styles,**

in Modular SPSs, 205

in Quick Start tutorial, 67

see also Styles, 43

**CSS stylesheets,**

also see Styles, 320

external stylesheets, 320

import precedence of external, 320

media applied to, 320

**Custom dictionaries,**

for SPS spell-checks, 491

**CustomCatalog.xml, 97**

**Customize dialog,**

for customizing StyleVision, 454

**Customizing StyleVision, 509**

**Cut command, 447**

## D

**Database,**

**Database,**

toolbar buttons for editing, 421

**Database (Enterprise and Professional editions),**

see under DB, 13

**Data-entry devices, 148**

menu commands for inserting, 458

symbol in Design View, 393

**Date,**

formatting of, 310

**Dates,**

examples of data manipulation with XPath 2.0, 359

formatting of, 359

how to use in SPS, 359

**DB Parameters,**

creating and editing, 452

**Decimals,**

formatting of, 310

**Default user dictionary,**

for SPS spell-checks, 491

**Delete,**

column from table in SPS, 484

row from table in SPS, 484

table in SPS, 482

**Delete command, 447****Design elements, 418****Design Entry Helper windows,**

docking, 30

floating, 30

**Design Entry Helpers,**

Auto Hide, 30

description of, 30

Hide, 30

switching display on and off, 455

**Design Filters,**

switching on and off, 455

**Design Fragment,**

insert, 473

**Design Fragments, 225****Design Overview,**

sidebar window, 32

**Design structure, 172****Design Tree,**

and Modular SPSs, 205

see also Design Entry Helpers, 30

sidebar window, 38

**Design View,**

and JavaScript Editor, 27

description of, 27

display of markup, 27

symbols in SPS design, 393

**Dictionaries,**

for SPS spell-checks, 491

**Disabled command, 470, 480****disable-output-escaping, 308****Distribution,**

of Altova's software products, 634

**Docking,**

Design Entry Helper windows, 30

**Document element,**

definition of, 20

**Document elements (see Root elements), 174****Document node,**

definition of, 20

**Document properties, 238****Document styles, 238****Document views,**

in GUI, 26

**Documentation,**

overview of, 23

**Documents,**

opening and closing, 26

**DPI, 443****DTD,**

declaring unparsed entities, 338

**DTDs,**

as SPS source, 175

**DTDs and catalogs, 96****Dynamic content,**

in Quick Start tutorial, 55

**Dynamic lists, 138, 140, 464****Dynamic table,**

toolbar buttons for editing, 417

**Dynamic tables, 118**

and global templates, 121

difference from appended/inserted rows, 121

headers and footers in, 121

nested dynamic tables, 121

see also SPS tables, 121

see also Tables, 128

## E

**Edit menu, 447****Edit Parameters dialog, 452**

**Edit Template Match command, 112**

**Edit XPath Expression dialog,**

see XPath dialog, 397

**Element templates,**

user-defined, 115

**Elements,**

user-defined, 115

**Enclose With menu, 475**

**Encoding,**

for output files, 513

**Encoding command, 443**

**Encoding of output documents, 443**

**End User License Agreement, 634, 636**

**Entities,**

unparsed, 338

using as URI holders, 338

**Entity references,**

and output escaping, 308

**Entry helpers in Design View,**

switching display on and off, 455

**Environment variables used in catalogs, 97**

**Environment variables, 100**

**Evaluation key,**

for your Altova software, 520

**Evaluation period,**

of Altova's software products, 634

**Event handlers,**

assigning functions to, 364

**Excel table content,**

copy-pasting into design, 107

**Exit command, 445**

**Extension functions for XSLT and XQuery, 608**

**Extension Functions in .NET for XSLT and XQuery,**

see under .NET extension functions, 617

**Extension Functions in Java for XSLT and XQuery,**

see under Java extension functions, 609

**Extension Functions in MSXSL scripts, 623**

## F

**FAQs on StyleVision, 524**

**Features,**

of StyleVision, 14

**File menu, 423**

command Exit, 445

File | Close, 429

File | Encoding, 443

File | New, 423

File | Open, 429

File | Print, 444

File | Print Preview, 444

File | Save As, 439

File | Save Design, 434

File | Save Generated Files, 440

**File modification alerts,**

in Modular SPSs, 205

**Files,**

open recently used, 445

**Filters,**

for viewing templates selectively, 420

**Filters for design templates,**

switching on and off, 455

**Filters on node-templates, 223**

**Find,**

using regular expressions, 447

**Find & Replace sidebar, 48**

**Find command, 447**

**Find in Design View, 48**

**Find Next command, 447**

**Floating,**

Design Entry Helper Windows, 30

**FO processor (Enterprise edition),**

setting up, 19

**FO transformations, 387**

**Footers,**

adding in table, 483

in tables, 128

**Form controls,**

menu commands for inserting, 458

**Format strings,**

defining for Input Formatting, 487

**Formatting,**

also see Presentation, 305

for tables, 128

lists, 416

nodes on insertion, 105

of numeric fields, 310

overview of procedures, 305

predefined HTML formats, 416

text alignment, 416

text properties, 416

toolbar buttons for, 416

**Formatting numbers,**

in Auto-Numbering, 293

**Form-based designs, 159, 423**

**Functions,**

in XPath, defined by user, 344

## G

**General usage procedure, 89**

**Generated files, 95**

**Global styles,**

see under Styles, 323

**Global templates, 215**

effect on rest-of-contents, 106

in Quick Start tutorial, 84

**Global types,**

in templates, 215

**Graphics,**

overview of use in SPS, 143

see also under Images, 143

**Grouping, 250**

group-by example (Persons.sps), 252

group-by example (Scores.sps), 254

**GUI,**

description of, 25

document views in, 26

Main Window of, 26

multiple documents in, 26

## H

**Headers,**

adding in table, 483

in tables, 128

**Help,**

see Onscreen Help, 520

**Help menu, 520**

**Hide,**

feature of Design Entry Helpers, 30

**Hide markup, 27**

**Horizontal line,**

command for inserting in design, 463

in Quick Start tutorial, 62

**HTML document properties, 335**

**HTML import, 367**

creating a new SPS, 367

generating files from SPS, 373

of HTML lists, 371

of HTML tables, 371

schema structure, 369

SPS design, 369

**HTML output, 95**

and image support, 145

**HTML page content,**

copy-pasting into design, 107

**HTML tables, 133, 486**

**HTML to XML conversion, 367**

**Hyperlink,**

command for inserting in design, 467

**Hyperlinks, 154, 298**

and unparsed entities, 300

creating and editing, 300

enclosing with, 478

linking to bookmarks, 300

linking to external resources, 300

locating via hyperlinks, 338

removing and deleting, 300

symbol in Design View, 393

## I

**IE 9,**

see under Internet Explorer compatibility, 93

**Image,**

command for inserting in design, 461

**Images,**

accessing for output rendering, 143

and unparsed entity URIs, 143

example files, 147

in Quick Start tutorial, 62

locating via unparsed entities, 338

specifying URIs for, 143

supported types, 145

symbol in Design View, 393

**Import of XSLT templates,**

into SPS, 229

**Input fields, 149**

**Input formatting,**

defining format strings for, 487

of dates, 359

**Insert,**

column in SPS table, 483

**Insert,**

row in SPS table, 483

**Insert menu, 457**

- Bullets and Numbering, 464
- Insert | Auto-Calculation, 459
- Insert | Bookmarks, 466
- Insert | Condition, 468
- Insert | Contents, 457
- Insert | Design Fragment, 473
- Insert | Disabled, 470
- Insert | Horizontal Line, 463
- Insert | Hyperlink, 467
- Insert | Image, 461
- Insert | Paragraph, 460
- Insert | Rest of contents, 458
- Insert | Special Paragraph, 460

**Inserting design elements via the toolbar, 418****Integer,**

formatting of, 310

**Interface,**

see GUI, 25

**Internet Explorer compatibility, 93****Internet usage,**

in Altova products, 633

**J****Java and .NET functions (Enterprise edition only),**

in Auto-Calculations, 240

**Java extension functions,**

- constructors, 614
- datatype conversions, Java to Xpath/XQuery, 617
- datatype conversions, XPath/XQuery to Java, 616
- for XSLT and XQuery, 609
- in XPath expressions, 397, 409
- instance methods, instance fields, 615
- overview, 609
- static methods, static fields, 614
- support for, in Authentic View, 397, 409
- user-defined class files, 610
- user-defined JAR files, 613

**JavaScript,**

see under Scripts, 362

**JavaScript Editor, 362, 363**

in Design View, 27

**Joining cells,**

in SPS tables, 484

**K****Keyboard shortcuts,**

customizing for commands, 509

**Key-codes,**

for your Altova software, 520

**L****Layout,**

of views in the GUI, 30

**Layout Box, 473****Layout Boxes, 162****Layout Container, 473****Layout Containers, 159****Layout containers and elements, 418****Layout Modules,**

steps for creating, 159

**Legal information, 634****License, 636**

information about, 634

**License metering,**

in Altova products, 635

**Licenses,**

for your Altova software, 520

**Line,**

in Layout Containers, 473

**Links,**

see under Hyperlinks, 154, 298

**List properties, 487****Lists, 138**

- enclosing with, 477
- imported from HTML document, 371
- in Quick Start tutorial, 77

**Lists (static and dynamic), 464****Local styles,**

see under Styles, 325

**Local template, 215**

## M

- Main schema, 215**
- Main schema (Enterprise Edition only), 35**
- Main template, 215**
  - definition of, 20
- Markup tags in Design View, 27**
- Memory requirements, 632**
- Menu,**
  - customizing, 509
- Menu bar,**
  - moving, 25
- Messages sidebar, 48**
- Metadata of output HTML document, 335**
- Microsoft Office 2007 (Enterprise Edition only), 14, 28**
- MobileTogether design,**
  - export to, 440
- Modular SPS,**
  - activating and de-activating, 205
  - adding the SPS module, 205
  - and CSS styles, 202, 205
  - and file modification alerts, 205
  - and module objects, 202
  - and namespace declarations, 202
  - and schema sources, 202, 205
  - and Scripts, 202
  - and Template XML Files, 202
  - and Working XML Files, 202
  - creating, 205
  - effect of order on precedence, 205
  - example project, 209
  - overview, 201
  - the SPS module to add, 205
  - working with, 205
- Modules,**
  - managing in Design Overview sidebar, 32
- MS Word document content,**
  - copy-pasting into design, 107
- msxsl:script, 623**
- Multiline input fields, 149**
- Multiple document-outputs, 474**
- Multiple output-documents, 231**
  - and output previews, 235
  - linking between, 233
  - location of files, 235

## N

- Named templates, 215**
  - Namespaces,**
    - adding to the SPS, 35, 90, 95, 174
    - in the SPS, 35
    - overview of, 38
  - New command, 423**
  - New document templates, 231**
    - and design structure, 233
    - inserting, 232
    - URLs of, 233
  - New from XSLT, 340**
  - Node,**
    - changing what it is created as, 169
  - Node-templates,**
    - and chaining to child templates, 223
    - and global templates, 223
    - and XPath filters, 223
    - operations on, 223
    - User-Defined, 112
  - Numbering nodes automatically, 293**
  - Numbers,**
    - formatting of, 310
  - Numeric fields,**
    - formatting of, 310
- 
- ## O
- Office Open XML (Enterprise Edition only), 14, 28**
  - Onscreen help,**
    - index of, 520
    - searching, 520
    - table of contents, 520
  - OOXML (Enterprise Edition only), 14, 28**
  - Open,**
    - recently used files, 445
  - Open (SPS) command, 429**
  - Ordering Altova software, 520**
  - OS,**
    - for Altova products, 632
  - Otherwise condition branch, 245**
  - Output encoding, 443**

**Output escaping, 308****Output files,**

generating, 95

**Output Views,**

description of, 28

## P

**Paragraph,**

command for inserting in design, 460

enclosing with, 476

**Parameters, 263**

and Authentic View, 263

and command line, 263

creating and editing, 452

for design fragments, 264

for schema sources, 267

general description, 263

in SPS, 263

locating nodes in in multiple documents with, 267

managing in Design Overview sidebar, 32

overview of user-defined parameters, 38

**Parser,**

built into Altova products, 632

**Paste command, 447****PDF output (Enterprise edition), 95**

and image support, 145

**Pixels,**

and print media lengths, 443

and screen resolution, 443

**Platforms,**

for Altova products, 632

**Precedence,**

of styles, 41

**Predefined format strings,**

for input formatting, 487

**Predefined formats,**

command for inserting in design, 460

on inserting a node, 105

symbol in Design View, 393

**Presentation,**

also see Formats, Formatting, 305

overview of procedures, 305

**Print command, 444****Print Preview command, 444****Problems with preview, 19****Processors,**

for download, 524

**Product features,**

listing of, 14

**Project options, 513****Properties,**

and property groups, 44

defining, 44

of SPS tables, 417, 485

see also Design Entry Helpers, 30

sidebar window, 44

**Properties Entry Helper,**

Event group, 364

**Properties menu, 487**

Bullets and Numbering, 487

**Properties of output documents, 238****PXF files, 377**

creating, 377

deploying, 381

editing, 380

saving as, 439

## Q

**Quick Start tutorial,**

Auto-Calculations, 73

class attributes, 67

combo boxes, 77

conditions, 77

contents placeholder, 55

creating new SPS document, 51

CSS styles, 67

dynamic content, 55

generating XSLT stylesheets, 88

global templates, 84

horizontal lines, 62

images, 62

introduction, 50

lists, 77

predefined formats, 67

required files, 50

rest-of-contents, 84

setting up new SPS document, 51

static content, 62

static text, 62

**Quick Start tutorial,**

testing Authentic View (Enterprise and Professional editions), 88

**R****Radio buttons, 153****RaptorXML, 383**

and FOP, 387

**Recently used files, 445****Redo command, 447****Registering your Altova software, 520****Regular expressions,**

find and replace using, 447

**Replace,**

using regular expressions, 447

**Replace command (Enterprise and Professional editions), 447****Replace in Design View, 48****Rest-of-contents, 106**

and global templates, 215

command for inserting in design, 458

in Quick Start tutorial, 84

**Restore toolbars and windows, 513****Root elements, 35****Root elements (aka document elements),**

and schema sources, 174

selecting for schema, 174

**RootCatalog.xml, 97****Row,**

append to SPS table, 483

delete from table in SPS, 484

insert in SPS table, 483

**Rows (of tables),**

expanding/collapsing in HTML output, 132

**RTF output (Enterprise and Professional editions), 95****RTF output (Enterprise edition),**

and image support, 145

**S****Save Design command, 434****Save Generated Files command, 440****Scheduled task,**

creating a StyleVisionBatch command as, 391

StyleVisionBatch batch files in, 391

**Schema Manager,**

CLI Help command, 191, 503

CLI Info command, 192, 504

CLI Initialize command, 192, 504

CLI Install command, 193, 505

CLI List command, 193, 505

CLI overview, 191, 503

CLI Reset command, 194, 506

CLI Uninstall command, 195, 507

CLI Update command, 196, 508

CLI Upgrade command, 196, 508

how to run, 185, 497

installing a schema, 189, 501

listing schemas by status in, 187, 499

overview of, 182, 494

patching a schema, 189, 501

resetting, 190, 502

status of schemas in, 187, 499

uninstalling a schema, 190, 502

upgrading a schema, 189, 501

**Schema sources, 90, 423**

and root elements (document elements), 174

changing sources, 267

managing in Design Overview sidebar, 32

multiple in SPS (Enterprise edition), 174

multiple sources and locating nodes, 267

multiple sources and XPath, 267

overview of, 38

selecting for SPS, 174

sidebar window, 35

**Schema Sources window,**

see also Design Entry Helpers, 30

**Schema tree options, 513****Schemas,**

as SPS source, 175

looking up via catalogs, 98

user-defined, 180

**Schemas and catalogs, 96****Scripts,**

and JavaScript functions, 362

defining JavaScript functions, 363

in the Design Tree, 362

JavaScript functions as event handlers, 364

overview of, 38

using in an SPS, 362

**Scripts in XSLT/XQuery,**

see under Extension functions, 608

- Scroll buttons,**
  - in Main Window, 26
- Select All command, 447**
- Setting up new SPS document,**
  - in Quick Start tutorial, 51
- Setting up StyleVision, 19**
- Shortcuts,**
  - customizing for keyboard, 509
- Show markup, 27**
- Simple global template, 215**
- Software product license, 636**
- Sorting, 258**
  - example files, 260
  - of groups and within groups, 250, 252, 254
  - Sorting mechanism, 258
  - Sort-keys, 258
- Sort-keys, 258**
- Source files for SPS, 90**
- Special paragraph,**
  - command for inserting in design, 460
  - enclosing with, 476
- Spell-checker,**
  - in StyleVision, 490
- Spell-checker options,**
  - for SPSs, 491
- Split table cell,**
  - horizontally, 484
  - vertically, 484
- SPS,**
  - and Authentic View (Enterprise and Professional editions), 18
  - and StyleVision, 18
  - and XSLT stylesheets, 18
  - closing, 429
  - general description of, 18
  - opening, 429
  - reloading, 429
- SPS design overview, 91**
- SPS file structure, 172**
- SPS tables,**
  - see also Dynamic tables, 118
  - see also Static tables, 118
- Static content,**
  - in Quick Start tutorial, 62
- Static lists, 138, 464, 477**
- Static table,**
  - inserting, 482
  - inserting in SPS, 417
  - toolbar buttons for editing, 417
- Static tables, 118**
  - see also SPS tables, 120
  - see also Tables, 128
- Static text,**
  - and output escaping, 308
  - in Quick Start tutorial, 62
- Status bar, 454**
- Structure of SPS design, 172**
- Style Repository,**
  - and external CSS stylesheets, 320
  - and global styles, 323
  - see also Design Entry Helpers, 30
  - sidebar window, 41
- Styles,**
  - and property groups, 43
  - assigning CSS stylesheets to SPS, 320
  - cascading order, 319
  - combining several, 332
  - CSS rules combined, 332
  - defining, 43
  - defining global styles in SPS, 323
  - defining local styles, 325
  - from XML data, 329
  - media for assigned external stylesheets, 320
  - precedence of, 41
  - precedence of styles, 323
  - see also Design Entry Helpers, 30
  - sidebar window, 43
  - terminology of, 319
  - via XPath expressions, 329
  - working with in StyleVision, 319
- Styles of output documents, 238**
- Stylesheets,**
  - also see under CSS stylesheets, 320
  - also see under XSLT stylesheets, 320
- StyleVision,**
  - product features, 14
  - user manual, 13
- StyleVision Power Stylesheet,**
  - see under SPS, 13
- StyleVisionBatch, 19, 383**
- Support for StyleVision, 524**
- Support options, 23**
- Symbols in Design View,**
  - of Auto-Calculations, 393
  - of bookmarks (anchors), 393
  - of conditional templates, 393

**Symbols in Design View,**

- of data-entry devices, 393
- of hyperlinks, 393
- of images, 393
- of predefined formats, 393
- of XML document content, 393
- of XML document nodes, 393

**T****Table,**

- adding headers and footers, 483
- append column to, 483
- append row to, 483
- cell content, 482
- delete column from, 484
- delete row from, 484
- deleting in SPS, 482
- editing properties of, 485
- headers and footers, 482
- insert column in, 483
- insert row in, 483
- inserting a static table, 482
- navigating, 482
- show/hide borders in StyleVision, 485
- vertical alignment of cell content, 486

**Table menu, 482****Table of contents,**

- see under TOC, 271

**Tables,**

- Close button to hide columns, 132
- conditional processing in, 125
- creating, 463
- creating dynamic tables, 121
- creating static tables, 120
- expanding/collapsing rows, 132
- formatting, 128
- headers and footers in PDF, 128
- hiding empty columns, 132
- imported from HTML document, 371
- joining cells in, 484
- overview, 118
- styles for alternate rows, 329

**Tables (SPS),**

- editing of properties, 417
- toolbar buttons for editing, 417

**Tables in Design View,**

- enclosing with and removing templates, 126
- representation of, 126

**Tags,**

- expanding and collapsing, 453

**Technical Information, 632****Technical support for StyleVision, 524****Template,**

- changing the node match for, 169
- enclosing with, 475
- inserting, 470

**Template filters, 420****Template XML File (Enterprise and Professional editions), 90**

- definition of, 20

**Templates,**

- enclosing table rows and columns with, 126
- removing from around table rows and columns, 126
- switching view on and off, 455
- tree of, 38

**Templates for nodes,**

- see Node-templates, 223

**Temporary output document, 19****Terminology,**

- used in StyleVision, 20

**Text output (Enterprise and Professional editions), 95****Text references, 297****TOC,**

- example, hierarchical and sequential, 290
- example, simple, 286
- marking items for inclusion, 274
- menu commands, 473
- overview of usage, 271

**TOC Bookmarks, 274**

- and levels, 278
- creating, 278
- enclosing with, 480
- wizard for, 278

**TOC items,**

- constructing, 284
- formatting, 284

**TOC Levels, 274, 275**

- enclosing with, 480

**TOC references, 284****TOC template,**

- creating and editing, 281
- formatting, 284
- level references in, 283

**TOC template,**

- reflevels in, 283
- structuring, 283

**TOCrefs,**

- see under TOC references, 284

**Toolbar buttons,**

- adding and removing, 415

**Toolbars, 414**

- adding/removing icons in, 414
- customizing, 454
- Formatting toolbar, 416
- Insert Design Elements toolbar, 418
- moving, 25
- positioning in GUI, 414
- resetting, 414
- Standard toolbar, 421
- switching display on and off, 454
- switching display on/off, 414
- Table toolbar, 417

**Tools menu, 490****Type-based templates, 215****Types as processing units,**

- in global templates, 215

## U

**User-Defined Elements, 115****User-Defined XML Text Blocks, 116****Undo command, 447****Unicode support,**

- in Altova products, 633

**unparsed-entity-uri function of XSLT, 338****Updating nodes (Enterprise and Professional editions),**

- with an Auto-Calculation result, 240

**URIs,**

- holding in unparsed entities, 338

**Usage, 89****User Interface,**

- see GUI, 25

**User manual,**

- see also Onscreen Help, 520

**User reference, 392****User-Defined Elements, 115, 474, 481****User-defined schemas, 180****User-defined template,**

- enclosing with, 476

- inserting, 471

**User-Defined Templates, 112****User-Defined Text Blocks, 115, 474****User-defined XPath functions, 344**

## V

**Validator,**

- in Altova products, 632

**Value formatting, 310****Variable template, 222**

- enclosing with, 476
- inserting, 472

**Variables, 263, 268****Vertical alignment of table cell content,**

- in SPSS, 486

**Vertical text,**

- in layout boxes, 162
- in table cells, 128

**View menu, 454****Views,**

- layout of in GUI, 30

## W

**Window menu, 519****Windows,**

- support for Altova products, 632

**Word 2007 (Enterprise Edition only), 14, 28****Word 2007+ output (Enterprise edition), 95****Word document content,**

- copy-pasting into design, 107

**WordML (Enterprise Edition only), 14, 28****Working XML File, 35, 90**

- and Output Views, 28
- definition of, 20
- print preview, 444
- printing, 444

## X

**XML,**

- inserting in design, 116

- XML data,**
  - inserting in SPS design, 103
  - merging from multiple sources, 198
- XML document content,**
  - symbol in Design View, 393
- XML document nodes,**
  - symbol in Design View, 393
- XML Parser,**
  - about, 632
- XML Schemas and DTDs,**
  - as SPS source, 175
- XML tables (Enterprise and Professional editions), 118**
- XMLSpy, 17**
- XPath,**
  - locating nodes in multiple documents, 267
- XPath 1.0,**
  - and dates, 359
- XPath 2.0,**
  - and dates, 359
- XPath dialog,**
  - debugging expressions in, 401
  - description of, 397, 409
  - testing expressions in, 398
- XPath expressions,**
  - and styles, 329
  - building in Edit XPath Expression dialog, 397, 409
- XPath filter,**
  - on global templates, 215
- XPath filters on node-templates, 223**
- XPath functions,**
  - in XPath dialog, 397, 409
  - user-defined, 344
- XPath operators,**
  - in XPath dialog, 397, 409
- XPath version in SPS, 92**
- XQuery,**
  - Extension functions, 608
- XSLT,**
  - Extension functions, 608
  - inserting code fragment in design, 116
- XSLT import, 340**
- XSLT stylesheet preview,**
  - in Output Views, 28
- XSLT Templates, 38**
  - importing into SPS, 229
  - managing in Design Overview sidebar, 32
- XSLT to SPS, 340**
- XSLT transformations, 387**
- XSLT version,**
  - setting for SPS, 421
- XSLT version in SPS, 92**
- XSLTelements,**
  - inserting as code in design, 115